Structure-free Name Management for Evolving Distributed Environments

Douglas B. Terry

Computer Science Laboratory
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

Abstract: Name services facilitate sharing in distributed environments by allowing objects to be named unambiguously and maintaining a set of application-defined attributes for each named object. Existing distributed name services, which manage names based on their syntactic structure, may lack the flexibility needed by large, diverse, and evolving computing communities. A new approach, *structure-free name management*, separates three activities: choosing names, selecting the storage sites for object attributes, and resolving an object's name to its attributes. Administrative entities apportion the responsibility for managing various names, while the name service's information needed to locate an object's attributes can be independently reconfigured to improve performance or meet changing demands.

CR Categories and Subject Descriptors: C.2.4 **[Computer-Communication Networks]:** Distributed Systems; D.4.3 **[Operating Systems]:** File Systems Management - *Directory structures*; H.2.4 **[Database Management]:** Systems - *Distributed systems*; H.2.7 **[Database Management]:** Database Administration - *Data dictionary/directory*;

Additional Keywords and Phrases: naming, binding, name servers, distributed name management, structure-free name resolution.

Structure-free Name Management for Evolving Distributed Environments

Abstract: Name services facilitate sharing in distributed environments by allowing objects to be named unambiguously and maintaining a set of application-defined attributes for each named object. Existing distributed name services, which manage names based on their syntactic structure, may lack the flexibility needed by large, diverse, and evolving computing communities. A new approach, *structure-free name management*, separates three activities: choosing names, selecting the storage sites for object attributes, and resolving an object's name to its attributes. Administrative entities apportion the responsibility for managing various names, while the name service's information needed to locate an object's attributes can be independently reconfigured to improve performance or meet changing demands.

1. Introduction

The ability to name objects in an evolving distributed computing environment requires flexible techniques for managing the set of object names. Managing names involves maintaining information about named objects, hereafter referred to as *attributes*, and providing facilities for accessing this information. Often in distributed systems, a name space is cooperatively managed by a collection of active entities, called *name servers*.

Early name servers simply mapped host names to network addresses. Contemporary name services, such as Grapevine [Birrell *et al.* 82], Clearinghouse [Oppen and Dalal 83], or the Domain Naming System [Mockapetris 83], manage various types of information about various types of objects. Even file directory systems and database catalog managers [Lindsay 80] can be considered name servers for specific classes of objects.

The database facilities needed by individual name servers to store object attributes represent only one portion of the mechanisms present in a distributed name service for large computing environments with several participating organizations [Terry 85]. This paper examines three principal activities that require interactions between different name servers and between servers and their clients; these activities are central to distributed name management and have received much less attention than database management techniques:

> <u>Assigning names to objects</u> should be performed, or at least controlled, by the creators of the objects. Users of a name service attach mnemonics to names unbeknownst to the servers managing those names. The difficulty lies in naming large numbers of objects when many autonomous organizations are involved in creating objects.
>
> <u>Selecting name servers to manage a named object</u>, called the *naming authorities* for the object, should also be under the control of the object's owner. For large computing environments, not all name servers can be authorities for all named objects; the authority for objects must be divided among servers according to administrative concerns.
>
> <u>Locating a named object</u> or information about the object given only its name becomes a problem if names are distributed among name servers. *Name resolution* denotes the process

of determining the naming authorities for a named object. Once the authorities are discovered, standard database operations can be invoked to read or update the named object's attributes.

The emphasis in this paper is on separating these three activities: choosing names, selecting the storage sites for object attributes, and resolving an object's name. Being able to distribute names among name servers and resolve names independent of what structure those names may possess permits name services to be easily reconfigured in response to changes in the computing environment; clients of the name service need not be aware of such reconfigurations since the name space remains the same.

The next section examines current name management techniques and explores how they may fail to adapt to an evolving distributed environment. The second half of this paper describes a new approach to name management that achieves the desired flexibility: *structure-free name management*.

2. Towards Flexible Name Management

2.1. Traditional approaches to managing names

In existing distributed name services that do not rely on broadcast for locating names, the assignment of authority for parts of the name space, as well as the mechanisms for resolving object names, depends heavily on the name structure. Most current name spaces are tree-structured in which each edge of the tree has a label. Conceptually, information about objects resides in the leafs of the tree; an object's name simply consists of the labels along a path from the root of the tree to a leaf node. For instance, the Grapevine system manages a naming tree of constant depth two [Birrell *et al.* 82]. The Domain Naming Convention, on the other hand, allows names to have arbitrarily many labels [Su and Postel 82].

Current name services distribute the authority for names to various servers based on the structure and contents of the name; syntactically similar names, for some similarity criteria, have the same authorities. For example, in the Grapevine system, all of the names belonging to a particular *registry* have the same naming authorities. In the Domain Naming System, the name tree is divided into subtrees called *zones*; a name's zone determines its authorities [Mockapetris 83]. These naming schemes provide less than perfect administrative control over the placement of an object's attributes. Because of the syntactic assignment of naming authorities, the choice of a name for a new object is partially governed by an organization's concerns for the name servers that store the object's attributes. Changing an object's name servers requires changing its name or assigning new name servers for all objects in the same syntactic partition of the name space.

Name resolution proceeds by following a path through the name tree starting at its root, that is, the individual labels of a name are resolved in succession. The amount of information needed in name servers to resolve names at the various levels of the hierarchy is proportional to the degree of branching of the name space tree. For instance, all Grapevine servers must know the storage sites for every registry. The cost of resolving names is dictated, to a large extent, by the depth of the name tree.

Unfortunately, existing name services' reliance on syntactic structure in order to locate an object's attributes place undue constraints on the management of the name space. The inability to adapt to growing communities with changing requirements is the main deficiency of traditional name management techniques. Name services should be able to be reconfigured if the present servers become overworked or overburdened with data. With current services, reconfiguration occasionally requires objects to change their names because the name space is distributed among servers according to syntactic partitions.

2.2. Some scenarios

The lack of adaptability in existing name services can be evidenced in a few simple scenarios. The following problems can be alleviated through a more flexible approach to name management, such as the one proposed later in this paper.

scenario #1: splitting Grapevine registries

Current problems of scale in the Grapevine system exemplify a lack of flexibility. Some of Grapevine's registries are becoming quite large. Suppose that a particular registry grows too large to be feasibly managed as a single entity; what can be done?

With the current Grapevine mechanisms, as a registry grows over time, no provisions can be made for dividing its data between different name servers. The only solution is to split the registry into two separate registries. Some or all members of the registry must change their names, a costly operation. Similar problems arise if organizations divide and subsequently want separate naming authorities. At least one Grapevine registry has already been split, causing some of its members to be renamed.

scenario #2: adding more structure to names

The Grapevine designers argue that more labels can be added to a name to handle larger name spaces [Schroeder *et al.* 84]. For instance, Grapevine names were expanded to three parts in the design of the Clearinghouse [Oppen and Dalal 83]. Unfortunately, for well-established systems, adding more levels to the name space forces all objects to change their names.

Designing a system to manage large numbers of names is not particularly difficult; the major difficulty is trying to predict how big a name space will eventually become. At best, the system designers must carefully structure the name space according to the projected growth of the environment so that no indivisible set of names becomes unmanageably large.

Suppose the designers of a new name service, with incredible foresight, give names a lot of structure so that the name tree can handle substantial growth without becoming exceedingly broad. The name space will initially be quite sparse until it fills out. Nevertheless, name service clients must pay a performance penalty if names are resolved a label at a time, as in current systems; the cost of name resolution is a function of the size of names, not the size of the name space.

scenario #3: merging existing name spaces

Occasionally, existing distributed environments that have developed in isolation form interconnections between them so that information can be shared. Suppose two internets merge with each other; how can the existing name spaces must be merged together to get a combined name space? Two things make this well-known problem difficult. First, two different objects may have been created before the merge with identical names. Second, the different systems will likely have different conventions for naming objects.

If name conflicts do not exist, the second problem can be avoided by separating name management from the structure of names. A uniform name management policy can be adopted for the combined name space even though names may be of non-uniform structure.

2.3. Adapting to environmental changes

The previous scenarios indicate that effectively managing a distributed name space is complicated by the fact that internet computing environments are constantly evolving in various ways: new computing resources, software systems, and users frequently join the environment; the usage patterns of resources change over time as new liasons form between members of the environment and their work behavior fluctuates; networks occasionally interconnect with other networks forming larger internets. Trying to predict the changes that will occur within an environment during an object's lifetime is difficult.

Due to the unpredictability of an evolving system, a name service should separate how an object's name is managed from the name assigned when the object is created. In this way, name management can adapt to changes in the computing environment (the name service can be reconfigured), while object names remain immutable. Changing the name of an object is an expensive operation since all of the references to the named object become invalid.

If the selection of naming authorities for objects is done independent of the objects' names, then some of an overloaded name server's responsibilities may be offloaded to a lightly loaded server, a server may be upgraded to a different processor or get more disk space allowing it to store a larger part of the name space, additional servers may be introduced, and so on.

Also, it is desirable to separate the assignment of authority for the name space from the resolution of object names. In this way, the name resolution process can be dynamically adjusted to tune the performance of name service operations without impacting the authority assignments. Moreover, the distribution of object attributes among servers is likely dictated by administrative concerns for the integrity and privacy of information about objects, whereas the information needed to resolve names is used solely within the name service.

3. Structure-free Name Management

3.1. Functions

Distributed name services generally provide many operations for manipulating the set of attributes for a named object. The Clearinghouse client interface, providing dozens of operations, is a good example of the range of operations that might be desired [Oppen and Dalal 83]. This paper is not concerned with such database operations; rather, this section introduces functions for the three activities previously outlined.

Register[name] $\rightarrow$ {OK, AlreadyUsed}

A name service client presents a name to be registered. The name service returns OK if the name has not been previously registered; if the name is currently used for a different object then AlreadyUsed is returned. Names should be globally unambiguous so that they can be freely passed between processes at different sites, and names should be locationindependent so that their referents can migrate between sites without having to change their names. In a widely distributed environment, names may have structure agreed upon by clients to reduce the chance of independently generating conflicting names. The name service does not enforce this structure. When a name is registered with the name service, a default set of naming authorities may be assigned.

AssignAuthorities[name, servers] $\rightarrow$ OK

A client can specify the naming authorities, or *authoritative name servers*, for a given object. If attributes are currently being maintained for the named object, then the request represents a reassignment of authority; the object's attributes must be transferred to the new authorities. (Techniques for synchronizing updates to replicated data have been extensively studied in the literature [Ellis 77][Gifford 79], and are not addressed in this paper.) Typically, the creator of an object, wishing to choose the authorities for the object, calls **AssignAuthorities** immediately after registering the object. In this case, no updates to the object attribute database are required. Allowing organizations control over the storage sites for information about their objects, ensures that they do not have to give up their autonomy when they agree to participate in a shared global name space. In practice, only certain clients may be authorized to assign authorities for an object.

Resolve[name] $\rightarrow$ servers

A name can be resolved to a list of its authoritative servers by presenting the name to any name server. When the naming authorities are returned, either directly to a client or to a name server servicing a client request, one or more authorities can be contacted to perform an operation on the name service attribute database. Thus, the process of resolving a name remains separated from the database activity that manages object attributes.

The following sections describe the mechanics of *structure-free name management* that permit names to be resolved independent of the name structure and choice of naming authorities.

3.2. Authority attributes

Structure-free name distribution places no restrictions on the administrative control over parts of the name space. In particular, the owner of an object may choose its naming authorities, subject to administrative constraints, independent of the object's name. This permits maximum flexibility in the administrative assignment (and reassignment) of authority.

To determine the authoritative name servers for every named object, the name service maintains *authority attributes* that contain lists of the authoritative name servers for every object. Essentially, an object's naming authorities are attributes of that object used solely by the name service. These attributes that are internal to the name service comprise the *configuration database.*

If all name servers store the complete set of authority attributes, name resolution involves a single query on the local configuration database. However, for large numbers of objects, the configuration database is undoubtedly too cumbersome to be stored everywhere in its entirety. The configuration database itself must be distributed so that no server needs complete knowledge of the authorities for all named objects. The primary difficulty in resolving a name then lies in locating the authority attribute for an object. Several interactions between servers may be required as the name resolution activity migrates from one name server to a potentially more knowledgeable server until the set of authoritative servers is determined.

3.3. Contexts

The notion of a context has proven useful for partitioning a name space into smaller components [Saltzer 78]. Often, contexts represent a division of the name space along natural geographical, organizational, or functional boundaries. This paper adopts a more concrete working definition: a *context* is a collection of authority attributes. A name is said to exist in a context if and only if the authority attribute for the name is stored in the context.

For the purpose of name management, contexts provide a means of partitioning the configuration database so that it may be distributed among servers. Contexts represent indivisible units for storage and replication of authority attributes. A given context may be maintained at any collection of name servers, and a given name server may store any number of contexts. Means must exist for identifying contexts, but the choice of particular names for contexts is not important since context names are only used internally within the name service.

Since contexts do not contain information about objects provided by clients, the decomposition of the configuration database into contexts and the choice of authorities for those contexts can be done to facilitate name resolution, rather than being governed by administrative desires. Ideally, names should be assigned to contexts such that checking whether a given name exists in a particular context is a simple operation and can be performed without having a copy of the context.

3.4. Clustering conditions

A *clustering condition* is an expression that allows the name space to be conveniently partitioned into contexts. Specifically, a clustering condition applied to a name yields either a TRUE or FALSE value. Any procedure that exhibits this behavior might serve as a clustering condition. The particular value returned, TRUE or FALSE, indicates whether or not the given name exists in the particular context.

Names can be clustered *algorithmically* according to the value that results from applying a function to them. In this case, the clustering condition is of the form ‘‘function(name) = value’’. For instance, a hash function is a well-known technique for clustering names into buckets.

More typically, clustering is done *syntactically* through pattern matching. *Patterns* are templates against which a name is compared. They range from names that may simply contain wildcards, which are denoted by ‘‘*’’ and match any sequence of characters, to regular expressions. Names matching a particular pattern, such as names with a common prefix ‘‘prefix.*’’, are considered part of the same context. That is, the clustering condition, when applied to a name, returns TRUE if the name matches the pattern.

Clustering conditions are used to assign names to contexts, and they may be applied to an existing context to further partition the context into smaller contexts. Thus, starting with the complete configuration database as a single context, a sequence of clustering conditions can be applied to yield a group of reasonably sized contexts. To check if a given name exists in a particular context, a server need only apply the same clustering conditions to the name.

3.5. Context bindings

When presented with a name to be resolved, a server might first look in local contexts for an authority attribute for the named object; if the naming authorities can not be readily determined, additional configuration data, called *context bindings*, must exist locally for the name resolution to proceed. A *context binding* associates a clustering condition with the name of a context (the context containing names that satisfy the clustering condition) and the name servers that store the context. Contexts may contain both authority attributes and context bindings.

The *structure-free name resolution* algorithm works as follows: Given a name to be resolved in some context, the particular context is searched for either an authority attribute for the named object or a context binding containing a clustering condition that yields TRUE when applied to the name; in the latter case, the name is then resolved in the new context specified by the context binding, perhaps on a different server. Thus, resolving a name is a matter of successively binding names within contexts until the authoritative name servers for the named object are discovered. That is, the name resolution mechanism traverses a *resolution chain* of context bindings until it encounters an authority attribute.

When a name is originally presented for resolution, an *initial context* must be chosen in which to start the resolution chain. The initial context must contain authority attributes or context bindings for all names in the name space. Global names result if and only if the initial context is a global one, that is, all name servers share a common initial context. Relative names arise if the initial context used in name resolution is not a global one, but is relative to the particular server presented with the resolution request.

3.6. Name resolution trees: putting it all together

For simplicity, the set of clustering conditions chosen by a given naming system should partition the name space such that each name exists in exactly one context. Typically, the initial context will contain just context bindings. The bindings between contexts form a hierarchical tree structure in which contexts containing context bindings form internal nodes of the tree while contexts containing only authority attributes form the leafs. Names are resolved by following a path through this *name resolution tree*.

Importantly, the name resolution tree need bear little resemblance to the name space tree. Specifically, the name resolution tree may have different branching factors than the syntactic view of the name space. Moreover, the name resolution tree can change over time by choosing a different set of clustering conditions or adding new clustering conditions to further divide existing contexts. The name resolution chains for a given name may vary in length for the different clustering strategies. Such changes do not affect the actual name space, only the resolution of names. Therein lies the strength of structure-free name management.

As far as the name service is concerned, names could be completely void of structure. Algorithmic clustering allows a healthy name resolution tree to be built even for flat names. In practice, names will undoubtedly have structure so that parts of the name space can be administratively allocated to different organizations to avoid name conflicts [Abraham and Dalal 80]. Syntactic clustering allows names to be resolved in a manner similar to their structure, as is done by virtually all current name management systems; simple pattern matching suffices for emulating existing name services. A mixture of syntactic and non-syntactic clustering should prove useful for resolving names in evolving systems.

3.7. Scenarios revisited

scenario #1: splitting Grapevine registries

In Grapevine, with names of the form ‘‘subname.registry’’, pattern matching based on registry names is used to syntactically cluster the name space. Registries serve as contexts. The initial context, called "GV", contains a context binding for every registry.

Using structure-free name management, large registries can be algorithmically partitioned into smaller contexts. The resolution chains for some object names would grow from one link to two; the first context binding being done syntactically, while the second is done perhaps by a hash function. The structure of Grapevine’s names do not change. A sample name resolution tree for such a scheme is depicted in Figure 1.

- figure goes here -

Figure 1. **Clustering large Grapevine registries algorithmically**

scenario #2: adding more structure to names

Although existing name management mechanisms for hierarchical name spaces resolve names a label at a time, syntactic clustering conditions are not restricted to matching a single additional label in each step. That is, even using syntactic clustering, the length of the resolution chain for various names need not correspond exactly to the number of labels in the names. Name resolution can be tailored according to the desired response time for resolving names and the size of contexts.

Syntactic clustering in which a variable number of labels can be matched allows potential performance advantages to be obtained over traditional resolution schemes. Particularly, regions of the name space that are abnormally sparse may be clustered together for purposes of name resolution. As these sparse regions fill out over time, intermediary context bindings can be easily reintroduced; no names need to change, only their clustering.

scenario #3: merging existing name spaces

Merging independently created name spaces with different naming conventions can be accomplished with the techniques outlined in this paper since the name management mechanisms ignore the structure of names. Working out name conflicts becomes solely an administrative problem.

That is, if no name exists in both environments referring to different objects, then clustering conditions can be applied to the combined name space to allow the existing names to be resolved by clients in either environment. One way of dealing with conflicting names, which doesn’t require changing the names imbedded in software systems, is to treat the current names as partially qualified names; client software can expand partially qualified names into globally unambiguous names before submitting them to the name service.

4. Conclusions

This paper dispels the common belief that the structure of names directly dictates their resolution. The information maintained by a name service consists of two types: attribute data and configuration data (embodied in the name resolution tree). Name structure need not, but can, be exploited to partition and distribute the configuration data among name servers. The important result of the new name resolution mechanisms presented herein is that the name resolution tree can change in response to changes in the distributed environment without requiring changes to the actual name space.

The distinction between attribute data and configuration data is also important. Autonomous organizations may supply their own name servers or freely choose other servers to store the attributes for their objects. Organizations may enforce required access controls on attribute data maintained on their name servers; on the other hand, configuration data used to resolve names can be freely shared by all since it contains no private information.

Separating three activities: choosing names, selecting the storage sites for object attributes, and resolving an object’s name, is both feasible and desirable. Structure-free name management provides the flexibility needed to manage names in evolving distributed environments. Specifically, a distributed name service employing structure-free name management can be easily reconfigured to improve performance or meet changing demands.

5. Further reading

See [Terry 85] for a more formal and thorough discussion of structure-free name management. That report formalizes the space/time tradeoffs allowed by structure-free name management, presents a more general notion of context bindings that can support other styles of naming such as naming networks, and includes a prototype implementation of the mechanisms needed to support this new scheme for managing names.

6. References

[Abraham and Dalal 80]
S. M. Abraham and Y. K. Dalal.
Techniques for decentralized management of distributed systems.
Proceedings 20th IEEE Computer Society International Conference (COMPCON), San Francisco, California, February 1980, pages 430-436.

[Birrell *et al.* 82]
A. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder.
Grapevine: An exercise in distributed computing.
Communications of the ACM 25(4):260-274, April 1982.

[Ellis 77]
C. A. Ellis.
Consistency and correctness of duplicate databases.
Proceedings Sixth Symposium on Operating Systems Principles, Princeton, New Jersey, November 1977, pages 67-84.

[Gifford 79]
D. K. Gifford.
Weighted voting for replicated data.
Proceedings Seventh Symposium on Operating Systems Principles, Pacific Grove, California, December 1979, pages 150-162.

[Lindsay 80]
B. Lindsay.
Object naming and catalog management for a distributed database manager.
Proceedings Second International Conference on Distributed Computing Systems, Paris, France, April 1981, pages 31-40.
Also available as IBM Research Report RJ2914, August 1979.

[Mockapetris 83]
P. Mockapetris.
Domain names -- Concepts and facilities.
USC Information Sciences Institute, RFC 882, November 1983.

[Oppen and Dalal 83]
D. C. Oppen and Y. K. Dalal.
The Clearinghouse: A decentralized agent for locating named objects in a distributed environment.
ACM Transactions on Office Information Systems 1(3):230-253, July 1983.
An expanded version of this paper is available as Xerox Report OPD-T8103, October 1981.

[Saltzer 78]
J. H. Saltzer.
Naming and binding of objects.
In R. Bayer, R. M. Graham, and G. Seegmuller, editors. *Operating Systems: An Advanced Course*, Springer-Verlag, 1978, pages 99-208.

[Schroeder *et al.* 84]
M. D. Schroeder, A. D. Birrell, and R. M. Needham.
Experience with Grapevine: The growth of a distributed system.
ACM Transactions on Computer Systems 2(1):3-23, February 1984.

[Su and Postel 82]
Z. Su and J. Postel.
The domain naming convention for internet user applications.
Network Information Center, SRI International, RFC 819, August 1982.

[Terry 85]
D. B. Terry.
Distributed Name Servers: Naming and Caching in Large Distributed Computing Environments.
Ph.D. Thesis, University of California at Berkeley, February 1985.
Available as Xerox Palo Alto Research Center, Technical Report CSL-85-1.