Chapter 1

NDS Concepts

Overview

This chapter defines basic concepts used in Novell Directory Services (NDS). This chapter will discuss the following concepts:

- Client/server architecture
- Bindery services
- Global, distributed, replicated name service
- hierarchical naming
- Partitioning
- Replication
- Name server
- Synchronization
- Loose consistency
- Schema
- Object classes
- Super classes
- Containment
- Inheritance
- Authentication
- Background authentication

Note These concepts are not presented in alphabetic order but are grouped together in the logical order of the NDS discussion presented in this chapter.

You should read this chapter if you are not familiar with the basic concepts of Novell Directory Services.

Note NDS is also referred to as Directory Services.

Networking and NDS Concepts

Novell Directory Services changes the manner in which networks have been traditionally used and administered. Previously, networks were organized around specific servers; if you wanted to access a particular service, you needed to access the server that held those services. This meant that you had to know the server's name and address, and have a password stored on that server.

Instead of providing a server-centric environment, Novell Directory Services provides a network-centric environment. The Directory database provides a view of all services available on the network; once you log in to the network, you have access to all services you have rights to access. This means that the network administrator must apply new concepts and techniques to network administration. Novell Directory Services provides the tools for administrators and applications to manipulate the new network environment.

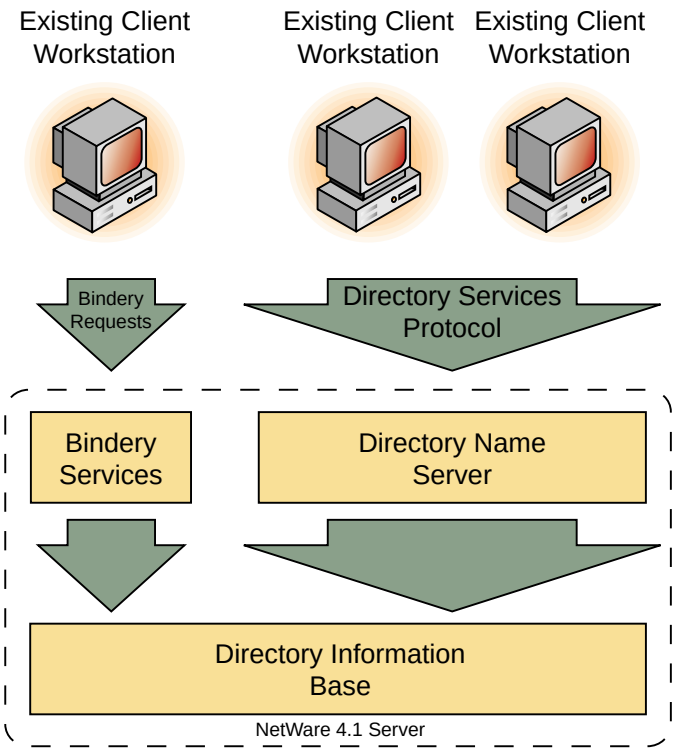
Novell Directory Services is based on the 1988 CCITT X.500 standard on directory services. This standard outlines a paradigm in which network objects are held on independent machines. This chapter will discuss the concepts involved in Novell Directory Services and how they relate to the NDS environment.

Client/Server Model

Novell Directory Services is built on the network client/server model. That is, one or more Directory servers are attached to one or more workstations (clients). At the client workstation, a user application invokes a client agent that formulates Directory requests on the user's behalf. The client agent establishes a session with a Directory server to which it submits the requests.

Client agents and Directory servers use the NDS protocol when they need to make access requests. Client workstations can also access some Directory information by using bindery services.

Figure 1.1
NDS Client/Server Architecture



Note The X.500 Directory Services standard refers to a client agent as a directory user agent (DUA). It also refers to Directory servers as Directory system agents (DSA).

Bindery Services

Because Novell Directory Services was originally NetWare-specific, it was written to be backwards compatible with earlier releases of NetWare. that used the bindery to store the database information. Currently, the NDS name space is used to store this information. It relies on bindery services to provide backward compatibility to applications that use NetWare bindery calls. Bindery services formulate Directory information into a bindery format that the server can use to respond to bindery-style requests.

Note Bindery services are limited; they allow backward compatibility but cannot express all of the information stored in the Directory.

NDS Name Service

NDS provides a global, distributed, replicated name service. NDS shifts the networking paradigm from one in which the entire database is stored

on one server to one in which the database is split across a network of servers. Thus, the NDS name service encompasses all servers and resources in the network. In other words, NDS allows the database to change while giving users across the network a consistent view of the Directory.

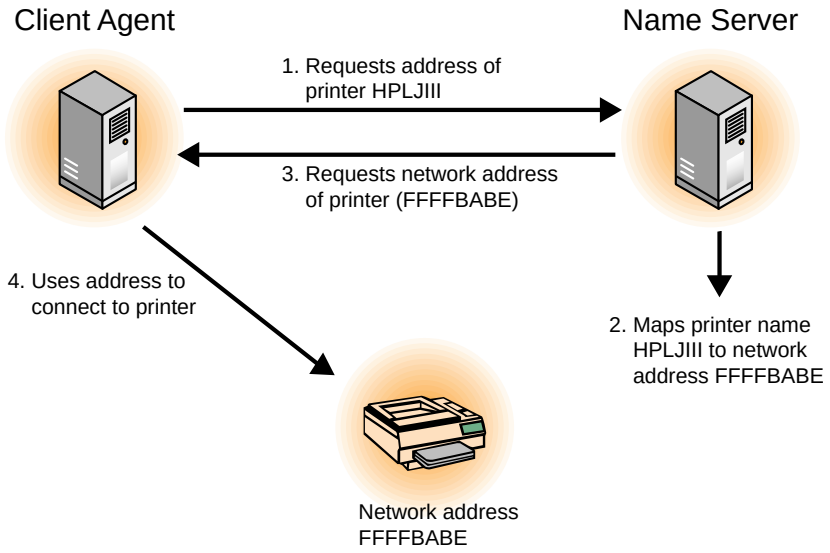
NDS is also described as a distributed database because portions of the database are placed among the Directory servers according to corporate needs, network use, and access time, among other factors. These portions of the database are called *partitions*. Partitions help save server disk space because servers do not have to store all of the Directory information.

These Directory partitions are replicated on different servers throughout the network tree. Among other uses, these replicas of the partitions provide fault tolerance and easier access to network resources.

Cooperation among Directory servers allows users to access the Directory as a single resource. This coordination of multiple partitions and replicas is referred to as the distributed operation of the Directory.

One of NDS's main functions is to provide a name service. A name service translates network or resource names to network addresses. This means that any modification to the network address of a resource or object is invisible to the user; the user is still able to access the resource by the resource's name. See Figure 1.2 for a basic illustration of this concept.

Figure 1.2
Mapping a Network Address to a Resource Name



Name Space

Novell Directory Services defines the name space (or naming structure) for the entire network. A name space is a set of rules that defines how all network users and resources are named and identified.

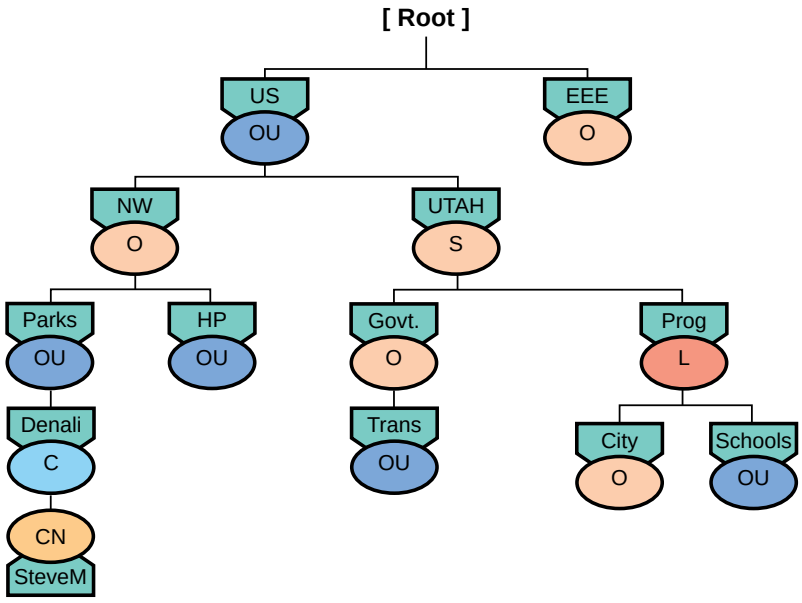
NDS uses an *n*-level hierarchical name space instead of the traditional flat name space such as the bindery. A flat name space contains one name for each object but does not include any information about the object's relationship with the rest of the database. For example, a printer object could be named HPLJIII. That name identifies a printer as HPLJIII, but it does not identify a specific physical printer or that printer's logical location in the database.

A hierarchical object name resembles a complete file name in a hierarchical file system such as DOS. A complete DOS file name contains each directory in the path from the drive root to the file; a complete hierarchical object name contains the name of every container object between itself and the root of the database.

NDS uses the hierarchical name space because an average network has about five to seven objects in the directory per user account. These objects provide the basic functions and resources of the network, such as printing, e-mail and user accounts. In a corporation of 5,000 employees, the NDS database size might consist of 25,000 to 35,000 objects. This database would also grow as its environment grows and more users and resources are added to the network. A flat naming space in a database of this size becomes very difficult to administer and scale.

An *n*-level hierarchical name space is usually represented as a tree structure (much like the representation of a hierarchical file system). Figure 1.3 shows an example of a user object SteveM and its relationship to its container objects in a tree form

Figure 1.3
Example NDS Tree



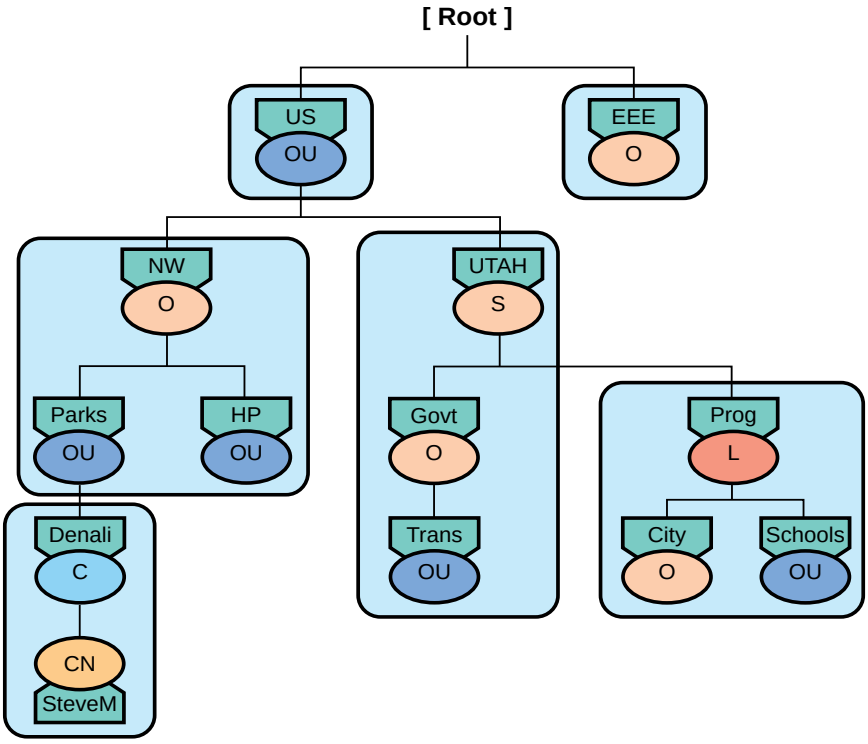
The name of user object SteveM shows its relationship to the database hierarchy and indicates its logical location as follows:

CN=SteveM.OU=Denali.OU=Parks.O=NW.C=US

Unlike the long file names in a hierarchical file system, an NDS object name starts with the most significant part of the name and proceeds to the most general part. Thus, the name of object SteveM, starts with the user's name (the most significant part) and traces its relationships back to the Tree Root object (the most general part).

Because a hierarchical name contains the logical relationship of the object to its environment, a hierarchical name structure allows NDS to split the name space into logical groupings of objects. These groupings become the *partitions* of the NDS tree (see Figure 1.4)

Figure 1.4
Example Partitioned NDS Tree



Partitioning allows organizations to group objects together to reflect their needs, locations, and organization. When these needs change, the Directory can be modified to reflect the change.

Distinguished Name

An object's Distinguished Name (DN) is a unique reference that identifies an object's location and identity in the Directory. The name .CN=SteveM.OU=Denali.OU=Parks.O=NW.C=US, for example, isolates Steve M's user object to only one object in the entire Directory because it contains a unique path of container object names.

Each object name has a naming attribute, which in the base schema is associated with either a container or a leaf object. For example, the naming attribute for a leaf object is usually CN= (Common Name). Container objects generally have one of the following naming attributes:

- C= Country
- L= Locality
- S= State
- O= Organization
- OU= Organizational Unit

In the example above, the Denali object has the naming attribute of OU, which in the base schema means that it is an Organizational Unit object.

A Distinguished Name that contains the type identifiers is also known as a *typeful name*. “OU=Parks.O=NW.C=US.T=GovTree” is an example of a typeful name. Distinguished Names can also use just the “.” between the object names and drop the object types. These names are called *typeless names.* “SteveM.Denali.Parks.NW.US.GovTree” is an example of a typeless name.

Relative Distinguished Name

The individual name assigned to an object is called the object’s *Relative Distinguished Name (RDN),* or partial name. The partial name must be unique in relation to the object’s superior object. In the above example, only one object named Denali can be subordinate to Parks, and only one object named SteveM can be subordinate to Denali, and so on.

Name Context

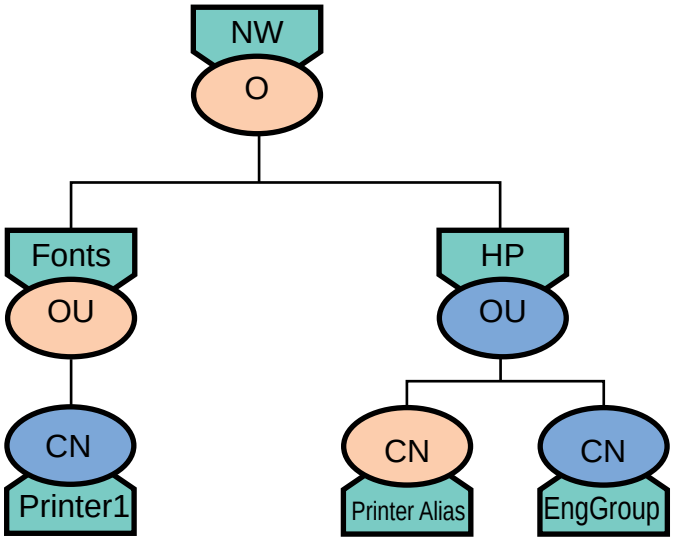
NDS requires that an object be specified by its complete name. Because it can be inconvenient to identify an object by its complete name, NDS allows a client to submit a portion of its complete name. This portion must be relative to a predefined *name context.* The name context is a complete name that serves as a default naming path for the operation. (It is the complete name of a container object; it is not the user’s complete name.)

Alias Objects

An *alias* is an object that points to another object in the Directory tree. Although aliases represent other object classes, they are used differently than other classes of entries. Each alias points to one object in the tree, which can be either a leaf or a container object.

For example, you can use an alias to allow users in one container to access a resource that is in another container. Figure 1.5 shows an example of how this could be done.

Figure 1.5
Alias Object



The alias, PrinterAlias.HP.NW, points to the real printer object, Printer1.Parks.NW. If the members of the EngGroup Group object have rights, they can use the PrinterAlias object to access the printer, even if the real object's name is changed.

You could give an Organizational Unit (OU) an alias to another OU. This would give the first OU the right to share the other's resources.

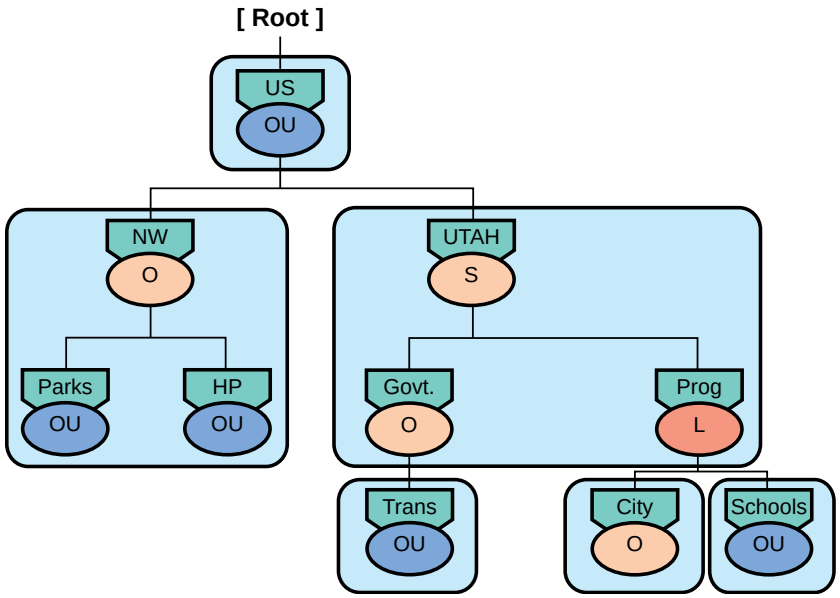
Although an alias object's base class is Alias, its actual entry record holds the base class of the object it points to; the base class in the entry record is then flagged as an alias.

A client can choose to "dereference" an alias, which means that the server will return information about the aliased object, not about the alias itself. In the *Resolve Name* operation, all aliases in the Distinguished Name are automatically dereferenced, except the leaf-most object. The "Dereference Alias" flag in the *DSAResolveName* request refers only to that leaf-most entry.

Partitioning

Each partition contains a subtree of the entire Directory tree. When taken together, all the partitions form a hierarchical tree of partitions leading back to a root partition that contains the Directory root (see Figure 1.6).

Figure 1.6
A Hierarchical Tree and Its Partitions



In Figure 1.6, the US partition is the parent of the NW and Utah partitions, who are its child partitions. The Utah partition is the parent of the Trans, City, and Schools partitions, who are its child partitions.

Note Where the boundaries of two partitions meet, the partition closer to the root is considered the *parent*, or superior, partition, and the one further from the root is considered the *child*, or subordinate, partition.

Important This tree of partitions is invisible to the typical Directory user, who sees only a global tree of Directory objects.

Partitions must obey the following three rules:

- They must contain a connected subtree
- They must contain only one container object as the root of the subtree
- They cannot overlap with any other partition in the NDS tree

A partition is named by its root-most container object (the container at the root of the subtree). This is called the *partition root.*

Replication

A single instance of a partition is called a *replica.* Partitions can have multiple replicas, but only one replica of a particular partition can exist on each server. (Keep in mind that servers can hold more than one replica, as long as each replica is of a different partition.) One of the replicas (usually the first created) of a given partition must be designated the master replica. Each partition can have only one master replica; the other replicas

are designated as either read/write or read-only replicas. (You can use the read-only replica only to read the information in the partition replica. You cannot write to a read-only partition.)

Replication adds fault tolerance to the database because the database has more than one copy of its information.

Name Server

Physically, a name server is a network node that administers zero or more Directory replicas. It provides name services for a Directory tree.

Logically, a name server is represented as an object in the Directory tree. This object is an instance of the NCP Server Object Class defined in the NDS schema.

Synchronization

Synchronization is the process of ensuring that all changes to a particular partition are made to every replica of that partition. The X.500 standard defines two synchronization mechanisms: master/slave synchronization and peer-to-peer synchronization.

The master/slave mechanism requires that all changes be made on the master replica. That replica is then responsible to update all the other replicas (slave replicas).

In a peer-to-peer synchronization system, updates can be made to any read-write or master replica. At a predetermined interval, all servers holding copies of the same partition communicate with each other to determine who holds the latest information for each object. The servers update their replicas with the latest information for each replica.

NDS uses both the master/slave and peer-to-peer synchronization processes, depending upon the type of change that is being made. The master/slave mechanism synchronizes operations such as partition operations that require a single point of control. The peer-to-peer mechanism synchronizes all other system changes.

Note In NetWare, the synchronization time interval ranges from between 10 seconds (fast synchronization) to 30 minutes (slow synchronization) depending upon the type of information updated.

Loose Consistency

Because the NDS database must synchronize replicas, not all replicas hold the latest changes at any given time. This concept is referred to as *loose consistency*, which simply means that the partition replicas are not instantaneously updated. In other words, as long as the database is being

updated, the network Directory is not guaranteed to be completely synchronized at any instance in time. However, during periods in which the database is not updated, it will completely synchronize.

Loose consistency has the advantage of allowing Directory servers to be connected to the network with different types of media. For example, you could connect one portion of your company's network to another by using a satellite link. Data travelling over a satellite link experiences transmission delays, so any update to the database on one side of the satellite link is delayed in reaching the database on the other side of the satellite link. However, because the database is loosely consistent, these transmission delays do not interfere with the normal operation of the network. The new information arrives over the satellite link and is propagated through the network at the next synchronization interval.

Another advantage to loose consistency is that if part of the network is down, the changes will synchronize to available servers. When the problem is resolved, the replicas on the affected servers will receive updates.

Note The X.500 standard refers to the concept of *loose consistency* as *transient consistency*.

Directory Schema

The directory schema defines the rules for adding entries to the Directory. A data dictionary specifies the rules and provides a standard set of data types from which objects can be created.

Every object in the Directory belongs to an *object class* that specifies the attributes that can or must be associated with the object. All attributes are based on a set of standard *attribute types* that, in turn, are based on standard *attribute syntaxes*. *Object classes*, *attribute types*, and *attribute syntaxes* are all defined by the Directory schema.

The Directory schema not only sets the rules for the structure of individual objects, but it also sets the relationships among objects in the Directory tree. To do so, the schema specifies *subordination* among object classes. Every object has a limited group of object classes to which it can be subordinate.

By limiting the potential pairs of superior and subordinate object classes, the schema provides a stable yet flexible structure for developing the Directory tree. The base schema is the schema created when the server is installed. However, the NDS schema is extensible, meaning that new information types can be added to the existing ones. Applications cannot subtract from the definitions provided by the base schema.

Object Classes

Object classes are used as the principle templates for storing information in the Directory. All Directory objects must belong to an object class. A class is defined by the types of attributes that characterize an object. These attributes types contain the following types of information:

- Super classes
- Containment
- Mandatory attributes
- Optional attributes
- Effective status
- Default Access Control Lists (ACLs)

Super Classes

Each new object class must have a *Super Classes* list. The super classes contain the schema inheritance rules. To determine the complete set of rules for a given object class, you must look at that class's super classes. An object class inherits all the features of its super classes. This is how hierarchies of classes develop through class inheritance. The classes at the top of a hierarchy provide general characteristics, while those at the bottom become more and more specialized. The complete set of rules for an object class is called the *expanded class definition.*

The class from which an object is created is called the object's *base class.* The information associated with an object includes the base class and the sum of information specified by all its super classes. When the Directory is searched, an object is considered a member of all its super classes. For example, the base class for the User object class is Organizational Person. The User object class also inherits information from the Person and Top classes.

Containment

Novell Directory Services employs a concept called *containment* that is very similar to the concept of subdirectories in a hierarchical file system. Some objects, such as Organizational Units, can contain other objects, such as User or resource-type objects. These objects are called container objects. The Directory tree is formed by container objects that hold other container and leaf objects in a specific organization. Often this organization reflects the corporate structure.

The containment classes and naming attributes for an object class constitute the classes' structure rules. In other words, the containment classes determine where the object can appear in the Directory tree.

Objects that cannot contain other objects are called *leaf objects*. Leaf objects are the network resources, such as User and Printer objects.

Inheritance

A superior (or containing) object class passes its attributes to a subordinate object class. This ACL attribute passing is called *inheritance.*

Objects can inherit attributes (but not attribute values) from their containing objects and super classes, and they can inherit the rights that belong to those objects and classes.

Authentication

Authentication provides verification that any requests the server receives are from valid clients. Authentication has two phases:

- Login — Obtains the private key.
- Authentication — Uses a signature to generate a proof that is used to authenticate (establish identity).

Authentication is invisible to the user. During login, the user enters a password, and the remainder of the operation is performed in the "background" by the authentication functions.

Authentication is session-oriented. The data that provides the basis of authentication is valid only for the duration of the current login session. The critical data used to create authenticated messages for a particular user is never transmitted in plain text across the network.

Authentication relies on encryption systems, or procedures that allow information to be transmitted in unreadable forms. NetWare 4 uses a private/public key form of encryption. Basically public/private key encryption means that the agent to receive messages generates a key pair. The receiving agent then distributes the public key and keeps the private key. Those that want to send encrypted messages use that public key to encrypt the message, and the receiver then uses the private key to decipher the message. Only the holder of the private key can decipher these messages. Conversely, a sender can encrypt data with the private key. The recipients of this message would then use the public key to decrypt the message. If the decryption is successful, the recipient can be sure that the message was encrypted with the corresponding private key. Only the holder of the private key could have generated the message. In this case, many agents can decrypt the message and be assured that the message is authentic.

Background Authentication

Background authentication refers to authentication to additional services subsequent to the initial login operation. The user asks for particular information or resources, and NDS finds the service providing the

resource, authenticates to it, and returns what was desired. The user does not have to retype a password.

Default Access Control List Templates

An *Access Control List* (ACL) is an optional property on each object that determines which operations a trustee can do on that object. Most object classes have a default ACL template assigned to them if no ACL is specified when the object is created. This default template provides a minimum of functionality and access control for the new object. p