Experience with an Interactive Attribute-Based User Information Environment

Craig E. Wills
Dominic Giampaolo*
Michael Mackovitch*

Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609

WPI-CS-TR-94-2

Abstract

This paper explores an attribute-based approach to storing information in the context of a file system that supports extended attributes about files and a mechanism to manipulate files based on logical queries and comparisons of attributes. The novel aspects of our system are that it is sophisticated enough to operate as a user's primary method of interaction with the operating system and that it supports derived attributes whose values are derived at look-up time. Organizational mechanisms are explored to aid in the navigation of the system. We implemented our system as a user-level NFS server that supports attribute-based naming of files and other objects. We discuss its details, our experiences with it and performance comparisons between it and a traditional hierarchical file system.

*Current address: Silicon Graphics, Inc. Mountain View, CA.

1 Introduction

As an increasing amount of information reaches our computers the problem of managing information becomes more important. The traditional means for organizing information is to store it in individual files objects and place the files in directories of a hierarchical tree structure. Other information objects, such as processes or users often use a flat name space with access through type-specific utilities.

Either of these approaches works well with a limited amount of information, but breaks down as the information grows. When a user tries to maintain large numbers of files and directories, navigation becomes problematic. Deep hierarchies require long path names to identify file objects. Shallow or flat hierarchies make it difficult to identify individual objects because many unrelated objects may exist in the same directory. Further, once a user establishes a hierarchy, attempting to view a subset of the objects in a different organization becomes quite difficult. Hierarchical systems are rigid in that once established, it becomes an arduous task to reorganize the hierarchy.

As an example, if a user organizes files by project and then by subcomponent (such as source code, documentation, examples, etc), it is problematic to try and invert the hierarchy to view the files by component and then by project. Even though hierarchical systems may provide tools to locate files matching certain criteria, they do not allow the user to easily manipulate the group of files as an entity.

Rather than organize files in fixed directories, our work has explored attribute-based naming and manipulation of files. Attribute-based naming of files and other information works by associating attributes (name/value pairs) with the objects to be named and supporting queries about desired attributes. Each object has an arbitrary set of attributes, and users query the system to determine which objects match a given set of criteria about the attributes. In this manner, users declaratively state the set of objects desired instead of trying to locate specific items in a fixed hierarchy. An attribute-based naming system separates an item's location from how one accesses it.

Our prototype system, referred to as AttrFS, supports the capability to interactively work in a dynamically-updated, attribute-based file system, which is a significant departure from the traditional hierarchical interface to a file system and even other attribute-based work. Our work seeks to discover what benefits accrue and what issues arise in using such a system as the main mode of interaction with the computer. The novel aspects of our system are that it is sophisticated enough to operate as a user's primary method of interaction with the operating system and that it supports attributes whose values are derived at look-up time. The support of derived attributes makes it possible to integrate many types of volatile information into the name space in a clean fashion. A prototype implementation based on a user-level Network File System (NFS) server is used to test this approach. Although its use has primarily been for files it can and has been used for other types of objects found in a distributed environment such as machines, users and processes. Work on the system has also led us to explore mechanisms for better organization and user orientation in an attribute-based system.

The paper begins with an overview of the system followed by a demonstration of the system through examples. It continues with examples of organizational and navigational improvements in the system along with a description of the implementation. The performance of the system is compared with a traditional hierarchical file system. The paper concludes with a description of related work, our experiences and future work to be done on the project.

2 Overview

Initial work on the system concentrated on designing and building an attribute-based file system [Gia93]. The resulting system supports the following features:

- an arbitrary number of attributes per file,

- arbitrary sized attribute values of either ASCII or binary data,
- Full logical queries about files, including AND, OR, and comparison (=, ! =, <, >, <=, >=) operators as well as parenthesization to indicate precedence,
- comparisons of attributes that have their values derived dynamically at lookup time through internal or user-definable functions,
- interactive use and immediate update of changes made to files and attributes,
- a complete read/write file system supporting all standard file semantics, and
- integration with the Unix file system as a user-level NFS server.

There are several important features of our system. First, it is a full read/write file system, allowing users to "live" in an attribute-based name space. Previous attribute-based systems tended to be secondary interfaces, which supported only a browsing style of interface. Our work seeks to allow all interaction to take place inside the system. Second, derived attributes allow little-used and space-intensive attributes to be computed on demand. They also allow an easy integration path for many types of volatile and non-file information into a single name space—simplifying the conceptual model users must master to find and manipulate information. Third, the complete set of logical and comparative operators allows the creation of any view on the set of available objects in a straightforward manner. Finally, the clean integration with traditional file systems and tools allows immediate use. Together, these features provide a fully functional attribute-based file system for interactive use.

3 Usage

We constructed a prototype system of our system as an NFS server so that it could work with the Unix environment and existing commands. The following script of commands uses a standard shell to illustrate uses of the file system with `/attrfs` representing the mount point of an "attributized" branch of an existing UNIX file system hierarchy (typically at the granularity of a user's file system). The name space below `/attrfs` is interpreted by our modified NFS server. An initial set of attributes is constructed for files based on their position in the original hierarchy as well as heuristics about the file type.

```
1> cd /attrfs/Tag=papers/Tag=sysint
2> ls
main.aux        main.blg        paper.aux      paper.blg
main.bbl        main.tex        paper.bbl      paper.tex
...
3> la main.tex
    Name=main.tex
    Tag=sysint
    Tag=papers
    Type=Text
    Type=TeX Input
```

The first two commands create and show the contents of a view of all files with the tagged attributes `papers` and `sysint`. The '`/`' character represents the logical operator AND. Because the attribute name '`Tag`' is common, the use of of `Tag=` is optional. The order of the attributes is not important. The *la* command lists non-derived attributes for a given file.

```
4> cd /attrfs/src/proj1
5> ls
Makefile        main.c
func.c
6> vi hello.c
7> rm func.c
8> cc -o hello hello.c
9> ls
Makefile        hello.c
hello           main.c
10> la hello
    Name=hello
    Tag=src
    Tag=proj1
11> hello
hello world
```

Commands 4-9 illustrate creating another view of source files, creating a file in this view, removing a file and showing the revised contents of the view. Command 10 shows that when a file is created it inherits the attributes of the current view. Note that in command 10 type attributes are not automatically generated for the executable file. Ideally tools would be "attribute aware" and automatically generate attributes about files, such as type, in their normal course of processing. Additional attributes can be given at creation time or with the add attribute command *aa*. Similarly, the command *da* deletes attributes. Although there is no specific structure when attributes are created, later queries can utilize the attributes as though an explicit structure had been created. Modifications to a file's set of attributes cause the contents of the current view to change accordingly.

```
12> cd '/attrfs/(proj1,proj2,proj3)/Type!=C source'
```

This command creates a view of all non-C source files that have the tag `proj1`, `proj2` or `proj3`. The '`,`' indicates OR and parentheses indicate grouping. The capability to use disjunctive queries is a feature not present in previous work. Because the use of parentheses and other meta-characters conflicts with the shell, special characters must be quoted. We see this as a short-term problem of this prototype, which could be overcome with use of an alternate shell or more appropriately with graphical browsing and selection of attributes.

```
13> ls /attrfs/src/proj1/days_old=0
hello   hello.c
14> ls '/attrfs/src/proj1/Type=C source/mainfunc=true'
hello.c main.c
15> ls '/attrfs/src/proj1/Type=C source/linecount>100'
...
```

These three commands illustrate the use of derived attributes, which can be used to compute attribute values that change frequently or are accessed infrequently. Command 13 shows the use of a derived attribute that accesses the inherent age attribute of a file to show only newly created files. The `days_old` function is an internal procedure to our file system. The attribute `size` is handled in a similar manner. The other two commands illustrate the use of external Unix commands to derive attributes. These commands must be located in a special directory for protection. The boolean attribute `mainfunc` computes source files containing a main function. The numeric attribute `linecount` counts the number of lines in a file. Derived attributes incur run-time costs to compute, particularly external functions, but they require less space than pre-computing all attributes as done in other systems. External functions could also be dynamically loaded to reduce run-time costs.

4 Organization and Orientation

In some cases, orientation and abstraction were found to be difficult problems in using the prototype. As opposed to a hierarchy, users must not only explicitly specify attributes of files to be viewed, but also not to be viewed. Command 17 shows a partial solution to the problem of helping users be aware of what attributes can be specified. With the well-known name `ATTR:`, users can list to find what attributes are available in the current view and the number of files with each attribute. This feature is useful, but the contents of `ATTR:` are not always easy to compute and can degenerate into a search of many files.

```
16> cd /attrfs/proj1
17> ls ATTR:
Tag=RCS:9       Tag=src:27      Type=C source:3
Tag=proj1:40    Type=C header:1 Type=Text:7
```

Problems with organization and orientation within an attribute-based name space led to followup work on the system to explore better mechanisms [Mac94]. The principal result of this work was the development of a *view object*. This mechanism allows for an organization structure to be incorporated in the environment without losing the flexibility of an attribute-based system. View objects allow other types of objects, such as files, to be organized into groups so that only these groups and not their contents are shown when navigating the attribute-based name space. Thus the user does not see a large collection of objects, but rather a small set of groups of objects. The display of view objects does not affect the accessibility of objects, it only affects the image of the environment presented to the user during navigation and browsing.

These view objects have similarities to directories that are found in hierarchical name spaces, but have several important differences:

- objects are not physically located inside a view object,
- an object can appear in many view objects, and
- objects do not have to be manually placed inside view objects thus allowing objects to be grouped *after* they are created.

Examples of how view objects affect the display of objects are shown in the following. Names of view objects use a colon as the last character to distinguish them from other objects.

```
18> cd /attrfs/proj2
19> ls
INSTALLATION    Makefile        README          aa.c
attribute.c     attribute.h     attribute_mod.h block.c
...
20> mkview include: 'Type=C header'
21> mkview src: 'Tag=src'
22> mkview utils: 'Tag=utils'
23> ls
INSTALLATION    Makefile        README          include:
src:            utils:
24> la header:
    Name=header:
    Tag=proj2
    Type=C header
25> vi aa.c
```

These examples illustrate the creation of three view objects using the utility *mkview* (views can be deleted with *rmview*). These view objects collect the objects specified in the associated query for display during navigation. Just as with files, the created view objects inherit the attributes of the current view. However, as shown in command 25, the files within the view object can still be accessed directly, unlike the use of a directory. Similar to files, views and view objects can be created for other types of objects. View objects can also contain derived attributes, but the contents of these objects are created on demand. While affording no computational gain, view objects with derived attributes may provide organizational benefit.

5 Implementation

As previously indicated, our system is implemented as a modified user-level NFS server as shown in Figure 1. Operations on files in the `/attrfs` directory within the NFS work routines are passed to AttrFS routines on the server side. Because the prototype does not store the actual data of the file, operations that operate on the file contents are passed through to the local Unix kernel (from the level of the NFS work function). On the client side, most standard utilities work as is, including shells, editors and command line utilities. A shared library is used to avoid some interaction between the pathname queries that the system expects and processing done by the client kernel. This overloading of the NFS naming protocol is not ideal, but allows compatibility with existing tools. The AttrFS routines then treat the entire file name (after the `/attrfs`) as a query for processing.

The heart of the system implementation is a set of bitmaps that provide quick access to the contents of commonly used attributes. These bitmaps are initially created when the AttrFS is built and are updated as the contents of the file system change. These bitmaps are stored on disk and cached in memory as needed. The organization is shown in Figure 2 with the bitmap containing all files with the attribute `Tag=dict`. All files with the particular attribute are stored in the bitmap corresponding to their inode. This implementation limits the number of objects that may be stored in the prototype system, but experience shows it can still be

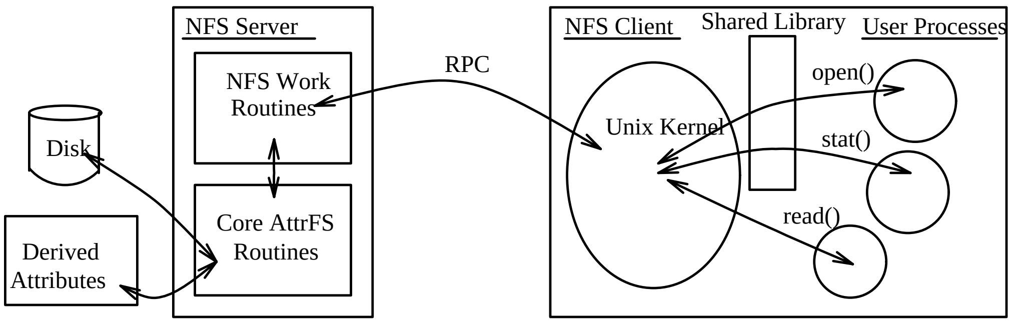

Figure 1: The Design of AttrFS

used for a large user file system. The actual size of the system can be varied at system creation.

In followup work the use of common attribute value bitmaps was extended to also create bitmaps for view objects, which provides better performance for accessing the contents of these objects. Updates to the view object bitmaps are handled in a similar manner as the common attribute value bitmaps, although the view object itself is an object and may be included in other views and view objects.

6 Performance

Table 1 shows the storage requirements for a file base of 5,093 files and 498 directories. As shown, the overhead incurred by our system for storing attributes and file-system structures is approximately 2.9 times the storage space of a traditional hierarchical file system. However, relative to the amount of data stored, our overhead is less than 5%. We consider this overhead to be acceptable.

Table 1: Storage Requirements

	File System Overhead	Data Storage	Total Storage
AttrFS	3.8 MB	76.3 MB	80.1 MB
SunOS4.1.1	1.3 MB	76.3 MB	77.6 MB

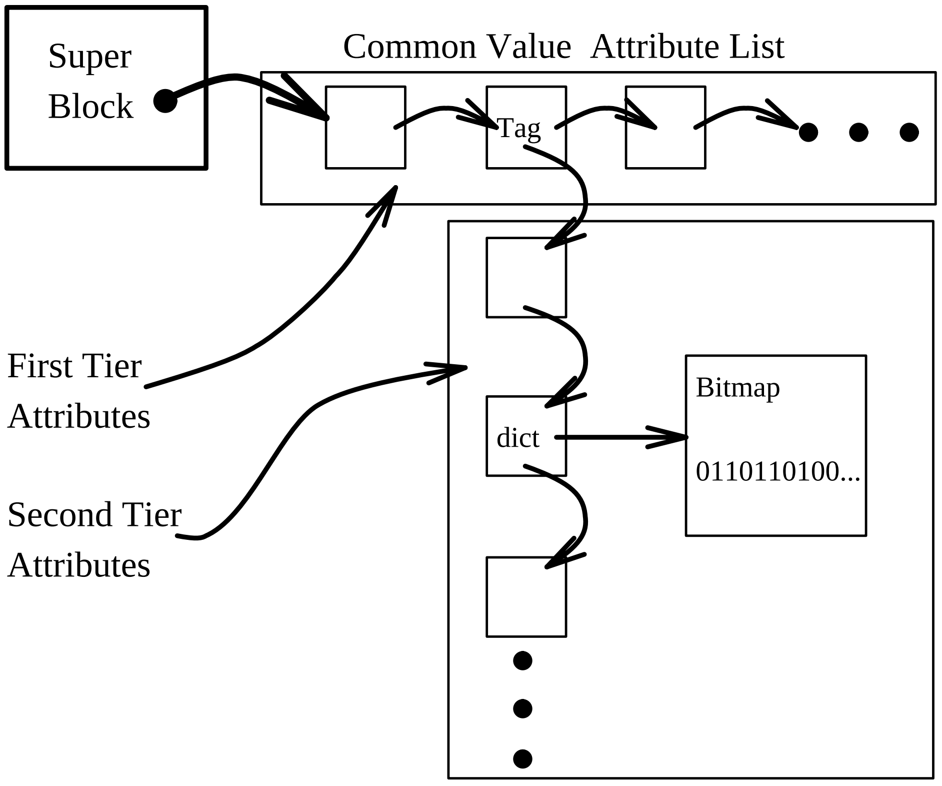

Figure 2: The Structure of Common Value Attributes in AttrFS

In terms of performance of operations, Table 2 shows the average performance of various file operations. The first test was the creation of views consisting of the conjunction of a number of tagged attributes with no view objects. This operation is comparable to the Unix system *chdir()* command. The second test was the creation of views consisting of the conjunction of tagged attributes with view objects. The third test was an extended query using the AND, OR and NOT operators for which no straightforward Unix operation exists. The final two test involved creation and deletion of a file. For all operations, the performance of our attribute-based file system is within an order of magnitude of the hierarchical file system. There is increased overhead for the addition of view objects in the system, which is reflected in the times.

We find these figures encouraging because they indicate that it is possible to implement a full attribute-based name space with performance acceptable for interactive use. A key to this good performance is the use of the bitmap structure for files, which allows logical operations to be performed quickly and the contents of common queries to be pre-defined with view objects.

Table 2: File Operation Performance

	AttrFS	Sun SPARC/SunOS4.1.1
create view (chdir with no view objects)	0.03 sec	0.002 sec
create view (chdir with view objects)	0.004 sec	–
extended query (chdir)	0.11 sec	–
create file (creat)	0.38 sec	0.05 sec
delete file (unlink)	0.22 sec	0.02 sec

7 Related Work

The integration of naming environments within a hierarchical environment has been explored in previous work such as Plan9 [PPT+93] and the Spring operating system [NR94]. The Plan9 work integrates objects using the existing the Unix operating system file space while the Spring work uses name objects, which are contexts in which objects can be placed. In each case the resulting name space is hierarchical.

Attribute-based naming has been used in such applications as directory lookup services for access to information contained in a directory or phone-book like database [BPY90, Neu89]. However these systems do not work on everyday information such as files.

More specific to our work are previous efforts at attribute-based naming of files [Mog86, GJSO91, Sec91, Ols93]. Mogul did early work on exploring the use of properties with files [Mog86]. Properties represent information about files, but do not store information based on derivations of the contents of the files.

The Semantic File System (SFS) interfaces to the rest of the computer as a regular, read-only file system, overlaying its extended semantics on top of standard operations [GJSO91]. The SFS processes path names as requests about files and uses the components of the path name to specify attributes. It determines which files match the criteria and creates a pseudo-directory containing the files that match. It only allows conjunctions of attributes. The SFS generates attributes for files by using "transducer" programs, which are similar to our external functions, but these operations are done offline, usually once per evening. This approach results in much storage being used and the chance of these pseudo-directories becoming out-of-date, which limits their use on information users are manipulating on a daily basis.

The work done by Sechrest in [Sec91] describes a simulation that implements a flexible attribute-based approach to naming of files. Their system supports attribute rules that define implicit attributes for a file. If the attributes of a file satisfy a rule then additional attributes are defined for the file. Other work by Sechrest [SM92] explores a hybrid approach that blends attribute-based naming and hierarchical naming so as to impose structure on a attribute-based system, although little experience is reported with actual use.

The InversionFS [Ols93] provides a set of file manipulation function calls that interface with the POSTGRES database [SK91]. Although the focus of the InversionFS is not specifically attribute-based naming of information, it is one of the features they support. Because file data are stored in a database, full logical queries about file attributes are possible. Although the function call interface to the system is analogous to the standard file I/O calls, they are different and programs must be specifically written to take advantage of them. In contrast, our system easily integrates and is used with traditional tools.

8 Experience and Future Work

Our use of the system has yielded much experience about the navigation and manipulation of files in the name space. The use of logical queries and derived attributes allow views to be formed that are not readily available in a hierarchical file system. Previously hidden files in obscure parts of the hierarchy were rediscovered when they appeared in queries that matched their attributes. On the other hand, other view specifications may require explicit negation of particular attributes to capture the desired set of viewed files. The addition of view objects helped with organization, but additional work needs to be done in refining their use.

The use of derived attributes is a powerful feature for dynamically adding attributes to the system and more easily adding new types of objects. One issue is the security involved in maintaining external functions.

The use of the NFS protocol was good for its integration with existing systems and utilities. However this decision caused problems with trying to fit attribute-based queries and concepts into the semantics of NFS operations, which were designed with a hierarchical system in mind. Another problem with this approach was conflicts with the syntax of existing command-line shells and of some utilities that tried to interpret file names themselves.

A point that obviously needs to be addressed is scaling the prototype beyond the set of file objects available to a user. The performance we obtained in the prototype is acceptable, but directly scaling the same system will not work for a large number of users. One approach that could be explored is to reorganize the name space into high-level contexts, which each could represent a user's file space or other system objects. Within each context, an attribute-based name space could be used. More work could also be done on the implementation itself.

The last issue in moving to an attribute-based name is the need to have better means of automatically assigning attributes to objects as well as using these attributes. Tools need to be more "attribute-aware" both in generating attributes for objects at creation time and using attributes of objects to automatically control processing.

9 Summary

The paper describes work on an attribute-based environment to help users more flexibly and uniformly organize their information. The significance of this work is that it is capable of interactive use, full logical queries and dynamically derived attribute values. Our system supports flexible specification of views and derived attributes while integrating with the rest of the operating system as an NFS server. Through the use of derived attributes, it is possible to incorporate non-traditional information into a single file system name space. Our experimentation with the system indicates that attribute-based naming is powerful, but that some of the features of hierarchical systems (such as levels of abstraction) are also desirable and need to be merged with attribute-based naming. We have made a step in this direction with the definition of view objects to capture a set of objects. Overall we feel the work thus far has been successful by demonstrating the feasibility of an attribute-based environment that can be a user's interface to the underlying system.

References

[BPY90] Mic Bowman, Larry L. Peterson, and Andrey Yeatts. Univers: An attribute-based name server. *Software–Practice and Experience*, 20(4):403–424, April 1990.

[Gia93] Dominic Giampaolo. CATFS—a content addressable, typed file system. Master's thesis, Computer Science Department, Worcester Polytechnic Institute, May 1993.

[GJSO91] David K. Gifford, Pierre Jouvelot, Mark A. Shelton, and James O'Toole Jr. Semantic file systems. *Operating Systems Review*, 25(5):15–25, October 1991.

[Mac94] Michael S. Mackovitch. Organization and extension of an attribute-based naming mechanism. Master's thesis, Computer Science Department, Worcester Polytechnic Institute, May 1994.

[Mog86] Jeffrey C. Mogul. *Representing Information about Files.* PhD thesis, Department of Computer Science, Stanford University, March 1986. Also available as Technical Report STAN-CS-86-1103.

[Neu89] Gerald W. Neufeld. Descriptive names in X.500. In *SIGCOMM '89 Symposium, Communications Architectures and Protocols*, pages 64–71, September 1989.

[NR94] Michael N. Nelson and Sanjay R. Radia. A uniform name service for spring's unix environment. In *Proceedings of the 1994 Winter USENIX Conference*, pages 201–209, January 1994.

[Ols93] Michael A. Olson. The design and implementation of the Inversion file system. In *Proceedings of the 1993 Winter USENIX Conference*, pages 205–217, January 1993.

[PPT$^+$93] Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey, and Phil Winterbottom. The use of name spaces in plan 9. *Operating Systems Review*, 27(2):72–76, April 1993.

[Sec91] Stuart Sechrest. Attribute-based naming of files. Technical Report CSE-TR-78-91, Department of Electrical Engineering and Computer Science, University of Michigan, January 1991.

[SK91] Michael Stonebraker and Greg Kemnitz. The POSTGRES next generation database management system. *Communications of the ACM*, 34(10):78–92, October 1991.

[SM92] Stuart Sechrest and Michael McClennen. Blending hierarchical and attribute-based file naming. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 572–580, June 1992.