Appendix A

X.500 and NDS: How Close Are They?

Overview

This appendix discusses the compatibilities between X.500 and NDS. It addresses an audience of network administrators and developers that is acquainted with directory services concepts.

Introduction to X.500 Directory Services and NDS

Network administrators have always had to invest considerable time in configuring and managing network resources and services. Traditionally, these services have been tightly integrated into the operating system or into applications such as email. However with networks achieving global scale, administrators have had to become even more concerned with managing network resources. So, replicated and distributed directories have become important because they simplify and streamline network administration. Also, as global networking has become more common, companies have seen the advantage to compatible networks on this scale. This led to the definition of a directory services standard. The X.500 directory services specification has become one of the most accepted standards for duplicated, replicated databases and directory services. *Novell Directory Services* (NDS) is functionally equivalent to, and can coexist with X.500, compliant directory services. NDS's main purpose is to manage network resource configuration information.

NDS is a full-fledged directory service modeled after X.500. But NDS has different goals. Currently, X.500 directory services are primarily viewed as a way to store email addresses. NDS, on the other hand, is currently used to administer users, servers, services, and network resources. NDS allows administrators to effectively manage global networks on a host by host or domain by domain basis. Because of NDS's broader scale, it must be flexible enough to allow the network administrator to enhance the network environment, without bringing the network down. In other words, while X.500 is currently being used as a global postmaster, NDS is designed to operate, and is currently operating as, a global services registry. A directory service implementation requires the following elements:

- The Directory Information Base (DIB)
- Authentication and authorization methods
- APIs and protocols

This paper discusses the functional compatibilities between the NDS and X.500 directory services in these areas and concludes with a statement of NDS's future direction.

Functional Comparisons

Table A.1 briefly compares NDS with X.500 directory services.

Table A.1
Comparison of X.500 Directory Services and NDS

Directory Services Features and Functions	X.500 1988	X.500 1993	NDS
Directory Information Base (DIB)	Distributed and replicated. Changes propagated from one server.	Distributed and replicated.	Distributed and replicated. Changes propagated from any server.
Information Storage Model	Identical	Identical	Identical
Directory Information Tree (DIT) or Namespace	Hierarchal and identical.	Hierarchal and identical.	Hierarchal and identical.
Aliases	Allowed	Allowed	Allowed
Metaschema	Compatible	Compatible	Compatible
Schema	Compatible	Compatible	Compatible
Extensible schema	Yes, but typically must edit file and down the server to reload schema. Includes compound attributes.	Includes dynamic compound attributes.	Yes. Can extend schema without downing network. Defines hard-coded attribute syntaxes in the place of compound attributes.
Object Classes	Defined in X.521 standards.	Defined in ITU recommended X.521 standard (ISO 9594-7).	Objects identical to those defined by X.521. Also defines additional objects. String representations use a 2-byte character representation for unicode.
Attributes	Defined in X.520 standards. Uses OID.	Defined in ITU recommended X.520 standard (ISO 9594-6).	X.520 compliant. Also defines additional attributes. String representations use a 2-byte character representation for unicode. Does not support OID.
Attribute Syntaxes			String representations use a 2-byte character representation for unicode.
Security	Uses X.509 standards.		Uses RSA GQ algorithm.
APIs	Uses XDS APIs.		Compatible with XDS APIs.

Table A.1
Comparison of X.500 Directory Services and NDS (Continued)

Directory Services Features and Functions	X.500 1988	X.500 1993	NDS
Protocols	Large and slow.		Defined smaller, faster protocols.
Protocol Verbs	Supports 8 verbs.		Supports 75 verbs.
Logical Operations	Identical		

The Directory Information Base (DIB)

The X.500 specification describes a Directory Information Base (DIB) that contains information about the objects stored in the directory. The DIB is composed of directory entries, each of which is made up of attributes that contain information about that entry. The attributes each entry contains depend upon the object class of the entry.

The DIB entries are arranged in the form of a logical tree structure called the *Directory Information Tree* (DIT). Entries in the containers and branches of the tree usually represent countries and organizations. Entries in the tree leaves usually represent people, services, applications and physical objects.

NDS uses an identical DIT/DIB structure.

Namespaces and Tree Structures

The X.500 specification mandates that each entry have a distinguished name that uniquely and unambiguously identifies the entry. The DIB tree structure lends the entry its distinguished name: the object's name is composed of each container's name in the object's path back to the root.

X.500 also allows *alias entries.* An alias entry points to an object entry. That is, the alias name maps to a distinguished name, allowing users and administrators to more easily access objects.

The NDS and X.500 namespaces are the same. Both use identical naming schemes and both allow alias entries.

Metaschema

X.500 and NDS use identical metaschemas. A metaschema is briefly defined as the way you describe how you describe objects. For example, an NDS object class is defined with the following kinds of information:

- Structure Rules
- Object Class Flags
- ACL Templates

- Super Classes
- Object Class Attributes

That definition is the metaschema of an NDS object class. A base schema object class is created by defining values and attributes for each of the information types. Attributes and attribute syntaxes are also governed by metaschemas.

NDS does not support the X.500 compound attributes and sequence syntaxes. NDS does define a string syntax which X.500 does not. This syntax allows NDS to represent strings in a 2-byte Binary Multiple Plain (BMP) character representation which supports unicode. The 1993 X.500 implementation supports a universal string format. As a result, NDS can represent any X.500 or T61 string value, but neither the 1988 or 1993 X.500 implementations can represent all the unicode values. (Unicode is equivalent to the ISO 1046 universal character set.)

Schema

The X.500 and NDS schemas are also compatible.

Both X.500 and NDS use extensible schemas. That is, both specifications allow network administrators and developers to define new objects and expand the base schema.

The X.520 Standards. X.500 directory services uses a schema described by the X.520 specification. The X.520 specification describes attributes and attribute definitions. The NDS schema is also X.520 compliant. In addition, the NDS schema has added attributes. As mentioned under the metaschema, the NDS schema uses a 2-byte character representation for unicode.

The X.521 Standards. The X.521 specification defines object classes. Some of the NDS objects are identical to those defined by X.521. In addition, the NDS schema has added object classes required by the common NDS implementations. For example, the X.520 specification describes a residential person. NDS does not have much need for a residential person, so the NDS schema defines a user that is more business oriented. However, developers can easily add a residential person object class.

NDS also requires object classes that X.521 did not explicitly define. NDS handles this with macros that represent information required by NDS, but not defined by X.521. NDS uses a very similar class hierarchy as that described by X.521. X.500 directory services use object identifiers (OIDs). An OID allows you to assign numbers to unique object classes, syntaxes and attributes with the absolute assurance that the number will not be assigned to any other object class, syntax, or attribute. For example, Novell is registered as an organization and has been assigned the OID XXX. Whenever that OID is used to access an object class, that object

class will always be Novell, Inc. NDS does not currently use OIDs. However, it will use them in future implementations.

X.500 Schema Compatibilities. NDS uses X.500 attributes with X.500 attribute syntaxes. In addition, the NDS schema is flexible enough that you can add to the schema without taking the system down. NDS allows the administrator to add to the schema on the fly and while the network is running.

As discussed above, the NDS schema is very compatible with X.500 directory services. NDS has also augmented the X.500 schema by adding additional object classes and attributes that allow a network operating system to transparently handle objects and that also allow NDS to address the business needs of Novell's customers.

Partitioning and Replication

NDS differs from X.500 in its synchronization scheme. The X.500 standards specify that updated directory information be propagated from one server. In reality, this model is too unwieldy to synchronize a globally distributed and replicated network. Therefore, NDS allows updated information to be propagated from any server. However, this difference does not affect NDS compatibility with X.500.

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 10 seconds to 5 minutes depending upon the type of information updated. s

Authentication and Authorization

X.509 Authentication

X.500 directory services uses an authentication algorithm defined by the X.509 standards. This algorithm is the RSA authentication algorithm

which is based on public and private keys and provides a great deal of security. The X.509 authentication standards have two major disadvantages for NDS:

- Services that use the X.509 standards must allow users constant access to their private key
- Products using X.509 are not always exportable

Private Key Access

Because NDS currently addresses mainly DOS and Microsoft Windows workstations, and because both DOS and Windows have memory restrictions and limited memory security, our customers requested that NDS not leave the private key in memory. In response to our needs, RSA supplied the Guillou-Quisquater (GQ) zero-knowledge proof of identity scheme. GQ is a dual signature algorithm. In essence, this algorithm uses a secret key derived from another secret key. This enables NDS to use the private key to sign the authentication credentials, then discard it. NDS then uses a key derived from the private key to authenticate. NDS does not use this key to encrypt or decrypt information because the key is temporary. GQ also allows NDS the convenience of multiple keys and proxy authentication.

Issues with Exporting and Importing

Ironically, NDS cannot use X.509 authentication methods because of their excellent information encryption and decryption capabilities. Because of potential national security risks, the National Security Agency (NSA) does not always allow companies to export or import products that use these algorithms. So, even though the X.509 security algorithms are excellent, NDS will continue to use the GQ security algorithms and will use the X.509 capabilities primarily as add-on services and applications.

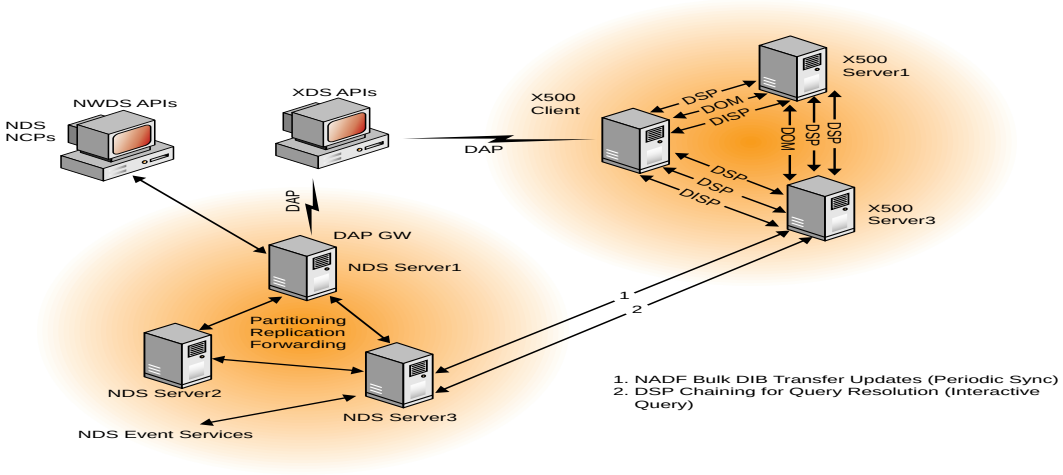
X.509 Access Control List

The X.509 standards define a very powerful and flexible access control list. NDS's access control list is just as powerful and flexible.

APIs and Protocols

Both X.500 directory services and NDS use multiple protocols to access the same information. In fact, a network using NDS and a network using an X.500 directory service can coexist, as shown in Figure10.1.

Figure10.1
Coexisting X.500 and NDS Networks



NWDS APIs and the X.500 Protocols

Although the X.500 specification defines extremely powerful and adaptable protocols, NDS defines its own protocols that are compatible with the XDS protocols. The X.500 protocols are very big and very slow. Because NDS currently addresses mostly DOS and Windows machines, it has to fit in 640 Kilobytes of memory. We cannot load the OSI stack and have room in memory for anything else.

So, we developed a lighter weight version of the protocol. We developed our protocol at about the same time that LDAP (Lightweight Directory Access Protocol) was being developed. Our protocol is very LDAP-like. However, LDAP uses *printf* to place all the attributes onto the wire and then retrieves them from the wire with *scanf*. Our protocol uses binary representation, not string representation which allows it to be smaller and faster.

NWDS and XDS APIs

The NWDS APIs are really a superset of the XDS APIs, and the NWDS API arguments map directly to the XDS protocol arguments with one exception: the XDS protocols include a *session* argument that the NDS protocols do not. This *session* parameter is basically a file pointer that holds an open connection. XDS uses this parameter to deal with some general overhead. We thought that our bindery users would be inconvenienced with this overhead, so we decided to not trouble our users with it and put the overhead in our libraries so that NDS could be more self-reliant. Table A.2 illustrates the relationship between the XDS and NWDS APIs.

Protocol Verbs

NDS supports 75 verbs. X.500 allows 8 verbs.

Logical Operations

NDS's logical operations come straight from X.500 specification. NDS uses search and list filters exactly the same way as X.500.

Table A.2
Relationship Between the XDS and NWDS APIs

XDS Function	XDS Parameters	NWDS Function	NWDS Parameters	Comments
ds_abandon	OM_private_object session; OM_sint invoke_id;	none		NDS supports only synchronous calls
ds_add_entry	OM_private_object session; OM_private_object context; OM_object name; OM_object; OM_sint *invoke_id_return;	NWDSAddObject	NWDSContextHandle context; char *objectName; NWDS_ITERATION NWFAR *objectInfo; NWFLAGS more; NWDS_BUFFER NWFAR *ObjectInfo;	context ↔ context name ↔ objectName entry ↔ objectInfo
ds_bind	OM_object session; OM_private_object *bound_session_return;	NWDSAuthenticate	NWDSCONN_HANDLE Conn; NWDS_FLAGS OptionsFlag; NWDS_SESSION_KEY *sessionKey;	uses strong authentication to bind a connection
ds_compare	OM_private_object session; OM_private_object context; OM_object name; OM_object; OM_private_object *restructuring; OM_sint *invoke_id_return;	NWDSCompare	NWDSContextHandle context; char NWFAR *objectName; NWDS_BUFFER NWFAR *buf; NWFLAGS NWFAR *matched;	context ↔ context name ↔ objectName ava ↔ buf result_return ↔ matched
ds_initialize	void - returns OM_workspace	NWDSCreateContext	void - returns NWDSContextHandle	Not an exact mapping. Both must be called before any other functions.
ds_list	OM_private_object session; OM_private_object context; OM_object name; OM_private_object *result_return; OM_sint *invoke_id_return;	NWDSList	NWDSContextHandle context; char NWFAR *objectName; NWDS_ITERATION NWFAR *iterationHandle NWDS_BUFFER NWFAR *subordinates;	context ↔ context name ↔ objectName result_return ↔ subordinates
ds_modify_entry	OM_private_object session; OM_private_object context; OM_object name; OM_object changes; OM_sint *invoke_id_return;	NWDSModifyObject	NWDSContextHandle context; char NWFAR *objectName; NWDS_ITERATION NWFAR *iterationHandle NWFLAGS more; NWDS_BUFFER NWFAR *changes;	context ↔ context name ↔ objectName changes ↔ changes

Table A.2
Relationship Between the XDS and NWDS APIs (Continued)

XDS Function	XDS Parameters	NWDS Function	NWDS Parameters	Comments
ds_modify_rdn	OM_private_object session; OM_private_object context; OM_object name; OM_object new_RDN; OM_object delete_old_RDN; OM_sint *invoke_id_return;	NWDSModifyRDN	NWDSContextHandle context; char NWFAR *object; char NWFAR *newDN/ *newRDN; NWFLAGS deleteOldRDN;	context ↔ context name ↔ objectName newRDN ↔ newDN/ newRDN delete_old_RDN ↔ deleteOldRDN
ds_read	OM_private_object session; OM_private_object context; OM_object name; OM_object selection; OM_private_object *result_return; OM_sint *invoke_id_return;	NWDSRead	NWDSContextHandle context; char NWFAR *objectName; NWDS_TYPE info-type; NWFLAGS allAttrs; NWDS_BUFFER NWFAR *attrNames; NWDS_ITERATION NWFAR *iterationHandle NWDS_BUFFER NWFAR *objectInfo;	context ↔ context name ↔ objectName selection ↔info_type, allattrs, attrNames result_return ↔ objectInfo
ds_search		NWDSSearch	NWDSContextHandle context; char NWFAR *baseobjectName; NWDS_SEARCH_SCOPE scope; NWFLAGS SearchAliases; NWDS_BUFFER *filter; NWDS_TYPE InfoType; NWDS_BUFFER AllAttrs; NWDS_BUFFER *iterationHandle; NWDS_NUM_OBJ CountObjectsToSearch; NWDS_NUM_OBJ *countObjectsSearched; NWDSBuffer *objectInfo;	

The Future of NDS

Issues To Be Resolved

NDS still has a few issues to be worked out. We need to iron out issues with character sets, attribute syntax conflicts, and OID compatibilities with Class/Attribute/Syntax names. NDS also does not use ASN1 for its attribute definitions. However, NDS will be ASN1 compliant in the future.

Statement of Direction

NDS is also currently moving beyond the desktop and Windows worlds. NDS needs a full set of APIs that will access information and allow it to

virtually run itself. NDS will soon be using distributed objects.

Conclusion

NDS is compliant with X.500 directory services. However, the X.500 standards merely provided a sound starting point. NDS has already moved beyond the X.500 standards in terms of supplying services and capabilities. The next releases of NDS will run with more operating systems and provide even more sophisticated access to network services. r

Appendix A

X.500 and NDS: How Close Are They?

Overview

This appendix discusses the compatibilities between X.500 and NDS. It addresses an audience of network administrators and developers that is acquainted with directory services concepts.

Introduction to X.500 Directory Services and NDS

Network administrators have always had to invest considerable time in configuring and managing network resources and services. Traditionally, these services have been tightly integrated into the operating system or into applications such as email. However with networks achieving global scale, administrators have had to become even more concerned with managing network resources. So, replicated and distributed directories have become important because they simplify and streamline network administration. Also, as global networking has become more common, companies have seen the advantage to compatible networks on this scale. This led to the definition of a directory services standard. The X.500 directory services specification has become one of the most accepted standards for duplicated, replicated databases and directory services. *Novell Directory Services* (NDS) is functionally equivalent to, and can coexist with X.500, compliant directory services. NDS's main purpose is to manage network resource configuration information.

NDS is a full-fledged directory service modeled after X.500. But NDS has different goals. Currently, X.500 directory services are primarily viewed as a way to store email addresses. NDS, on the other hand, is currently used to administer users, servers, services, and network resources. NDS allows administrators to effectively manage global networks on a host by host or domain by domain basis. Because of NDS's broader scale, it must be flexible enough to allow the network administrator to enhance the network environment, without bringing the network down. In other words, while X.500 is currently being used as a global postmaster, NDS is designed to operate, and is currently operating as, a global services registry. A directory service implementation requires the following elements:

- The Directory Information Base (DIB)
- Authentication and authorization methods
- APIs and protocols

This paper discusses the functional compatibilities between the NDS and X.500 directory services in these areas and concludes with a statement of NDS's future direction.

Functional Comparisons

Table A.1 briefly compares NDS with X.500 directory services.

Table A.1
Comparison of X.500 Directory Services and NDS

Directory Services Features and Functions	X.500 1988	X.500 1993	NDS
Directory Information Base (DIB)	Distributed and replicated. Changes propagated from one server.	Distributed and replicated.	Distributed and replicated. Changes propagated from any server.
Information Storage Model	Identical	Identical	Identical
Directory Information Tree (DIT) or Namespace	Hierarchal and identical.	Hierarchal and identical.	Hierarchal and identical.
Aliases	Allowed	Allowed	Allowed
Metaschema	Compatible	Compatible	Compatible
Schema	Compatible	Compatible	Compatible
Extensible schema	Yes, but typically must edit file and down the server to reload schema. Includes compound attributes.	Includes dynamic compound attributes.	Yes. Can extend schema without downing network. Defines hard-coded attribute syntaxes in the place of compound attributes.
Object Classes	Defined in X.521 standards.	Defined in ITU recommended X.521 standard (ISO 9594-7).	Objects identical to those defined by X.521. Also defines additional objects. String representations use a 2-byte character representation for unicode.
Attributes	Defined in X.520 standards. Uses OID.	Defined in ITU recommended X.520 standard (ISO 9594-6).	X.520 compliant. Also defines additional attributes. String representations use a 2-byte character representation for unicode. Does not support OID.
Attribute Syntaxes			String representations use a 2-byte character representation for unicode.
Security	Uses X.509 standards.		Uses RSA GQ algorithm.
APIs	Uses XDS APIs.		Compatible with XDS APIs.

Table A.1
Comparison of X.500 Directory Services and NDS (Continued)

Directory Services Features and Functions	X.500 1988	X.500 1993	NDS
Protocols	Large and slow.		Defined smaller, faster protocols.
Protocol Verbs	Supports 8 verbs.		Supports 75 verbs.
Logical Operations	Identical		

The Directory Information Base (DIB)

The X.500 specification describes a Directory Information Base (DIB) that contains information about the objects stored in the directory. The DIB is composed of directory entries, each of which is made up of attributes that contain information about that entry. The attributes each entry contains depend upon the object class of the entry.

The DIB entries are arranged in the form of a logical tree structure called the *Directory Information Tree* (DIT). Entries in the containers and branches of the tree usually represent countries and organizations. Entries in the tree leaves usually represent people, services, applications and physical objects.

NDS uses an identical DIT/DIB structure.

Namespaces and Tree Structures

The X.500 specification mandates that each entry have a distinguished name that uniquely and unambiguously identifies the entry. The DIB tree structure lends the entry its distinguished name: the object's name is composed of each container's name in the object's path back to the root.

X.500 also allows *alias entries.* An alias entry points to an object entry. That is, the alias name maps to a distinguished name, allowing users and administrators to more easily access objects.

The NDS and X.500 namespaces are the same. Both use identical naming schemes and both allow alias entries.

Metaschema

X.500 and NDS use identical metaschemas. A metaschema is briefly defined as the way you describe how you describe objects. For example, an NDS object class is defined with the following kinds of information:

- Structure Rules
- Object Class Flags
- ACL Templates

- Super Classes
- Object Class Attributes

That definition is the metaschema of an NDS object class. A base schema object class is created by defining values and attributes for each of the information types. Attributes and attribute syntaxes are also governed by metaschemas.

NDS does not support the X.500 compound attributes and sequence syntaxes. NDS does define a string syntax which X.500 does not. This syntax allows NDS to represent strings in a 2-byte Binary Multiple Plain (BMP) character representation which supports unicode. The 1993 X.500 implementation supports a universal string format. As a result, NDS can represent any X.500 or T61 string value, but neither the 1988 or 1993 X.500 implementations can represent all the unicode values. (Unicode is equivalent to the ISO 1046 universal character set.)

Schema

The X.500 and NDS schemas are also compatible.

Both X.500 and NDS use extensible schemas. That is, both specifications allow network administrators and developers to define new objects and expand the base schema.

The X.520 Standards. X.500 directory services uses a schema described by the X.520 specification. The X.520 specification describes attributes and attribute definitions. The NDS schema is also X.520 compliant. In addition, the NDS schema has added attributes. As mentioned under the metaschema, the NDS schema uses a 2-byte character representation for unicode.

The X.521 Standards. The X.521 specification defines object classes. Some of the NDS objects are identical to those defined by X.521. In addition, the NDS schema has added object classes required by the common NDS implementations. For example, the X.520 specification describes a residential person. NDS does not have much need for a residential person, so the NDS schema defines a user that is more business oriented. However, developers can easily add a residential person object class.

NDS also requires object classes that X.521 did not explicitly define. NDS handles this with macros that represent information required by NDS, but not defined by X.521. NDS uses a very similar class hierarchy as that described by X.521. X.500 directory services use object identifiers (OIDs). An OID allows you to assign numbers to unique object classes, syntaxes and attributes with the absolute assurance that the number will not be assigned to any other object class, syntax, or attribute. For example, Novell is registered as an organization and has been assigned the OID XXX. Whenever that OID is used to access an object class, that object

class will always be Novell, Inc. NDS does not currently use OIDs. However, it will use them in future implementations.

X.500 Schema Compatibilities. NDS uses X.500 attributes with X.500 attribute syntaxes. In addition, the NDS schema is flexible enough that you can add to the schema without taking the system down. NDS allows the administrator to add to the schema on the fly and while the network is running.

As discussed above, the NDS schema is very compatible with X.500 directory services. NDS has also augmented the X.500 schema by adding additional object classes and attributes that allow a network operating system to transparently handle objects and that also allow NDS to address the business needs of Novell's customers.

Partitioning and Replication

NDS differs from X.500 in its synchronization scheme. The X.500 standards specify that updated directory information be propagated from one server. In reality, this model is too unwieldy to synchronize a globally distributed and replicated network. Therefore, NDS allows updated information to be propagated from any server. However, this difference does not affect NDS compatibility with X.500.

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 10 seconds to 5 minutes depending upon the type of information updated. s

Authentication and Authorization

X.509 Authentication

X.500 directory services uses an authentication algorithm defined by the X.509 standards. This algorithm is the RSA authentication algorithm

which is based on public and private keys and provides a great deal of security. The X.509 authentication standards have two major disadvantages for NDS:

- Services that use the X.509 standards must allow users constant access to their private key
- Products using X.509 are not always exportable

Private Key Access

Because NDS currently addresses mainly DOS and Microsoft Windows workstations, and because both DOS and Windows have memory restrictions and limited memory security, our customers requested that NDS not leave the private key in memory. In response to our needs, RSA supplied the Guillou-Quisquater (GQ) zero-knowledge proof of identity scheme. GQ is a dual signature algorithm. In essence, this algorithm uses a secret key derived from another secret key. This enables NDS to use the private key to sign the authentication credentials, then discard it. NDS then uses a key derived from the private key to authenticate. NDS does not use this key to encrypt or decrypt information because the key is temporary. GQ also allows NDS the convenience of multiple keys and proxy authentication.

Issues with Exporting and Importing

Ironically, NDS cannot use X.509 authentication methods because of their excellent information encryption and decryption capabilities. Because of potential national security risks, the National Security Agency (NSA) does not always allow companies to export or import products that use these algorithms. So, even though the X.509 security algorithms are excellent, NDS will continue to use the GQ security algorithms and will use the X.509 capabilities primarily as add-on services and applications.

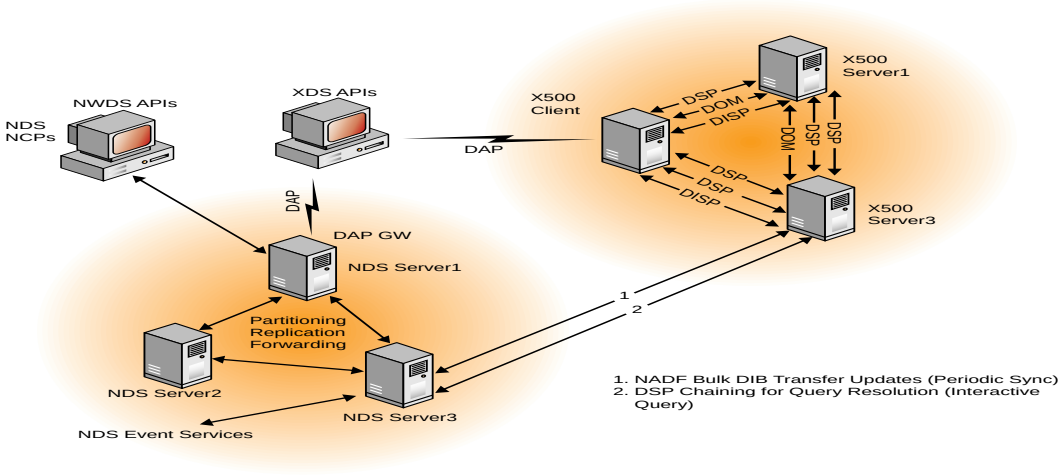
X.509 Access Control List

The X.509 standards define a very powerful and flexible access control list. NDS's access control list is just as powerful and flexible.

APIs and Protocols

Both X.500 directory services and NDS use multiple protocols to access the same information. In fact, a network using NDS and a network using an X.500 directory service can coexist, as shown in Figure10.1.

Figure10.1
Coexisting X.500 and NDS Networks



NWDS APIs and the X.500 Protocols

Although the X.500 specification defines extremely powerful and adaptable protocols, NDS defines its own protocols that are compatible with the XDS protocols. The X.500 protocols are very big and very slow. Because NDS currently addresses mostly DOS and Windows machines, it has to fit in 640 Kilobytes of memory. We cannot load the OSI stack and have room in memory for anything else.

So, we developed a lighter weight version of the protocol. We developed our protocol at about the same time that LDAP (Lightweight Directory Access Protocol) was being developed. Our protocol is very LDAP-like. However, LDAP uses *printf* to place all the attributes onto the wire and then retrieves them from the wire with *scanf*. Our protocol uses binary representation, not string representation which allows it to be smaller and faster.

NWDS and XDS APIs

The NWDS APIs are really a superset of the XDS APIs, and the NWDS API arguments map directly to the XDS protocol arguments with one exception: the XDS protocols include a *session* argument that the NDS protocols do not. This *session* parameter is basically a file pointer that holds an open connection. XDS uses this parameter to deal with some general overhead. We thought that our bindery users would be inconvenienced with this overhead, so we decided to not trouble our users with it and put the overhead in our libraries so that NDS could be more self-reliant. Table A.2 illustrates the relationship between the XDS and NWDS APIs.

Protocol Verbs

NDS supports 75 verbs. X.500 allows 8 verbs.

Logical Operations

NDS's logical operations come straight from X.500 specification. NDS uses search and list filters exactly the same way as X.500.

Table A.2
Relationship Between the XDS and NWDS APIs

XDS Function	XDS Parameters	NWDS Function	NWDS Parameters	Comments
ds_abandon	OM_private_object session; OM_sint invoke_id;	none		NDS supports only synchronous calls
ds_add_entry	OM_private_object session; OM_private_object context; OM_object name; OM_object; OM_sint *invoke_id_return;	NWDSAddObject	NWDSContextHandle context; char *objectName; NWDS_ITERATION NWFAR *objectInfo; NWFLAGS more; NWDS_BUFFER NWFAR *ObjectInfo;	context ↔ context name ↔ objectName entry ↔ objectInfo
ds_bind	OM_object session; OM_private_object *bound_session_return;	NWDSAuthenticate	NWDSCONN_HANDLE Conn; NWDS_FLAGS OptionsFlag; NWDS_SESSION_KEY *sessionKey;	uses strong authentication to bind a connection
ds_compare	OM_private_object session; OM_private_object context; OM_object name; OM_object; OM_private_object *restructuring; OM_sint *invoke_id_return;	NWDSCompare	NWDSContextHandle context; char NWFAR *objectName; NWDS_BUFFER NWFAR *buf; NWFLAGS NWFAR *matched;	context ↔ context name ↔ objectName ava ↔ buf result_return ↔ matched
ds_initialize	void - returns OM_workspace	NWDSCreateContext	void - returns NWDSContextHandle	Not an exact mapping. Both must be called before any other functions.
ds_list	OM_private_object session; OM_private_object context; OM_object name; OM_private_object *result_return; OM_sint *invoke_id_return;	NWDSList	NWDSContextHandle context; char NWFAR *objectName; NWDS_ITERATION NWFAR *iterationHandle NWDS_BUFFER NWFAR *subordinates;	context ↔ context name ↔ objectName result_return ↔ subordinates
ds_modify_entry	OM_private_object session; OM_private_object context; OM_object name; OM_object changes; OM_sint *invoke_id_return;	NWDSModifyObject	NWDSContextHandle context; char NWFAR *objectName; NWDS_ITERATION NWFAR *iterationHandle NWFLAGS more; NWDS_BUFFER NWFAR *changes;	context ↔ context name ↔ objectName changes ↔ changes

Table A.2
Relationship Between the XDS and NWDS APIs (Continued)

XDS Function	XDS Parameters	NWDS Function	NWDS Parameters	Comments
ds_modify_rdn	OM_private_object session; OM_private_object context; OM_object name; OM_object new_RDN; OM_object delete_old_RDN; OM_sint *invoke_id_return;	NWDSModifyRDN	NWDSContextHandle context; char NWFAR *object; char NWFAR *newDN/ *newRDN; NWFLAGS deleteOldRDN;	context ↔ context name ↔ objectName newRDN ↔ newDN/ newRDN delete_old_RDN ↔ deleteOldRDN
ds_read	OM_private_object session; OM_private_object context; OM_object name; OM_object selection; OM_private_object *result_return; OM_sint *invoke_id_return;	NWDSRead	NWDSContextHandle context; char NWFAR *objectName; NWDS_TYPE info-type; NWFLAGS allAttrs; NWDS_BUFFER NWFAR *attrNames; NWDS_ITERATION NWFAR *iterationHandle NWDS_BUFFER NWFAR *objectInfo;	context ↔ context name ↔ objectName selection ↔info_type, allattrs, attrNames result_return ↔ objectInfo
ds_search		NWDSSearch	NWDSContextHandle context; char NWFAR *baseobjectName; NWDS_SEARCH_SCOPE scope; NWFLAGS SearchAliases; NWDS_BUFFER *filter; NWDS_TYPE InfoType; NWDS_BUFFER AllAttrs; NWDS_BUFFER *iterationHandle; NWDS_NUM_OBJ CountObjectsToSearch; NWDS_NUM_OBJ *countObjectsSearched; NWDSBuffer *objectInfo;	

The Future of NDS

Issues To Be Resolved

NDS still has a few issues to be worked out. We need to iron out issues with character sets, attribute syntax conflicts, and OID compatibilities with Class/Attribute/Syntax names. NDS also does not use ASN1 for its attribute definitions. However, NDS will be ASN1 compliant in the future.

Statement of Direction

NDS is also currently moving beyond the desktop and Windows worlds. NDS needs a full set of APIs that will access information and allow it to

virtually run itself. NDS will soon be using distributed objects.

Conclusion

NDS is compliant with X.500 directory services. However, the X.500 standards merely provided a sound starting point. NDS has already moved beyond the X.500 standards in terms of supplying services and capabilities. The next releases of NDS will run with more operating systems and provide even more sophisticated access to network services. r