CHAPTER 2

Solaris Naming Services Architecture

The Solaris operating environment provides a sophisticated infrastructure that supports a variety of naming services. The architecture on which it is based is extensible and able to accommodate new naming services without the need for a rewrite of important operating system utilities that access naming services. The Solaris 8 LDAP naming service plugs into this architecture and is thus accessible to system utilities that formerly had only NIS, NIS+, and DNS available.

Reading this chapter is not an absolute requirement for deployment, but if you become familiar with some of the architectural nuances, you can better understand the deployment strategies presented in later chapters. Each naming service has its own unique characteristics which may dictate how you deploy them. Although the focus of this BluePrint is LDAP, it is helpful to understand the feature set of legacy Solaris naming services to see how this new technology compares.

Evolution of Solaris Naming Services

The UNIX operating system was developed to operate in a timesharing environment where users access the server via physically attached ASCII terminals. Users typically accessed only one server, so information about user accounts, group memberships, and so on, only needed to be maintained on that server. Storing that information in a text file worked quite well.

The Berkeley version of UNIX introduced the notion of distributed computing built on top of the TCP/IP protocol. Computers running the UNIX operating system could now easily communicate with one another. However, for things to work smoothly, information about users and other systems in the network needed to be maintained on each server. Storing this data in text files meant that any time something changed, the text files on every server needed to be updated.

In 1985, Sun Microsystems produced NIS (Network Information Service), one of the first UNIX-based distributed naming service as a replacement for storing information in text files. The text files would be converted to binary maps that would only be stored on selected computers, called NIS servers, in the network. The other computers in the network would contact the NIS servers when they needed access to the information.

However, some text files still needed to be maintained for two reasons: 1) some data was required during the booting process before access to the network was established and 2) there had to be a way to log in if the computer was disconnected from the network. Moreover, some mechanism was required so that the operating system utilities could search both text files and NIS, since NIS could not completely replace the text files.

The introduction of NIS presented a new system administration model, by which information was administered from a central repository and not all administrators were granted permission to update it. Since some users still wanted to be able to manage local accounts and system information, they needed some way to do this without administering of NIS maps.

NIS and Files Coexistence

To solve the problem of providing a centrally administered naming service while maintaining some local control, Sun's first implementation of NIS searched the local files before the NIS naming service was consulted. A special character was inserted into the text files to tell the operating system when to start searching the NIS maps. Any line beginning with a "+" character was the signal to contact NIS. For example, the `/etc/host` file would look like this:

```
127.0.0.1 localhost
129.148.181.130 tiger
129.154.86.22 galaxy
+
```

In this example, the `/etc/hosts` file would be searched for the specified host. If the host specified is not `tiger` or `galaxy`, then the NIS host map is searched. If the host name does not appear in the NIS map either, an error is returned.

Note – The "+" character only has an effect when the Solaris 1 operating environment is running. It will have no effect if the Solaris 2 or later operating environment is running except when run in the Solaris 1 compatibility mode.

NIS and DNS Coexistence

About the same time that Sun introduced NIS, standards for a universal naming system were being defined in RFC 1034 and RFC 1035. Later, implementations of this specification called the Domain Naming System (DNS) began to appear, like the Solaris `in.named` program, which was derived from Berkeley Internet Name Demon (BIND), found in Berkeley UNIX. Although NIS worked well to store host names and IP addresses of computers within an organization, DNS could scale much better and gained industry-wide adoption.

Companies deploying NIS tended to store the host name and IP addresses of their Sun workstation and server networks in NIS maps, but used DNS to look up names of computers outside of the network. To enable the two naming services to interoperate, Sun added a DNS forwarding capability to the NIS server.

The way DNS forwarding works is that if a search is made in an NIS map that has this feature enabled, the search request is passed on to a DNS server for resolution if the host name is not found. To implement this idea, the `hosts.byname` and `hosts.byaddr` maps must have the `YP_INTERDOMAIN` key in them. Creation of this key requires a simple modification to the NIS `Makefile`.

The alternative to enabling DNS forwarding is to include DNS as an option in the `nsswitch.conf` file which is described in the next section. It is not advisable to use both schemes together because redundant searches are performed if the name cannot be resolved, that is DNS will be searched twice.

Solaris Naming Service Switch

With the release of the Solaris 2 operating system, Sun introduced a new naming service called NIS+ and an infrastructure for managing the coexistence of multiple naming services. With NIS and DNS already widely deployed, and NIS+ added to the mix, some mechanism for easy interoperability was required. The DNS forwarding mechanism and "+" notation used in NIS maps were not easily extensible to new naming services like NIS+.

To support the switch, Sun programmers developed a new Application Programming Interface (API) that system utilities and other applications could use instead of talking directly to the naming service. Programs written to this API do not need to know the implementation details of the naming service they are accessing. The switch also gives the system administrator the flexibility to choose which naming services are consulted and in which order.

Solaris Naming Service Switch Architecture

The main components that constitute the architecture are the Network Services libraries, the policy configuration file, and interfaces to the available naming services. A special tag identifies the location where the requested information is actually stored. As shown in FIGURE 2-1 the available tags are files, `nis`, `nisplus`, `dns`, `compat` (for `passwd`), with `ldap` added to the Solaris 8 operating environment.

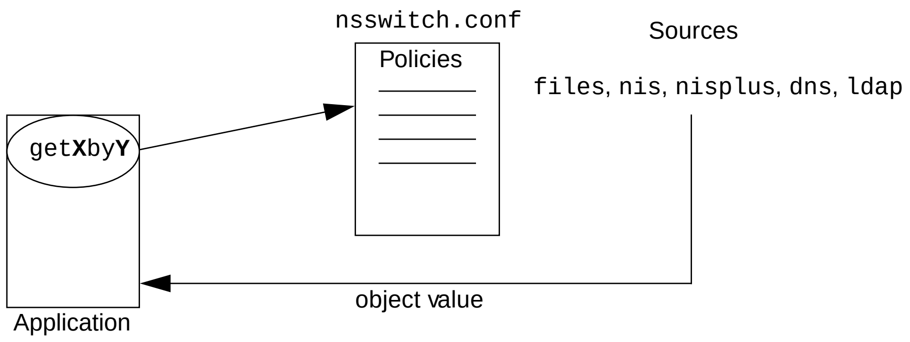

FIGURE 2-1 Naming Service Switch Functions

FIGURE 2-1 shows the flow of information when an application calls the Network Services libraries. These library calls are in the form of `getXbyY`, for example, `gethostbyname(),` and are independent of any naming service. When the application makes the call, the library routine consults the `nsswitch.conf` file to determine which naming services to consult. The specified naming services are then searched in order until a match is found or a `NOTFOUND` error is returned.

`nsswitch.conf` File

The policies that determine which naming service sources are searched and in what order reside in the `/etc/nsswitch.conf` file. Sample configuration files that favor a particular naming service are provided with the Solaris operating environment in the `/etc` directory. These files are copied and automatically used as the `nsswitch.conf` file when a primary naming service is chosen during the Solaris installation process.

An example of the configuration files that favor `nis` follows.

```
#
# /etc/nsswitch.nis:
#
# An example file that could be copied over to /etc/nsswitch.conf; it
# uses NIS (YP) in conjunction with files.
#
# "hosts:" and "services:" in this file are used only if the
# /etc/netconfig file has a "-" for nametoaddr_libs of "inet" transports.

# the following two lines obviate the "+" entry in /etc/passwd and /etc/
group.
passwd:     files nis
group:      files nis

# consult /etc "files" only if nis is down.
hosts:      nis [NOTFOUND=return] files
ipnodes:    files

networks:   nis [NOTFOUND=return] files
protocols:  nis [NOTFOUND=return] files
rpc:        nis [NOTFOUND=return] files
ethers:     nis [NOTFOUND=return] files
netmasks:   nis [NOTFOUND=return] files
bootparams: nis [NOTFOUND=return] files
publickey:  nis [NOTFOUND=return] files

netgroup:   nis

automount:  files nis
aliases:    files nis
```

The objects for which search policies can be set appear on the left. The search order, or policy, appears to the right of the object. In the case of the `passwd` object, the local `/etc/passwd` file is checked first for the user's name, and if the name is found, the password is returned. If the user's name is not found in the `/etc/passwd` file, the `nis passwd` map is searched.

The tag `NOTFOUND=return` is used to direct the switch to look only in the naming services listed to the left unless these services are not operational. In the sample file, `files` would only be consulted if `nis` is not responding. This tag speeds up search times by eliminating unnecessary searches and at the same time provides a backup if the primary naming service is down.

NIS Architecture Overview

Even though the first implementation of NIS appeared almost 15 years ago, NIS is still the most widely used Solaris naming service, and the basic architecture has not changed. This section looks at how NIS clients interoperate and how information is stored and updated in NIS.

NIS Client Server Architecture

Deployment of NIS consists of one or more servers and clients that access the servers. Clients and servers communicate with each other by the Remote Procedure Call (RPC) mechanism. NIS client and server implementations are available on many different platforms and can interoperate with one another.

FIGURE 2-2 shows the major components of NIS.

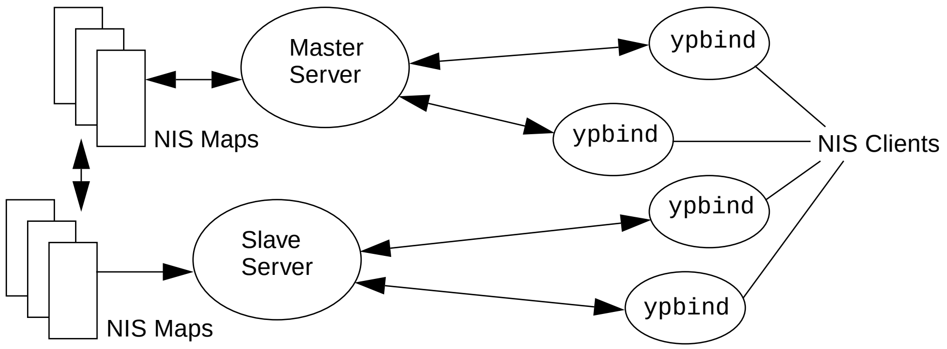

FIGURE 2-2 Major NIS Components

NIS uses a master-slave model by which all updates to NIS maps are performed on the master, then propagated to the slave servers. The propagation can be performed in either a push or pull manner, that is, either initiated by the master or by the client.

The map transfer protocol was not designed to accommodate large maps. Instead of only propagating incremental changes, entire maps are transferred. Careful planning of scheduling policies for map transfers is advisable to prevent overloading of a network during peak time.

How NIS Clients Bind to the NIS Server

A system running the Solaris operating environment typically becomes an NIS client at installation, although it could be configured as one later. A client is only required to supply two pieces of information: 1) the domain name it is joining and 2) how to locate the NIS server(s).

The domain name of the NIS client must exactly match the domain name of the NIS server to establish a connection. Unlike DNS domain names, NIS client names are case sensitive. A Solaris system can belong to both an NIS and a DNS domain. These domains could have the same or different names. The connection from client to server is referred to as *binding* which takes place at boot time. An NIS client can potentially bind to either an NIS master or an NIS slave server. There are two methods for locating a NIS server to bind to.

- Broadcast method — Send out a broadcast message and bind to the first server that responds.
- Specified Server method — Specify a server or list of servers to bind to.

The Broadcast method only works if there is an NIS server on the same subnet. The Specified Server method works regardless of where the NIS server resides. ”NIS High Availability Architecture Features“ on page 19, discusses the pros and cons of using each method.

NIS Maps

NIS uses a flat namespace where a series of maps reside. Each NIS domain contains its own set of maps. There is no relationship between maps or between NIS domains. The maps contain a pair of entries: the first is the keyword and the second is the value retrieved. TABLE 2-1 and TABLE 2-2 show examples of two different NIS maps.

TABLE 2-1 `hosts.byname`

Keyword	Value
tulip	192.9.200.1
geranium	192.9.200.2
sunflower	192.9.200.3
marigold	192.9.200.4

TABLE 2-2 `hosts.byaddr`

Keyword	Value
192.9.200.1	tulip
192.9.200.2	geranium
192.9.200.3	sunflower
192.9.200.4	marigold

In the preceding examples, the two maps contain the same information, but in different order. This ordering is necessary so a search can be performed both on a host name and an IP address. So that the two maps do not get out of sync, they are automatically created together whenever the map data is updated.

Creating NIS Maps

NIS maps are converted from text files to a binary `dbm` file by the `makedbm` command as shown in FIGURE 2-3.

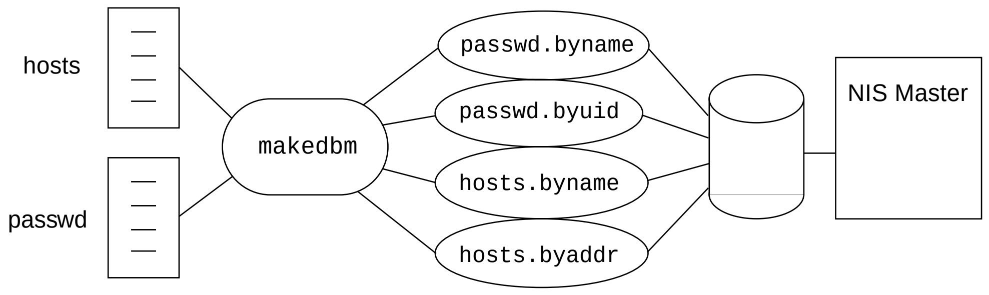

FIGURE 2-3 Creation of NIS Maps

In FIGURE 2-3 the source, or master, for the NIS maps is contained in text files shown on the left. The best practice is to create a copy of one of these files and only edit the copy. These files should be stored in a secure area and backed up frequently.

Once the source files have been created, the `makedbm` command is used to generate the new maps. To make things easier to administer a default, `Makefile`, is provided to perform the `makedbm` operation for the standard NIS maps.

Note – Updates to NIS maps are always performed on the NIS master server that owns the map.

Although it is possible to have NIS maps owned by different masters within a domain, joint ownership is not advisable. In this scenario, an NIS server could act as a master to some maps and as a slave to others. Keeping track of which server is master to which maps could be an administrative nightmare, so it is best to make one server master of all the maps.

NIS High Availability Architecture Features

The main high availability feature of NIS is master-slave data replication. All updates are performed on the master, then propagated to the slaves. If one of the NIS servers fails, an NIS client can bind to another one. However, if the master NIS fails, no updates can occur until it comes back online or another NIS master is created. This may seem like a severe restriction, but in practice the information stored in NIS maps is relatively static, so a few hours of downtime is usually acceptable.

How the NIS client handles the failover from one NIS server to another is determined by the method it uses to bind to its NIS server. FIGURE 2-4 and FIGURE 2-5 illustrate how NIS client failover is handled with both the Broadcast and Specified Server methods.

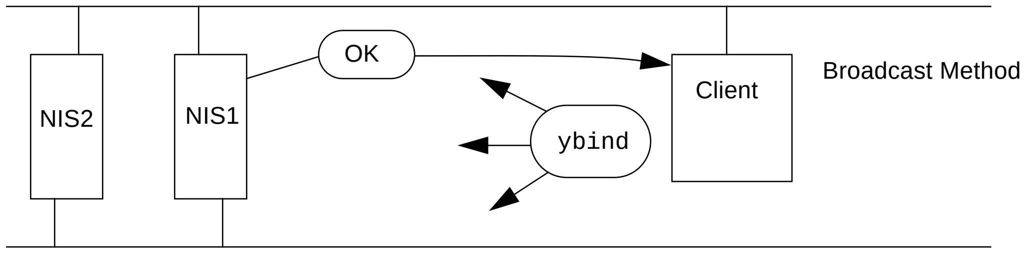

FIGURE 2-4 NIS Client Failover with the Broadcast Method

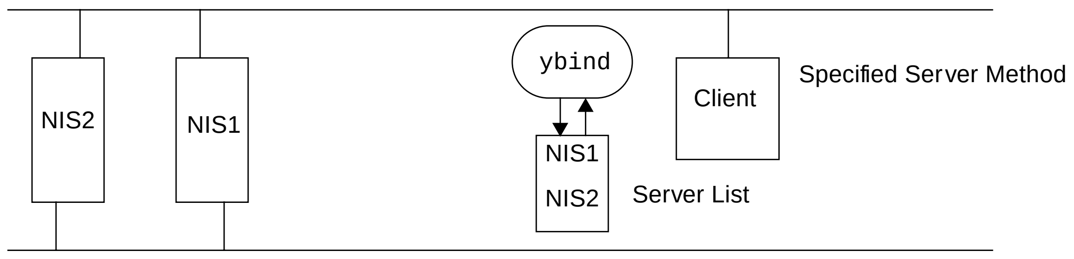

FIGURE 2-5 NIS Client Failover with the Specified Server Method

In the Broadcast method, the NIS client sends out a broadcast to locate an NIS server in the domain of which it is a member. The client then binds to the first server that responds to the broadcast. If the NIS server to which the NIS client binds to fails, then the next time an NIS look up is performed, the operation will time out and the client will issue another `ypbind` broadcast.

In the Specified Server method, the NIS client maintains a list of potential NIS servers. When the client boots, it attempts to bind to the first server in the list. If that server is unavailable, then the client attempts to bind to the next server in the list and so on. The downside of this method is that the time-out period can be lengthy, which gives the impression that the service is down.

A form of load balancing can be achieved with the Broadcast method since the least busy NIS server will respond to the clients `ypbind` request. The disadvantage is that multiple NIS servers must reside on each subnet.

NIS+ Architecture Overview

Sun introduced NIS+ as part of the Solaris 2 operating environment as a replacement for NIS. Several deficiencies in NIS were addressed in the NIS+ architecture. These included:

- Lack of hierarchal namespace
- Weak authentication
- No incremental updates between master and slaves

At the time NIS was developed, Sun's major business focus was the technical computing market. A typical network of Sun systems consisted of a couple of servers and maybe 20-30 workstations that were used by engineers working on the same design project. Verifying the authenticity of a NIS client was not an issue since networks were small and everyone knew who was attached to it.

Because not many companies were wired end to end, the number of names stored in NIS maps was limited and there was little interaction with groups in different locations. A flat namespace, where one NIS domain is not related to another, was sufficient, and since the number of NIS map entries was relatively small, propagating whole maps from master to slave servers was not a major problem.

However, as Sun moved into corporate data centers and companies began creating wide area networks (WANs), networks became larger and the need for a more scalable business-wide naming service became obvious.

NIS+ Client Server Architecture

The architecture of NIS+ is similar to that of NIS in that both naming services employ a master server, in which updates are made, and slave servers or replicas, in which a mirror of the data contained on the master is maintained. However, the similarity ends there.

NIS+ supports two types of masters:

- Root domain master
- Subdomain master

The root master, as the name suggests, acts as the top node in the hierarchal tree. Below the root masters are subdomain masters with other subdomain masters below them. At each level, replica servers can exist to provide redundancy for that section of the tree.

Note – An interesting feature of NIS+ is that the subdomain master is actually a client to the master above it, with the exception of the root master. The ramification of this is that NIS+ must be deployed in a top-down fashion since the domain above it must be configured before a subdomain master is configured.

Propagation of changes from master to replicas is different from NIS. Instead of pushing an entire map when changes are made, NIS+ propagates only the incremental changes. FIGURE 2-6 shows the NIS+ architecture.

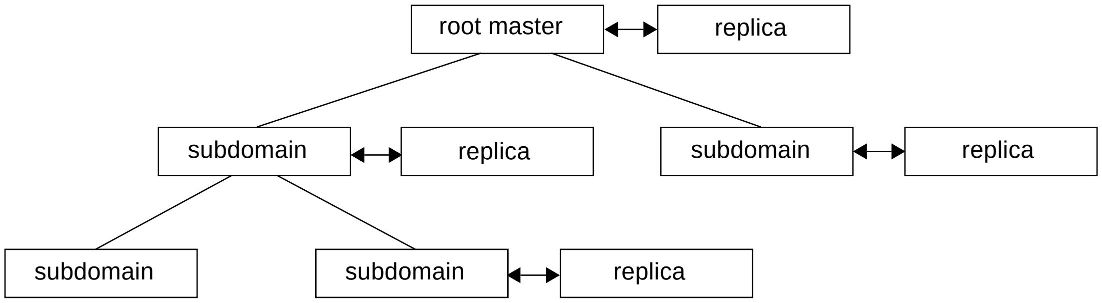

FIGURE 2-6 NIS+ Architecture

How NIS+ Clients Bind to the NIS+ Server

Unlike NIS, which does not authenticate its clients, NIS+ implements the notion of credentials. Two types of credentials exist in the NIS+ world:

- User credentials
- Workstation credentials

The process of creating credentials is quite complex and is beyond the scope of this book. Essentially, the process creates a private/public key pair and stores it in a secure area. During authentication only the public key is passed between the sender and the receiver. Data encrypted with one's private key can be decrypted with one's public key.

Unlike NIS client requests, NIS+ servers perform authentication to see who is sending the request, then authorize that user to perform specific types of access such as read, write, or modify. To gain access to the NIS+ tables, users must provide their credentials, that is in the form of `UID@domainname`. The exchange of credentials is protected by public and private key encryption. If the user is logged in as `root`, then additional credentials that identify the workstation must also be provided.

FIGURE 2-7 summarizes the NIS+ security process.

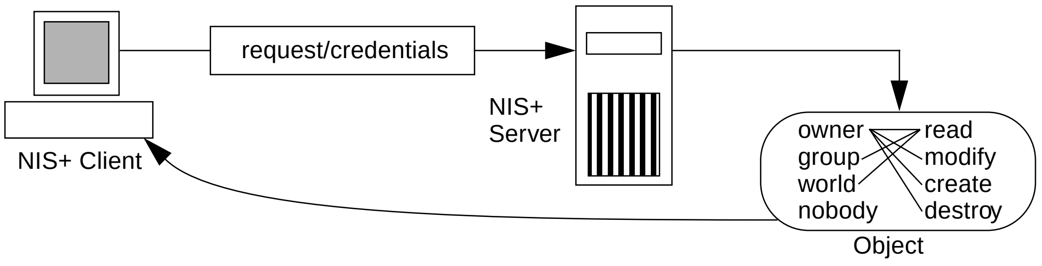

FIGURE 2-7 NIS+ Security Process

As shown in FIGURE 2-7, the following steps take place:

1. The client sends a request for access to the namespace along with its credentials.
2. The server authenticates the client's request by examining the sender's credentials.
3. The server examines the object's definition to determine access rights granted to the sender, or *principal,* as it is called.
4. The server then determines the class of principal: `Owner`, `Group`, `World`, or `Nobody`.
5. The server determines access rights granted to the principal's class.
6. If the access rights granted to the principal's class match the type of operation, the operation is performed.

NIS+ Tables

NIS+ stores information in tables that have a column-entry structure rather than the key-value structure of NIS maps. A client can access information not just by a key, but by any column that is searchable. This approach eliminates the need to create maps that have duplicate information.

The NIS+ tables in TABLE 2-3 come preconfigured and can be populated with the information shown.

TABLE 2-3 NIS+ Tables

Table Name	Information Contained
`Hosts`	Network address and host name of workstations in the domain
`Bootparams`	Location of the `root`, `swap`, and `dump` partition of diskless client in the domain
`Passwd`	User account information for or about every user in the domain
`Cred`	Credentials for principals who belong to the domain
`Group`	The group password, group ID, and members of every UNIX group in the domain
`Netgroup`	Netgroups to which workstations and users in the domain belong
`Mail_Aliases`	Information about the mail aliases of users in the domain
`Timezone`	Time zone of every workstation in the domain
`Networks`	Networks in the domain and their canonical names
`Netmasks`	Networks in the domain and their associated netmasks
`Ethers`	Ethernet address of every workstation in the domain
`Services`	Names of IP services used in the domain and their port numbers
`Protocols`	List of IP protocols used in the domain
`RPC`	RPC program numbers for RPC services available in the domain
`Auto_Home`	Location of all users' home directories in the domain
`Auto_Master`	Automounter map information

NIS+ Interaction with DNS

Unlike NIS, NIS+ has no automatic forwarding feature. To forward requests from NIS+ to DNS, the Naming Service Switch on the client must be configured to search DNS for hosts resolution.

Note – If an NIS+ server is run in NIS compatibility mode, the Naming Service Switch on the NIS+ server needs to be configured to search DNS.

NIS+ High Availability Architecture Features

The availability architecture for NIS+ is similar to that of NIS, but with the following key differences:

- Initialization of NIS+ clients
- Propagation of updates from master to replicas
- Format of mastered data

Unlike NIS clients, which do not require any authentication, NIS+ clients must present credentials to gain access to the service. These credentials are stored in the client's home domain. NIS+ can be initialized with one of three methods:

- Broadcast
- Specified Server
- Cold Start File

The Broadcast and Specified Server methods are similar to what NIS clients do. The Cold Start File method provides a file to a client that contains information about how to locate directory objects and also provides a set of credentials. This is the preferred method since it provides additional security. Only a trusted server can provide a Cold Start File.

Instead of pushing entire maps no matter how many changes are made, NIS+ masters only push out incremental changes. These changes are batched, then pushed out. The result is that the replicas are more likely to be in sync. Also, a transaction log keeps track of changes in case of a system failure before they can be pushed out.

Unlike NIS where the mastered data is kept in text files, NIS+ keeps mastered data in a binary format. This means that not only do these files need to be backed up, they also need to be checked periodically for corruption.

Solaris DNS Architecture Overview

The Domain Name System was created to solve the problem of locating computers on ARPANET, the forerunner of the Internet. As more and more systems were added, resolving hosts names to IP addresses by means of text files became unworkable. Large `hosts` files had to be maintained and propagated to every system in the network. Today, DNS is a requirement for access to the Internet.

DNS Client Architecture

Solaris system utilities that access DNS do so by using the resolver on the client. The resolver is actually a set of library routines that perform various types of queries. These queries get information about the location of the DNS servers by looking in the `/etc/resolv.conf` file. The following shows the format of this file:

```
domainname mydomain.com
nameserver IPaddr1
nameserver IPaddr2
nameserver IPaddr3
```

As you can see, more than one DNS server can be specified. In normal operation the resolver tries to contact the first server in the list. If contact cannot be established, the second server is tried, then the third. The current limit in the Solaris operating environment is three.

DNS Server Architecture

DNS supports a hierarchal namespace and replica or caching servers. The namespace is separated into zones that can have primary and secondary servers. Primary servers act as masters from which information is updated and then pushed out to the secondary servers.

The Berkeley Internet Name Domain (BIND) is the name server (`named`) that runs on a designated host in your organization. Since there are different features that are available in different versions, it is helpful to know what version you are running. TABLE 2-4 correlates the BIND version with the Solaris operating environment that it appears in.

TABLE 2-4 Solaris Versions of BIND

Solaris OE Version	BIND Version
SunOS 4.x	4.8.1
SunOS 2.0-2.5	4.8.3
SunOS 2.6	4.9.4-P1
SunOS 5.7	8.1.2
SunOS 5.8	8.1.2

DNS High Availability Features

DNS provides features for making itself more available and also features for making applications more available. Caching servers, which contain the same information and are synchronized, can be configured. Multiple IP addresses can be listed for a specific host name in a DNS record. Each time a request is made for that host, the next IP address in the list is handed out. This technique is often referred to as round robin; it is useful when a DNS client is provided with the address of an application server that is not operational because, with the round robin technique, the client will try again and get a different address.

DNS servers can be clustered to provide automatic failover of master servers, although this feature is not part of the architecture. With this technology, updates to DNS records can still be performed in case the master DNS server fails.

LDAP Architecture Overview

The Lightweight Directory Access Protocol (LDAP) is the newest addition to the list of Solaris naming services. Although included in the Solaris 8 release, it is an optional naming service that can coexist with legacy Solaris naming services. LDAP shares some characteristics with NIS and NIS+, but it is more sophisticated in the way data is structured and the methods used to access data.

LDAP's complex architecture is easier to explain if we divide it into the four models it supports and describe each model separately, as we do in the following sections. The four models are:

1. Information Model
2. Naming Model
3. Functional Model
4. Security Model

Each of these models are discussed in the following sections.

LDAP Information Model

The LDAP information model defines how entries in the directory are organized in the directory. Entries are arranged in a tree-like structure called the Directory Information Tree (DIT). At the top of the DIT is the directory root, which is identified by the server name and port number on which the directory service is running. Multiple instances of the directory service can be running on the same server with each instance having its own DIT.

Below the directory root is the directory suffix, of which there may be several per DIT. Suffixes can be expressed as an organization (`o=`) or as an Internet style domain component (`dc=`). The LDAP predecessor, X.500, dictated a specific format which included a country, locality, and organization. These names were registered to avoid duplication. Since LDAP does not enforce the same stringent naming rules as X.500 any organization name can be specified. The domain-based format typically mirrors a company's DNS domain address and is expressed as domain component (`dc`) entries. Since most companies have a registered DNS name which ensures uniqueness, this format essentially replaces the old X.500 style format.

FIGURE 2-8 is an example of a DIT:

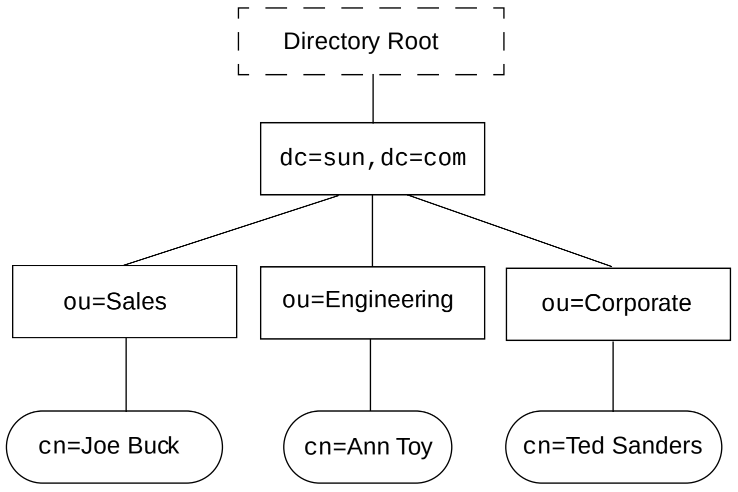

FIGURE 2-8 Sample Directory Information Tree

Located below the suffix are organization unit (`ou`) entries. These entries can nested, so an `ou` can contain other organization units. The name chosen for an `ou` only needs to be unique at the level at which it resides. You can use the same `ou` in a different portion of the DIT without creating a conflict. An `ou` entry called `ou=People` is created during the default iPlanet Directory Server installation; this entry is the default location for storing user account information, but any `ou` can be used for that purpose.

If you have multiple directory servers in a network, they can be linked by LDAP *referrals*. A referral is a mechanism that instructs an LDAP client searching the directory to continue the search on another directory server. The referral accomplishes this instruction by passing a uniform resource locator (URL) back to the client. Once the client receives the URL, it can access the specified directory server.

Overall, the topology of the directory resembles that of a Solaris file system. It is a hierarchal structure which has containers (`ou` entries) where directory entries reside. Referrals are similar to NFS mount points in concept, though implemented differently. Unlike the naming convention of a file system, that of an LDAP directory is quite different and the entries stored are much more complex than those in Solaris files.

LDAP Naming Model

Understanding the LDAP naming model is key to knowing how to configure and administer native Solaris LDAP. Most Solaris administrators are unfamiliar with this model and often are tripped up by some of the naming conventions. While the LDAP naming model may seem cryptic at first, keep in mind the goals of LDAP. It is designed to be flexible, but at the same time to provide a structure so that LDAP clients can access data in any LDAP-compliant directory.

Before a client can access data in a directory it must know how to locate that data. Unlike a Solaris file system where a search can always be initiated from the root file system (`/`), LDAP begins a search by specifying one specific entry, such as `dc=blueprints, dc=com`, as a search base. The entry name is specified as a distinguished name (DN) which is a series of relative distinguished names (RDNs). Each directory server contains a single root directory specific entry (DSE) which contains basic information about the LDAP server. The DSE is specified during base level searches on a directory when you do not know the name of a particular suffix.

As previously mentioned, each entry is identified by its DN. The DN is similar to a Solaris file system pathname, but is specified in the reverse order. However, for directory entries, unlike files, it is the value of their attributes which make each entry unique. To understand the role of attributes, a discussion on the structure of directory entries is useful.

Directory Objects and Attributes

The structure of a directory entry is defined by the object class to which it belongs. An object class defines a set of attributes that can be stored in a directory entry. LDAP object classes are extensible by creation of a new class that is a child of an existing one. All the attributes defined in the parent class are inherited by the child. The name of an object class must be unique within the directory server and can be registered as a standard LDAP object. These objects are assigned a numeric object identifier (OID) to ensure they will not conflict with another object class.

Attribute names are unique within the directory server and can be contained in more than one object class. The type of data that can be stored in an attribute is well-defined, as is the way LDAP searches treat the data. For example, data stored in a string can either be case sensitive or not. If the data is not case sensitive any combination of upper and lowercase characters in a string results in a match. If the data is case sensitive, an exact match is required. Attributes can also contain more than one value and can have aliases.

To promote interoperation, a set of standard LDAP object classes and attributes have been defined. Definitions of these ship with most LDAP servers in the form of *schema* configuration files. If they do not exist on a server, you can add the content of these schema files to your LDAP configuration files. For example, to use native LDAP, you need to add extra object classes and attributes to the iPlanet Directory Server configuration files, as discussed in Chapter 5, "Solaris 8 Native LDAP Configuration."

Directory Schema

The information specified in a directory schema includes the object class name, required and allowed attributes, an optional OID number, and the allowable syntax. TABLE 2-5 shows the schema definition for the `posixAccount` object class attributes that stores Solaris user account information.

TABLE 2-5 `posixAccount` Attributes

Attribute	**Description**	**Syntax**
`cn(commonName)`	Common Name of the POSIX account	`cis` (1-many)
`gidNumber`	Unique integer identifying group membership	`int` (single)
`homePhone`	The entry's home phone number	`tel`
`uid(userID)`	The user's login name	`cis`, 1
`uidNumber`	An integer uniquely identifying a user	`int`
`description`	A human-readable description of the object.	`cis`
`gecos`	GECOS comment field	`cis`
`loginShell`	Path to the login shell	`ces` (single)
`userPassword`	Entry's password and encryption method	`bin`, 1

In this example, `cn` is a case-insensitive string that can contain multiple values. The `gidNumber` and `uidNumber` are integers, and `homePhone` is represented by a special data type used for telephone numbers. Note that the LDAP `uid`, which is a string, is not the same as the numeric Solaris UID, which is represented by the LDAP attribute `uidNumber`. A complete description of all iPlanet Directory Server schema definitions can be found under documentation on the iPlanet Web site: `iplanet.com`.

Distinguished Names

Recall that a directory entry is identified by its DN, which is similar to a file system path name. However, entries are composed of many attributes, some of which are the same as other entries. To distinguish between entries that may have the same values for some attributes, one attribute is usually singled out as being unique. For user account entries defined in the `posixAccount` object class, that attribute is `uid`. To prevent duplicate values being used, the iPlanet Directory Server is configured by default to enforce `uid` attribute uniqueness. Entries that do not have a `uid` attribute are typically identified by the `commonName` (`cn`) attribute, which is available in most object classes, but is not required by all object classes such as organization (`o`) and organization unit (`ou`).

The form of a DN is:

attribute=value,container,suffix

where there may be multiple containers depending on the DIT topology. An example of a DN for an user account is:

```
cn=Cathy Miller,ou=People,dc=blueprints,dc=com
```

The RDN specifies the left-most portion of the DN, which uniquely identifies the entry relative to its parent. For example:

```
cn=Cathy Miller
```

In this case, `cn=Cathy Miller` has to be unique within the `ou=People` container.

LDAP Functional Model

Clients needing to access data on an LDAP server must begin by performing a bind operation. The bind operation requires, at a minimum, the DN of the user account entry the client wishes to bind as. If the entry has a password, then it is passed along with the DN. Alternatively, the client can perform an anonymous bind, which does not require a particular user name or password.

The type of authentication the directory server requires is specified as part of the bind request. The default is simple authentication, which compares the password sent with the password stored for the specified DN. Other authentication methods such as secure socket layer (SSL), CRAM-MD5, or Kerberos can be invoked instead by addition of another parameter to the bind operation call.

If the bind operation is successful, the client is considered authenticated. All subsequent client requests made on the connection established as a result of the bind are performed as the authenticated user. After the LDAP client requests are complete, an unbind operation is performed to release the connection. Chapter 5, ”Solaris 8 Native LDAP Configuration” describes how the Solaris LDAP client binds to an LDAP server.

Note – If an LDAP bind operation is made with a DN, with no password, the bind is successful, but is considered an anonymous bind.

LDAP Security Model

Access to LDAP entries on the server is protected by the rights established for the authenticated user. The rights can be assigned at the container, object, or attribute level. A portion of the DIT can be assigned stricter (or looser) control than other parts of the DIT. All entries of the same object class type can be assigned the same control. Control can also be established at the attribute level to protect certain information. For example, an employee’s password might have restricted access, while other information is available to everyone.

The mechanism used to assign access rights is called the access control instruction (ACI). A single ACI can protect the entire DIT, or several can be used to provide finer-grained protection. When multiple ACIs are created, the ACIs specifying `deny` access takes precedence. For example, if access is granted to everyone at the top level of the DIT but denied access to `ou=Contractors`, then the permissions set for `ou=Contractors` is enforced.

Note – ACIs are not defined in the LDAP v3 standard. Currently, each LDAP directory implementation has its own representation of ACIs.

Chapter 9, ”Preventive Maintenance” discusses how ACIs are created and provides a more in-depth explanation of how they work. Establishing the correct ACI is critical to configuring the iPlanet Directory Server to support native Solaris LDAP, so Chapter 5, ”Solaris 8 Native LDAP Configuration” provides examples. Note that the ACI syntax is not part of the LDAP specification, so the examples are specific to the iPlanet Directory Server implementation.

LDAP Replication

Replication is the mechanism by which directory data is automatically copied from one directory server to another. Using replication, you can copy anything from entire directory trees to individual directory entries between servers. Beside providing high data availability, some additional benefits include:

- Higher performance — By replicating directory entries to a location close to your users, you can vastly improve directory response times.
- Load balancing — By replicating your directory tree across multiple servers, you can reduce the access time load on any given machine, thereby improving server response time.
- Local data management — Replication allows you to own data locally and share it with other directory servers across your company.

To understand how replication works, you must first understand the roles LDAP servers play. To begin, every directory object must be mastered by one and *only* one directory server. The mastering directory server is called the *Supplier* server because it supplies the objects to other servers. Servers that receive directory objects from supplier servers are called *Consumer* servers.

Note – Any given directory server can be both a supplier of directory objects as well as a consumer of objects supplied to it from other servers. In future releases of iPlanet Directory Server, multi-master replication is supported which allows directory data to be updated by more than one server.

A Supplier server is responsible for the following:

- Managing any requests for changes to the replicated directory data. That is, whenever a request to add, delete, or change an entry in a replicated tree is received by a Consumer server, the request is referred to the Supplier server where the request is actually performed.
- Tracking the changes to the objects that it masters so that those changes can be replicated to Consumer servers.

You can configure the Supplier server to initiate replication, or you can configure your Consumer server(s) to initiate the replication process.

Consumer servers contain at least one directory entry that has been copied to it by a supplier server. Consumer servers can contain the following:

- The Supplier server’s entire tree.
- A subsection, or subtree, of the Supplier server’s directory tree.

Only read operations occur on the Consumer server. All other operations are handled on the Supplier server. Whenever an LDAP client tries to modify entries in a replicated tree, the Consumer server automatically refers the LDAP client’s request to the supplying server.

FIGURE 2-9 and FIGURE 2-10 are examples of replication configurations:

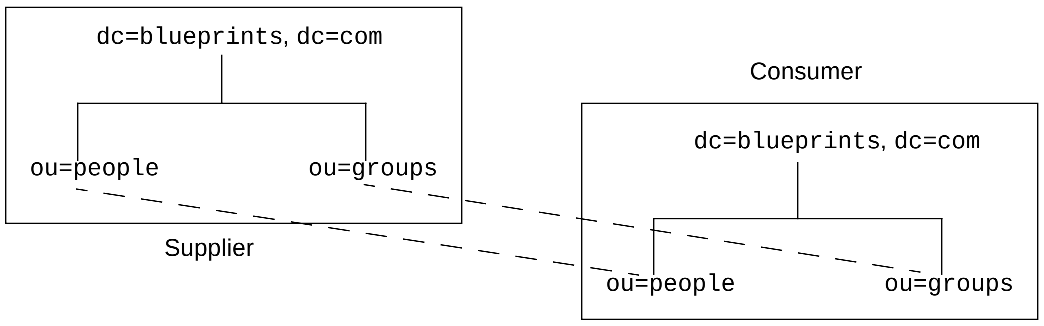

FIGURE 2-9 Full Tree Replication

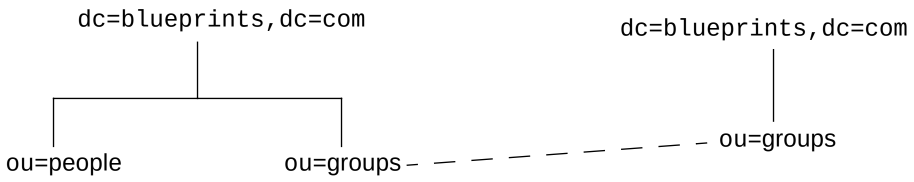

FIGURE 2-10 Subtree Replication

You choose which form of synchronization is used for each replication agreement. Replication synchronization can be initiated by either the Supplier or the Consumer server. A replication agreement indicates which directory entries will be replicated, which servers are participating in the replication, and when the replication can occur.

To decide on a synchronization method, follow these guidelines:

- If you want your consumer servers to be updated instantly, use Supplier-initiated replication.
- If you are using a dial-up connection to update your Consumer servers, use Consumer-initiated replication.

Comparison with Legacy Naming Services

Naming, or directory, services technology has evolved with the rise of network computing as the central concept of information technology. Host-based naming services, such as DNS, are widely deployed and have provided a key component of the network infrastructure in place today. Desktop LAN-based naming services like NIS have enjoyed much success in Solaris and UNIX environments but have not been widely accepted outside of these environments. Standards-based LDAP directories are starting to gain wide acceptance and look to be the backbone of corporate directory infrastructures in the future.

With so many Solaris naming services available, it is not always easy to keep the differences straight. To help you out, TABLE 2-6 summarizes the key features found in each of the naming services discussed in this chapter.

TABLE 2-6 Naming Service Feature Comparison

Naming Service	Hierarchal DIB	Dynamic Updates	Distributed DIB	Dynamic Replication	Extensible DIB
NIS					
NIS+	X	X	X	X	
DNS	X	X	X	X	
LDAP	X	X	X	X	X

Hierarchal Directory Information Base (DIB) — The ability to organize the name space in a layered, tree-like structure.

Dynamic Updates — The ability to add, modify, and delete information in the name space and have those changes be immediately visible to users of the service.

Distributed Directory Information Base (DIB) — The ability to service the namespace from multiple nodes on the network.

Dynamic Replication — The ability to dynamically propagate changes made to the DIB to other nodes that serve the DIB.

Extensible Directory Information Base — The ability to dynamically expand the type of information stored as part of the namespace.