A File Naming Scheme using Hierarchical-Keywords

Harumasa Tada, Osamu Honda
Graduate School of Engineering Science, Osaka University
1-3, Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan

Masahiro Higuchi
School of Science and Engineering, Kinki University
3-4-1, Kowakae, Higashi-Osaka, Osaka 577-8502, Japan

Abstract

In this paper, we propose a file naming scheme, called HK (Hierarchical-Keyword-based) naming. In file systems, hierarchical naming has been used for these several decades. As the number of files stored in file systems increases, the weakness of hierarchical naming is getting recognized. Some researchers have proposed hybrid naming schemes which introduce attribute-based naming into hierarchical naming. However, they are unsatisfactory or too complicated. HK naming is a hybrid naming scheme which integrates hierarchical naming and attribute-based naming in a simple way. In HK naming, a file is named by a set of keywords and keywords are organized hierarchically. We point out the usefulness of HK naming by showing that it can solve problems which occur in practical usage of large file systems.

1 Introduction

Naming plays an important part in file organizing. With the advance of secondary storage devices, the number of files stored in a file system is rapidly increasing. It becomes a problem how to find a necessary file among a vast amount of files. On the other hand, files which contain non-text data, such as images or sounds, are becoming popular. For such files, the content-based search, e.g. the search with the grep command in UNIX, is difficult or even impossible. Nowadays, organizing files by names is getting more important.

Hierarchical naming[1] is used in traditional file systems. It reflects the hierarchical structure of directories. A user specifies a file by an explicit path name, i.e. an absolute path name or a relative path name from the current directory. The chief merit of hierarchical naming is in its familiarity and ease of implementation. On the other hand, it is insufficient to organize a large number of objects. There are a number of works based on this observation[1, 2, 3, 4, 5, 6, 7].

Some of them[2, 6] proposed different naming schemes which are classified into attribute-based naming[1]. In attribute-based naming, each object is associated with a set of attributes instead of a path name and a user retrieves objects using queries.

Though some researchers tried to introduce the merit of attribute-based naming into existing file systems[1, 4, 7], their naming schemes do not get popularity so far. Their approaches are hybrid naming schemes between hierarchical naming and attribute-based naming. In these approaches, integration of two naming schemes is rather awkward and resulting naming schemes are not necessarily easy to use.
We consider that the problem lies in the concept of file containers as the metaphor for directories. A directory is actually a special file which contains information of other files. In hierarchical naming, directories are regarded as containers of files—they are represented as folders in most GUI systems. The problem of previous approaches is that they tried to introduce attributes while they preserve directories as file containers. We consider that the concept of file containers conflicts with file attributes and their integration results in mere composition of orthogonal naming schemes.

We discarded the concept of file containers and propose a new naming scheme. The main components of our naming scheme are keywords. Instead of putting files in directories, we attach keywords to files. Hierarchical structure is used for organizing keywords, not files. We call our naming scheme HK (Hierarchical-Keyword-based) naming. In HK naming, a file is given a filename and a set of keywords. A user specifies files by an arbitrary ordered, possibly incomplete, list of keywords. A keyword is a string of characters such as \\
//image/photo/ and //article/sports/baseball/. Since the set of keywords forms a keyword tree like a directory tree, such keywords are called hierarchical keywords. A user can find a keyword traversing the keyword tree.

The rest of this paper is organized as follows. In section 2, we explain two existing naming schemes and review previously proposed hybrid naming schemes. Our naming scheme is presented in section 3. In section 4, we show the usefulness of HK naming in some practical usages. The conclusion appears in section 5.

2 Existing Naming Schemes

2.1 Naming Schemes

A name is a string of characters which is used to specify a file or a set of files. In usual UNIX file systems, for example, a name is an absolute path name or a relative path name.

A filename is a string of characters which is assigned to a file so that users can identify it. For example, main.tex and fig.jpg are typical filenames. In this paper, we distinguish between names and filenames.

A naming scheme consists of syntactic representation and semantic interpretation of names. A set of names complying with a naming scheme forms a name space[8].

Hierarchical naming[1] is a naming scheme based on hierarchical structure. It reflects the implementation of traditional file systems. In traditional file systems, files are organized by putting them in hierarchically structured directories. Each file (also a directory) is assigned a filename which is unique in the directory which contains it. A name of the file is a path name which is a sequence of filenames. A path name describes where the file is located in the directory hierarchy. Such names form a hierarchical name space.

Attribute-based naming[1] is a naming scheme in which a name is a set of attributes. In attribute-based naming, each file is associated with some attributes. Each attribute is a pair of an attribute-name and a value such as “author : tada”. A user retrieves files using queries such as “author = tada & type = text”. Since each attribute in names determines a set of files, such names form a set-structured name space.

Keyword-based naming is the basis of our naming scheme. It is a variant of attribute-based naming. Each object is associated with several keywords. A keyword is an attribute of the object but it does not have an attribute-name. A name is a list of keywords such as “tada, text, report”. Keyword-based naming inherits most advantages of attribute-based naming.
2.2 Hybrid Naming Schemes

Since hierarchical naming and attribute-based naming have their own advantages\(^1\), some researchers have tried to integrate attribute-based naming with hierarchical naming.

### 2.2.1 Semantic File System

Semantic File System (SFS)\(^4\) extends usual tree-structured file systems to provide attribute-based access to files. In SFS, files are stored in usual tree-structured directories. Moreover, they have some attributes and can be accessed by queries. Attributes are automatically extracted from files by user programmable \textit{transducers}.

The main feature of SFS is that it enables attribute-based access through native directory commands such as \texttt{ls} and \texttt{cd}. SFS provides syntactic compatibility with existing path names by introducing the concept of a \textit{virtual directory}. Path names of virtual directories are interpreted as queries. For example, a query such as “\((\text{author} = \text{tada}) \land (\text{category} = \text{paper})\)” is represented by the path name like \texttt{/author:/tada/category:/paper}. Queries of SFS are limited to conjunctions of atomic expressions in the form “\textit{attribute} = \textit{value}”.

The drawback of SFS is that it simply provides two orthogonal name spaces. Each file characteristic should be represented in either a hierarchical name space or a set-structured name space. For example, consider a file which contains a photograph of a mouse on a table. Now we note the characteristic “mouse”. If “mouse” is represented as a directory name such as \texttt{/photo/animal/mouse/furniture/table}, some fixed order is imposed on “animal” and “furniture” which are semantically independent. If “mouse” is represented as an attribute such as “\textit{subject} : \textit{mouse}”, we cannot distinguish between a small rodent and a popular pointing device. Both ways are unsatisfactory.

### 2.2.2 Prospero File System

Prospero File System\(^7\), like SFS, also provides a composition of two orthogonal name spaces. The difference is that Prospero allows each user to have his or her own hierarchical name space which is called a \textit{virtual file system}. In a virtual file system, a directory contains a collection of links. A link maps a single component of a name to a file or a directory. Each link may have an associated function, called a \textit{filter}, which yields a virtual directory. A filter corresponds to a query in SFS. However, a filter is an arbitrary program written by users. It enables more flexible construction of virtual directories than SFS whose queries are conjunctions of atomic expressions. Prospero provides more flexibility than SFS at the cost of writing filters.

### 2.2.3 Multi-structured Naming

Sechrest et al. tried to integrate attribute-based naming with hierarchical naming using rule-based framework in their multi-structured naming\(^1\).

Multi-structured naming is based on the thought that each component name in a path name is an attribute attached to the file with a set of constraints upon their use in names. For example, one of such constraints

\(^1\)The advantages and disadvantages of these two naming schemes are discussed in our previous paper\(^9\).
The uniqueness of multi-structured naming is that a user can express these constraints as explicit rules. In hierarchical naming, these constraints are implicit in the hierarchical structure of a name space.

The rules of multi-structured naming are classified into scope rules, implicit value rules, aliasing rules, and so forth. We mention a scope rule as an example. Consider the name hierarchy shown in figure 1. We can give a scope rule which specifies that male and child do not introduce a new naming context. In this case, the names /photo/portrait/male/child/cry.jpg and /photo/portrait/child/male/cry.jpg refer to the same file.

Multi-structured naming provides enough flexibility. Using appropriate rules, users can define the structure of a name space without restriction of a hierarchy. However, the details of the way to write such rules is not described in [1].

The drawback of multi-structured naming is that it is too complicated for users to use. In multi-structured naming, the set of names forms a kind of name tree as usual hierarchical naming. Unlike hierarchical naming, however, constraints of hierarchical structure of the name tree can be arbitrarily relaxed by users. It confuses those who regard the name tree as a usual directory tree. Defining appropriate rules for each attribute is also difficult for users.

In [1], only the framework is described and the user interface is not mentioned. We consider that the user interface is an important and difficult issue of multi-structured naming.

3 HK Naming

As mentioned in section 2.2, existing hybrid naming schemes have shortcomings. All of them cling the notion that directories, whether real or virtual, are containers of files. However, it is not reasonable to use nested file containers and attributes together. These two concepts are difficult to combine because they are orthogonal, that is, file containers are physical objects whereas attributes are logical entities. To solve this problem, we consider that the concept of file containers should be discarded.

We propose another naming scheme called HK (Hierarchical-Keyword-based) naming. We chose keyword-based naming as the basis of our naming scheme because it provides great flexibility in finding and organizing objects. Nevertheless, we can point out two drawbacks of keyword-based naming. First, keywords are not organized, e.g. we cannot distinguish between the dining “table” and the multiplication “table”. Second, keyword navigation is not provided. If we cannot recall what keywords are attached to the file which we needed, there is no way to navigate us to such keywords. In order to find the file, we have to list all possible keywords and browse them. As we know from experience with hierarchical naming, hierarchical structure
provides organizing and navigational features. This is why we introduced hierarchical structure to keywords.

In the following sections, we give the notation of names in HK naming. Note that the notation described in this paper is not absolute one. It may change according to environment in which HK naming is used. The notation given in this paper is determined taking account of compatibility with UNIX path names.

### 3.1 Hierarchical Keywords

In HK naming, a file is given a filename and attached some hierarchical keywords (abbreviated as HK). A HK consists of one or more component keywords $k_1, k_2, \ldots, k_n$ and is denoted as $//k_1/k_2/\cdots/k_n//$. Each component keyword is a nonnull string of characters. The order of component keywords in a HK is meaningful, e.g. $//people/child//child/people//$. The parent-child relation may exist between two HKs. $//k_1/k_2/\cdots/k_n//_1$ is a parent of $//k_1/k_2/\cdots/k_n//$ and $//k_1/k_2/\cdots/k_n//$ is a child of $//k_1/k_2/\cdots/k_{n-1}//$. The ancestor-descendant relation is the transitive-reflexive closure of the parent-child relation. The set of HKs forms a hierarchical structure which we call the keyword tree.

### 3.2 Names

A name in HK naming is written using some HKs and a filename$^2$. A name is a comma-separated list of HKs, possibly followed by a filename without separators. For example, “//child//animal/dog/fig.jpg” is a name which consists of a list of two HKs and filename fig.jpg. Note that the name “//animal/dog//” includes a HK //animal/dog/ while “//animal/dog//” includes a HK //animal and a filename dog. In names, the order of HKs is meaningless. For example, two names “//child//animal/dog/fig.jpg” and “//animal/dog//,//child/fig.jpg” have the same meaning.

A set of files specified with a name $N$ contains all files that $N$ matches. Matching between files and names is defined as follows. A HK covers all descendants of it. For example, //animal/ covers //animal/, //animal/cat/, //animal/dog/poodle/, etc. A name $N$ matches a file $F$ if $F$ satisfies the following conditions.

1. Every HK in $N$ covers at least one of HKs attached to $F$.
2. If $N$ includes a filename, it is the same as $F$’s filename.

If $N$ does not include a filename, $N$ matches all files which satisfy the condition 1.

A full name of a file is a name which consists of a filename and all HKs attached to the file. Since HKs can be listed in any order, the full name of a file is not unique.

### 3.3 Filename-sensitive Hierarchical Keywords

According to definitions in section 3.2, a full name of a file may specify multiple files. For example, consider three files whose full names are “//sports/fig.jpg”, “//child//sports/baseball/fig.jpg”

$^2$Recall that names and filenames are different in this paper (see Section 2.1).
and “//sports/./woman/fig.jpg” respectively. Since “//sports/fig.jpg” matches all of them, the full name of the first file specifies three files. In this case, there is no way to specify the first file exclusively.

In hierarchical naming, every file has an absolute path name which specifies the file exclusively. It would be a serious drawback of HK naming that some files cannot be specified exclusively.

Hence we introduce special HKs called filename-sensitive HKs (abbreviated as FSHK) which enables exclusive file specification in HK naming. Though a normal HK begins with double slashes (//), a FSHK is written with a single slash at the head, such as /bin/.

A FSHK works as normal HKs except following limitations.

1. It cannot be attached to two or more files which have the same filename. For example, two files whose full names are “/image/./animal/dog/photo1.jpg” and “/image/./people/child/photo1.jpg” cannot coexist.

2. It does not cover its descendants. For example, a name “//tool/mail/./bin/mycom.pl” does not match the file whose full name is “//tool/mail/./bin/script/mycom.pl” because /bin/ does not cover /bin/script/.

These limitations guarantee that any name including a FSHK and a filename does not match two or more files. Thus a user can specify files exclusively using FSHKs.

In order to make sure to be specified exclusively, every file should be attached a FSHK. It is possible that a file has two or more FSHKs.

4 Practical Usages

Hierarchical naming works well if the structure of the name space is carefully designed. In the cases of applications’ source trees and UNIX system files, types of files stored in a directory tree are quite limited. In these cases, it is possible to design an appropriate directory hierarchy to store them.

On the other hand, attribute-based naming is suitable for the growing set of files, e.g. user files stored in home directories. Since it is impossible to know beforehand what kinds of files to come, name structure should be constructed in an ad hoc manner. In such cases, flexibility of attribute-based naming becomes valuable.

In this section, we mention some of such cases as examples. By these examples, we show that HK naming can solve problems which occur in practical usage of large file systems. Some of the following problems can also be solved in hybrid naming schemes mentioned in section 2.2. However, the solution of HK naming is simpler than that of multi-structured naming and more flexible than those of SFS and Prospero.

4.1 Photographs

Photographs are difficult to organize using hierarchical directories. We want to organize them according to many attributes such as subjects, places or seasons. However, all of these attributes are hard to express hi-
erarchically. Long path names like `/home/tada/photo/1997/August/Canada/lake/Louise/canoe/ca.jpg` are inconvenient in searching files. Currently, several image database applications, which help to organize photograph files, are on the market, for example, Thumbs Plus[10], ACDSee[11] and Media Center Plus[12]. Keywords are powerful tools for photograph organizing. We can assign to each photograph as many keywords as we want. Most of image databases support keywords in addition to hierarchical folders.

In HK naming, keyword-based organizing is supported by the operating system. Thus any application can specify files with keyword lists. On the other hand, keywords assigned in image databases can be used only in them. Moreover, image databases manage keywords and folders separately. This involves the same problem as that of SFS mentioned in section 2.2.1.

### 4.2 Shared Papers

Nowadays we can download many papers in PS and PDF formats via WWW. When a researcher downloaded a paper which concerns his or her research interest, he or she usually wants to share it among members of his or her research group. Thus the papers are stored in the shared disk space. Other members also store papers in the same disk space.

The names of the stored files should represent some attributes of the paper such as subject, author, year published and so forth. In hierarchical naming, even though they are semantically independent, they should be listed in fixed order like `/share/paper/OS/file-system/1994/Bowman/Bow94.ps`.

To keep consistency of the directory structure, all members who store papers should have a consensus about the order of attributes in names. However, this consensus is difficult to build, especially if papers concern two or more subjects.

In HK naming, independent attributes can be expressed by separate HKs like “/paper/, /year/1994/, /subject/software/OS/file-system/, /author/Bowman/Bow94.ps”. Hence the keyword hierarchy can be constructed based on logical inclusion of attributes. Thus it is easy to achieve a consensus among members. Moreover, HK naming enables hierarchical classification of attributes such as /subject/software/OS/file-system which is impossible in SFS and Prospero.

### 4.3 Scripts

In UNIX systems, we often write scripts using shells or other script languages like Perl[13]. To execute such a user script as a command, the script should be placed in a directory included in the command search path (the shell variable `PATH`).

The problem is that we often forget the name of the script which we wrote. One reason is that user scripts are not used so often as ordinary UNIX commands. Another reason is that we do not organize the scripts we wrote. If we put a script in a subdirectory such as `$HOME/bin/tool/preview`, we should add the directory to the command search path. Thus we usually put all scripts in a fixed directory such as `$HOME/bin`. As time goes by, the directory contains a lot of such unorganized scripts.

In HK naming, we can classify scripts by attaching HKs such as “/script/, /tool/build/, /tool/preview/, /document/latex/”. Including `/script/` in the command search path, shells can find all scripts that are attached `/script/`. Note that `/script/` is a FSHK, which prevents shells from finding two or more com-
mands with the same name. In other words, /script/ guarantees the uniqueness of command names.

4.4 Local Files referred in WWW Pages

Most of HTML documents in WWW sites refer several local files. Such local files may be image files used for illustrations, backgrounds and icons, or they may be CGI scripts, sound files and so forth.

Moving such files to subdirectories causes a trouble in hierarchical naming. The reason is that HTML documents refer local files with their path names. For example, consider a WWW site which uses background images in /home/tada/image. As other image files in /home/tada/image increases, the webmaster moved background images into /home/tada/image/background. At this point, all references to these images in HTML documents become invalid. To recover the background of WWW pages, all HTML documents should be modified.

In HK naming, such a trouble does not occur. In the above example, if an image file back1.jpg is attached a FSHK /image/, it can be always referred with /image/back1.jpg. When detailed organization is required, more HKs can freely be attached to back1.jpg. Unless /image/ is detached, it is guaranteed that the name /image/back1.jpg is still valid. In this way, FSHKs provide a permanent way to specify files.

4.5 Mail Threads

Generally, mails are classified according to values of fields such as From:, To:, Subject: and so forth. Usual mail agents classify mails by putting them into folders which correspond to directories. If mails are stored in separate folders according to values of From: fields, it is hard to follow a mail thread, i.e. a sequence of related mails which are linked using References: field.

In HK naming, mails are not separated by HKs. Even if different HKs are attached to mails, it is always possible to list all mails. A user can classify mails by senders by attaching HKs like //mail/From:/higuchi/. Then he or she can search a mail by sender and read the following thread.

5 Concluding Remarks

Some researchers have proposed hybrid naming schemes between hierarchical naming and attribute-based naming. However, they are less flexible or difficult to use because two conflicting concepts, file containers and file attributes, coexist in them. To solve the problem, we introduced the concept of hierarchical keywords instead of file containers and proposed HK naming. We showed that HK naming can solve some problems which occur in practical file usage.

One of our future work is to study the efficient implementation of HK naming. Performance is an important and difficult issue in HK naming. HK naming is costly for the following reasons.

1. It is inherently heavier than hierarchical naming. This is because HK naming involves dynamic creation of file sets while hierarchical naming requires only directory lookups.
2. It encourages users to attach many hierarchical keywords to files. This causes increases in both the number of HKs and the number of files which matches each HK.

3. It may involve handling of large sets of files. Since a HK covers its all descendants, top level HKs may match large amount of files.

In order to make HK naming practical, the efficient implementation is essential. The studies of data structure, disk allocation, indexing of HKs, caching of name resolution results, and others are required.

References


