X/Open Federated Naming

The X/Open Federated Naming (XFN) specification defines uniform naming interfaces for accessing a variety of naming systems. XFN specifies a syntax for composite names, which are names that span multiple naming systems, and provides operations to join existing naming systems together into a relatively seamless naming federation.

by Elizabeth A. Martin

Naming of objects is a fundamental need in a computing system. A naming service maps human-readable names to internal location information that programs use to access the named objects. Current distributed computing environments that take advantage of large computer networks present new problems and requirements for the naming service.

Heterogeneous naming systems are a reality. Unlike the naming service in a single-host system, the naming service in a distributed system is usually not a monolithic component but consists of various naming systems embedded in different pieces of the system. The naming systems in the Open Software Foundation (OSF) Distributed Computing Environment (DCE, see article, page 6) include the X.500 directory service,[1] the DCE Cell Directory Service (CDS),[2] and the DCE Distributed File System (DFS, see page 6). The DCE security service (see article, page 41) and the DCE daemon (see article, page 6) also support namespaces. A typical DCE installation will have applications that have their own naming systems, such as databases, email, desktops, and spreadsheets.

These different naming systems have arisen in part because they meet different requirements. The DCE security server uses special and somewhat inconvenient measures to protect the keys of principals in the system. DCE CDS directories are replicated but a desktop is not. A spreadsheet names fine-grained objects (its cells) which present unique scaling problems. New naming systems will continue to appear, particularly in applications.

Up to now, there has been no basic and consistent naming interface. Each naming system has its own API, so application programmers must write custom software for each naming system that their applications use. When applications are ported to different systems, they must be modified to use that system's naming interfaces. As new naming systems are introduced, applications that need to use them must be extended.

There has also been no first-class, generic support for composite names. A few distributed systems support composite names—names that span multiple naming systems. This support is limited and specialized. The DCE name /.../ch.hp.com/sec/principal/jsmith is a composite name. ch.hp.com is resolved in the Internet DNS[3,4] namespace, sec is resolved in the DCE CDS namespace, and principal/jsmith is resolved in the security service's namespace. In DCE only the security, file system, and DCE daemon namespaces can be accessed through composite names. UNIX® rcp uses composite names in a different way from DCE. For example, ajax:/usr/jsmith/naming/memo.txt is an rcp name with two components: ajax is a host name and /usr/jsmith/naming/memo.txt names a file on ajax. DCE and rcp use their own syntaxes and conventions for their names.

Another area of inconsistency between naming systems is their policies for how the namespace is structured. Many systems have very little policy and what policy there is has evolved in a haphazard way. Application writers who use the namespace to advertise their services must follow different conventions for the various environments in which their programs will run or they must invent their own policies for using the namespace. Administrators who configure a site are also faced with confusing, inconsistent, or no policy for how to use the namespace. End users need intuitive ways of finding and naming objects.

Overview of X/Open Federated Naming (XFN)

Several vendors of distributed computing systems realized that they shared these naming problems. Engineers from Digital, HP, IBM, OSF, SNI, and Sunsoft started work on a naming specification in June 1993. In March 1994 version 1 of the Federated Naming Specification[5] went to X/Open® for fast-track review. The specification achieved preliminary status in July 1994. The multivendor team continued to work on extensions to the specification and on validating it before it became part of the X/Open Common Application Environment (CAE) in 1995.

The XFN specification defines application programming interfaces (APIs) and corresponding remote procedure call (RPC) interfaces. XFN specifies a naming syntax for composite names and provides operations to join different naming systems together into a relatively seamless naming federation. XFN also specifies some naming policy.

Fig. 1 illustrates an XFN configuration. The XFN API is layered over a framework into which different context implementations are inserted. A specific context implementation is required for each naming system in a federation. A context implementation maps XFN operations into operations that are native to its naming system. For example, the NIS+[6] context implementation maps operations in the XFN API to corresponding operations in the NIS+ API. A naming system's software below the context implementation is not changed.

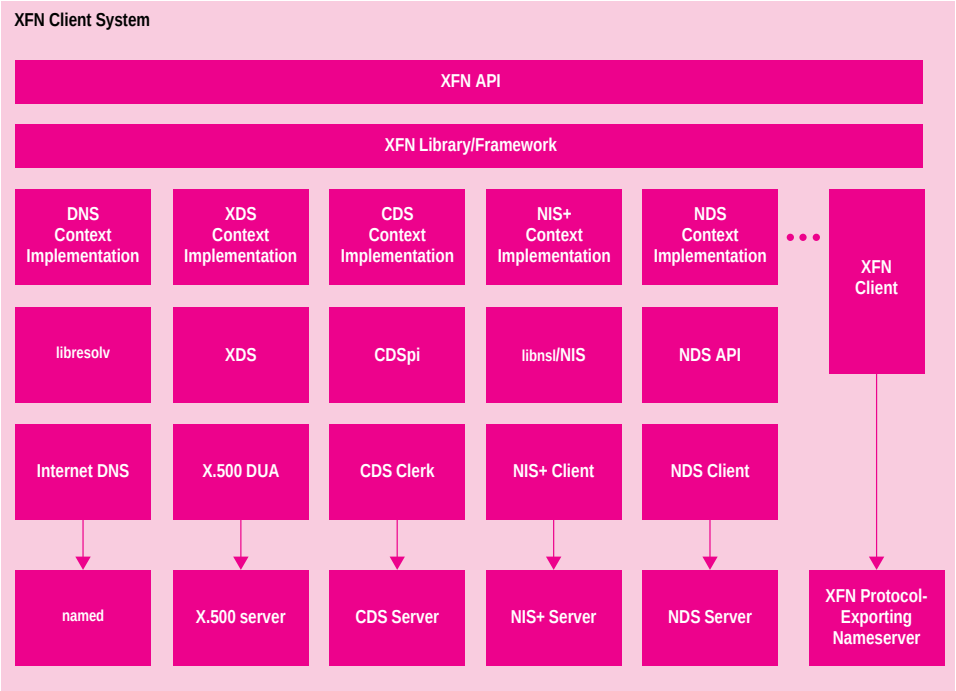

Fig. 1. XFN configuration using client context implementations. A program seeking internal location information for a human-readable name passes the name to the XFN API. The name is broken apart and processed by the appropriate naming systems, and the desired location information is returned by the naming system servers (bottom row).

To join a federation, a naming system must simply provide its specific context implementation.

In Fig. 1 the client-side software for five naming systems runs on the XFN client system. In addition, an XFN client module that imports an XFN protocol is on this system. The XFN client module may do caching and other typical naming client jobs. Each naming client on the system uses its native protocol to communicate with its server.

Definitions

In this section and hereafter in this article, paragraphs in quotation marks are taken directly from the X/Open Federated Naming Specification.[5]

"Every name is generated by a set of syntactic rules called a *naming convention*. An *atomic name* is an indivisible component of a name, as defined by the naming convention. A *compound name* represents a sequence of one or more atomic names composed according to the naming convention."

Case sensitivity, the choice of escape, quote, and delimiter characters, and the order of atomic names in a compound name are common features of a naming convention.

"In UNIX pathnames, atomic names are ordered left to right, and are delimited by slash (/) characters. The UNIX pathname usr/local/bin is a compound name representing the sequence of atomic names, usr, local, and bin. In names from the Internet DNS, atomic names are ordered from right to left, and are delimited by dot (.) characters. Thus, the DNS name sales.Wiz.com is a compound name representing the sequence of atomic names com, Wiz, sales."

"The *reference* of an object contains one or more communication endpoints (addresses). The association of an atomic name with an object reference is called a *binding*. A *context* is an object whose state is a set of bindings. Every context has an associated naming convention."

A UNIX directory is a type of context. An atomic name in one context can be bound to a reference to another naming context object, called a *subcontext*.

"A *naming system* is a connected set of contexts of the same type (having the same naming convention) and providing the same set of operations with identical semantics. In the UNIX operating system, for example, the set of directories in a given file system (and the naming operations on directories) constitute a naming system. A *naming service* is the service offered by a naming system. It is accessed using its interface. A *namespace* is the set of all names in a naming system."

The XFN API

XFN defines uniform naming interfaces that support basic naming functionality. As illustrated in Fig. 1, the XFN interface is layered over specific naming services' APIs. The details of the underlying naming system are hidden from the application. Applications that use the XFN API can access a variety of current and future naming systems without modification.

The operations in the XFN interface range from simple to complex. Simple naming systems are not expected to support the more complicated operations, but the functionality offered by sophisticated naming systems can still be accessed via the XFN API.

The XFN base context interface includes operations to bind an atomic name in a context to an XFN reference and to unbind a name. Other operations in the XFN base context interface look up a name and return its reference, look up a link, list all names and bindings in a context, and create a subcontext.

XFN supports the notion of attributes (or properties) associated with a name. Attributes can be used to provide summary characteristics about the object being named. For example, a

printer might be named /.../Wiz.com/eng/os/service/prntr1. The name prntr1 would be bound to an XFN reference that contains the address of the server for that printer. Attributes could also be associated with the name prntr1 that describe its type (LaserJet, inkjet, etc.) and the formats it supports.

Attributes are accessed through the XFN attribute interface, which includes operations to set, modify, and get attributes associated with a name in a context. An attribute consists of an identifier, a syntax, and one or more values. Operations to search for names whose attribute values match a filter expression are also defined. In the printer example, a search operation could be used to locate a LaserJet printer in the eng/os department that supports the PostScript™ format.

The XFN API has been mapped to Internet DNS, CCITT X.500, DCE CDS, and ONC NIS+. Since X.500 provides the most functionality of these naming systems through its XDS/XOM API, this naming system presented the most challenges for XFN. XFN captures the functionality of XDS/XOM but is a simpler, more intuitive API.

Support for Composite Names

XFN specifies a syntax and parsing rules for composite names. Operations to manipulate these names are also provided.

"A *composite name* consists of an ordered list of zero or more components. Each component is a string name from the namespace of a single naming system and uses the naming syntax of that naming system. A component may be an atomic name or a compound name from that namespace." The string form of a composite name is "the concatenation of the components from left-to-right with the XFN component separator (/) between each component."

In the DCE composite name /.../ch.hp.com/sec/principal/jsmith mentioned earlier, the ch.hp.com component is a compound name in the DNS naming system, whose syntax is right-to-left '.' separated. The second component, sec, is in the DCE CDS naming system, whose syntax is left-to-right '/' separated, like XFN's syntax. The final two components, principal/jsmith, are in the DCE security naming system. This naming system's syntax is also left-to-right '/' separated. Since a component is defined as the name between two XFN separators, principal/jsmith is two components even though both are in the same naming system.

Composite names are formed when naming systems are joined by binding location information about a context in one naming system into its parent context in another naming system. This location information about a context in another naming system is called a *next-naming-system pointer*. For most naming systems a next-naming-system pointer is bound to a leaf name in its namespace and is treated like any other name in its namespace. The location information is represented in an XFN reference. The XFN bind operation can be used to create next-naming-system pointers.

Fig. 2 shows how the name /.../Wiz.com/user/jsmith/fs/naming/memo.txt is composed. /... is a reserved token that indicates the root of a global naming system. The Wiz.com component is a name in the DNS naming system, user/jsmith/fs is in the DCE CDS naming system, and naming/memo.txt names a DFS file. Location

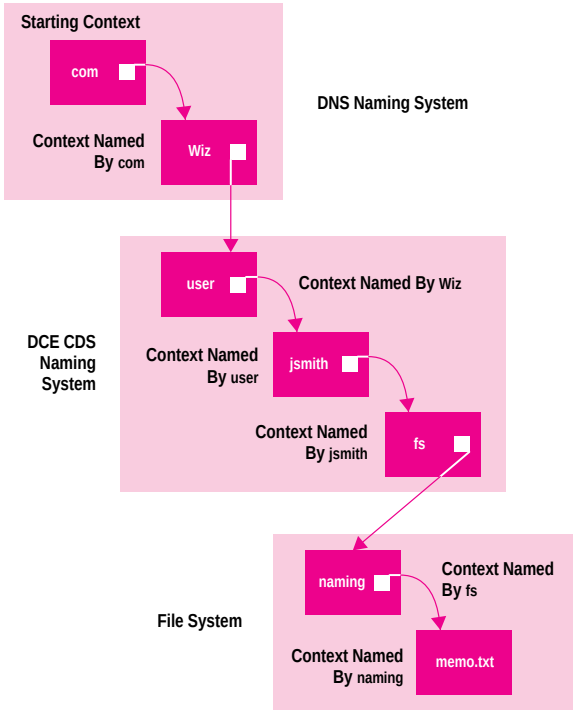

Fig. 2. Next-naming-system pointers (Wiz and fs).

information about the DCE CDS context in which user is bound is associated with the name Wiz in DNS. The atomic name fs is bound in the CDS context user/jsmith to an object reference with location information of jsmith's home directory in DFS. Wiz and fs are next-naming-system pointers.

The XFN framework controls path resolution of a composite name. To resolve /.../Wiz.com/user/jsmith/fs/naming/memo.txt the XFN framework first invokes the DNS context implementation to resolve Wiz.com. The DNS context implementation makes libresolve calls to gather the information it needs to form the XFN reference associated with Wiz.com, which it returns to the framework. The framework inspects the reference and invokes the context implementation specified in the reference. The framework passes to the context implementation the location of the starting context for resolution and the remaining components to be resolved. In this example, the context implementation is for DCE CDS, the starting context is the one named by Wiz.com, and the name to be resolved is user/jsmith/fs/naming/memo.txt. CDS can only resolve user/jsmith/fs. It returns the XFN reference bound to user/jsmith/fs and the remaining components to be resolved back to the framework. The framework then passes the remaining name, naming/memo.txt, to the file system to complete the resolution.

XFN Protocols and Configurations

XFN specifies client-server RPC interfaces for use with two RPC protocols: DCE RPC and ONC RPC. The protocols support the operations in the XFN API. New naming systems and some current ones are expected to use one of these protocols for their client/server communications.

"The advantage for naming systems that export an XFN protocol is that any existing XFN client that imports the protocol can be used to communicate with it. This is particularly useful for applications that need to export naming interfaces. Application programmers do not have to duplicate the client-side implementation and they do not have to invent new naming interfaces. This provides additional benefits such as the ability to use caching and other mechanisms provided by the XFN client implementations, and a direct (and possibly more efficient) mapping of XFN operations to the naming operations."

The XFN naming model presents a hierarchical namespace that incorporates different naming systems. The naming systems are connected together into three levels. The top level is a global namespace; X.500 and DNS are expected to control this level. The next level is an enterprise namespace; DCE CDS, ONC NIS+, Banyan Streettalk, and Novell NDS[7] are considered enterprise naming systems. The third level is the application namespace. The DCE security service, a file system, and a desktop support application namespaces.

The XFN model, API, and protocols provide a toolkit for configuring naming federations in various ways. Fig. 1 illustrates a heavyweight XFN client system with context implementations and client-side code for five naming systems and a module that imports an XFN protocol. Fig. 3 shows a lightweight XFN client system that only runs the naming module that imports an XFN protocol. Multiple name servers export the XFN protocol. Two of the name servers use a variation of their context implementations to map arriving XFN calls to their naming systems' native operations. These servers also export their native protocols to support clients running

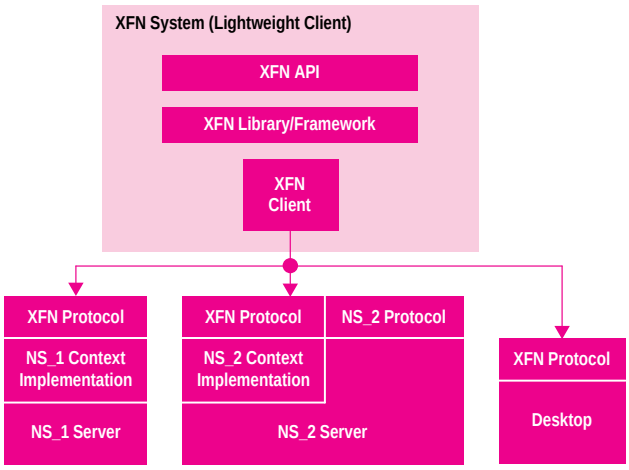

Fig. 3. Lightweight XFN client configuration with multiple name servers.

legacy software. The desktop application was originally written to export its namespace with the XFN protocol.

The two systems shown in Fig. 4 are a lightweight XFN client and a server that acts as an intermediary. Like the client in Fig. 3, the XFN client in Fig. 4 only runs the naming module that imports an XFN protocol. None of the legacy systems' client-side software needs to run on this system. Depending on the client system's requirements, the XFN client can be implemented and configured to consume more or less resources. For example, the XFN client might defer to the caching mechanisms provided by the native naming

Fig. 4. Lightweight XFN client configuration with surrogate client on server.

system clients. "The legacy naming system clients in Fig. 4 reside on a remote system (similar to Fig. 1) that also exports the DCE XFN protocol. This remote client can be viewed as a surrogate or proxy client that acts on behalf of the initial requestor and performs the native naming system functions."

Another common XFN configuration combines Figs. 3 and 4. Some name servers export the XFN protocol and can be accessed directly from the lightweight XFN client. Other name systems are accessed via an XFN surrogate client.

XFN Enterprise Policies

The three-level hierarchy of global, enterprise, and application namespaces is an XFN policy that was mentioned in an earlier section. Major entities, such as countries and organizations, are named in the global namespace. Names in a global naming system change infrequently and require sanction from a global authority to do so. The enterprise namespace is assumed to contain names that are local to an organization. XFN policies provide some guidelines for structuring an enterprise namespace. These policies do not apply to the global or application namespaces.

XFN policy recognizes that there are commonly named objects in an enterprise. These are organizational units, users, hosts, services, and files. XFN policy reserves tokens to identify namespaces for these objects and also applies a relationship between them. Table I summarizes XFN enterprise policies. Some examples of names that use XFN policies are:

- /.../Wiz.com/_orgunit/r-d/eng/os/_user/jsmith/_fs/naming/memo.txt. Names jsmith's file naming/memo.txt. jsmith is a user in the r-d/eng/os department of the Wiz.com company.
- /.../Wiz.com/_orgunit/sales/_user/mjones/_service/calendar. Names the calendar service for mjones who is a user in the sales department of the Wiz.com company.
- /.../Wiz.com/_orgunit/newton/bldg300/conf-rm/chaos/_service/calendar. Names the calendar service for the Chaos conference room in building 300 of the Newton site of the Wiz.com company.

Programs that use XFN policies are more portable across computing environments and enterprises. A distributed application, such as a calendar service, has a standard place (a _service context) to put its binding information. An administrator can put information about each user and each host in a central, predictable place. An end user can more easily figure out how to name another user's files, for example.

Despite the fact that XFN policies are minimal, they are controversial. Standard token names raise concerns of name collisions. XFN specifies these tokens on the premise that the benefits of a more structured namespace outweigh the risk that XFN tokens will collide with names that are already in a namespace. An XFN implementation can sacrifice its portability and customize its own tokens to identify the namespaces for common objects. An XFN implementation can conform to the XFN API but to some or none of the XFN policies. An enterprise namespace will normally have many contexts that are outside of the XFN policy domain and may have additional policies of its own.

Table I
XFN Enterprise Policies

Context Type	Context Type Token	Parent Context	Subordinate Context
Organizational Unit	_orgunit	enterprise root	user, host, file system, service
User	_user	enterprise root, organizational unit	service, file system
Host	_host	enterprise root, organizational unit	service, file system
Service	_service	enterprise root, organizational unit, user, host	not specified
File System	_fs	enterprise root, organizational unit, user, host	not specified

Other Naming APIs

Some naming APIs, such as the DCE RPC Name Service Independent (NSI) Interface[8] and the OMG Common Object Service's Naming Service Interface,[9] are customized interfaces that may be layered over an XFN API and its implementation.

The RPC NSI provides a high level of abstraction for navigating a namespace and yielding DCE RPC location information in the form of RPC binding handles. The OMG naming interface is a subset of the XFN basic context interface. The OMG interface maps names to CORBA object references. Unlike RPC NSI and the OMG naming interface, XFN accepts many different types of object references and provides mechanisms to extend the set of object references. Also, neither the DCE RPC NSI nor the OMG naming interface has support for attributes.

When these customized interfaces are implemented over XFN, they take advantage of XFN benefits such as portability and federation and they leverage all the software that supports the XFN API.

Conclusions

Among the benefits that XFN provides are:

- A uniform naming interface for accessing different naming systems.
- Application programming interfaces as well as RPC interfaces.
- A naming syntax for composite names.
- Operations to join different naming systems together into a naming federation.
- A framework that supports the addition of new naming systems to an XFN federation with no changes to applications or to current member naming systems. A naming system that joins a federation must only supply a context implementation that maps the XFN API or an XFN protocol to its native

operations. Otherwise, the naming system's software is not changed.
- Support for small clients.
- Easier administration of the various naming systems in a distributed computing environment. Browsers and editors that are written to the XFN API can access an entire federated namespace.
- Application power. XFN applications can access a wide variety of naming systems through the same simple, yet functional API.

Future Directions

Future work needs to be done on policy. Different vendors that offer similar applications need guidelines for sorting out their uses of the namespace. Users sometimes want to select among similar or replicated services based on network topology or load balance. Administrators often have common information about a group of users and customized per-user information. Namespace policies and software could support these requirements.

Acknowledgments

This paper is a summary of the X/Open Federated Naming Specification. Quoted paragraphs are taken directly from the specification as are some of the figures and tables. The X/Open Federated Naming architecture team includes: Rangaswamy Vasudevan, Rosanna Lee, and Vinnie Ryan from Sunsoft, Ellen Stokes and Dave Bachmann from IBM, Norbert Lesser and Arthur Harvey from OSF and the author from HP. Joseph Pato from HP, Arthur Gaylord from the University of Massachusetts at Amherst, and Richard Curtis from Banyan were early reviewers and are consultants to the architecture team. Peter Dejong, Larry Derany, Michael Kong, and Joseph Pato provided valuable review comments.

References

1. *Information Technology—Open Systems Interconnect—The Directory*, CCITT X.500 (1988, 1993)/ISO Directory, ISO/IEC 9594: 1988, 1993.
2. *X/Open DCE: Directory Services*, X/Open Preliminary Specification, December 1993.
3. P.V. Mockapetris, *Domain Names—Concepts and Facilities*, Internet RFC 1034, November 1987.
4. P.V. Mockapetris, *Domain Names—Implementation and Specification*, Internet RFC 1035, November 1987.
5. *Federated Naming: The XFN Specification*, X/Open Preliminary Specification, July 1994.
6. R.Ramsey, *All About Administering NIS+*, SunSoft Press.
7. D. Bierer, et al, *Netware 4 for Professionals*, New Riders Publishing, 1993.
8. *X/Open DCE: Remote Procedure Call*, X/Open Preliminary Specification, October 1993. Specifies RPC NSI.
9. "Naming Service Specification," *OMG Common Object Services Specification, Volume 1*, March 1994.