INTERNET DIRECTORY SERVICES USING THE LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL

Greg Lavender and Mark Wahl
Sun Microsystems, Inc.

ABSTRACT

We survey the history, development and usage of directory services based on the Lightweight Directory Access Protocol (LDAP). We present a summary of the naming model, the schema model, the principal service models, and the main protocol interactions in terms of a C language application programming interface.

1 INTRODUCTION

The landscape of network-based directory technology is fascinating because of the evolution of distributed systems ideas and Internet protocol technologies that have contributed to the success of the Internet as a collection of loosely coordinated, interoperable network-based systems. The success of open systems directory technology based on the Lightweight Directory Access Protocol (LDAP) is attributed to the persistence of many people in academia, industry and within international standards organizations. Today, LDAP-based technology is widely used within national and multi-national intranets, wired and wireless service provider value-added networks, and the public Internet. This success is due to an Internet community process that worked to define and evolve practical X.500 and LDAP directory specifications and technologies towards a more ubiquitous Internet directory service.

There are many different types of directory services, each providing useful capabilities for users and applications in different network-based settings. We are primarily concerned with directories that have a structured data model upon which well-defined operations are performed, most notably search and update. Directories are used to service a much higher percentage of authentication operations (technically called "bind" operations) and search operations, rather than update operations, which requires them to optimize for reading rather than writing information. However, as directories become increasingly authoritative for much of the information that is used to enable a wider range of web services as part of private intranet and public internet infrastructures, they are increasingly required to provide the kind of update rates one expects of a transactional database management system.

Directories in general can be characterized by a hierarchical tree-based naming system that offers numerous distributed system advantages:

- names are uniquely determined by concatenating hierarchical naming components starting at a distinguished root node (e.g., "com")
- object-oriented schema and data model supporting very fast search operations
- direct navigation or key-based querying using fully qualified or partially specified names
- distribution and replication based on named sub-tree partitions
- delegated or autonomous administration based on named sub-tree partitions

In addition, Internet directories add various authentication services and fine-grained access control mechanisms to ensure that access to information contained in the directory is only granted to authorized users. The type of information contained in a directory can be either general purpose and extensible, or specialized for a particular optimized directory service. In the following section, we briefly distinguish a general purpose directory service based on LDAP from more specialized directory services, all of which typically coexist as part of an organizational information service, but each providing an important component of a multi-tiered networked information service.

The most widely known directory service on the Internet today is the Domain Name System (DNS). Others that have appeared and are either lightly used or no longer used include: *whois*, *whois++,* and *WAIS*. The primary purpose of DNS is to provide name to address lookups where a name is a uniquely determined hierarchical Internet domain names and an address is an IP or other network layer protocol addresses, resulting in a very specialized name-to-address mapping service. DNS has a hierarchical name structure, it is widely replicated, and it is autonomously administered. However, we distinguish DNS as a specialized directory service because its information model and query mechanism is specialized to the purpose of providing very specific host addressing information in response to direct or reverse lookup queries. There have been various extensions to DNS to extend its rather limited data model, namely in the area of service (SRV) records, but such records do not significantly enhance the main DNS hostname-to-address lookup service.

Another class of directories provides various specialized naming services for network operating systems, called NOS directories. Popular NOS directories include NIS for UNIX® systems, NDS for Novell Netware™, and Active Directory for Microsoft Windows™. NOS directories are typically based on proprietary protocols and services that are tightly integrated with the operating system, but may include some features and services adopted from X.500 and LDAP directory specifications in order to permit a basic level of interoperability with client applications that are written to such open systems specifications. NOS directories are very well suited to local area network environments in support of workgroup applications (e.g., file, print and LAN email address book services), but have historically failed to satisfy other business-critical applications requiring Internet scale levels of service and reliability.

Network-based file systems, such as NFS or AFS , may also be considered as specialized directory services in that they support an RPC-based query language that uses hierarchical file names as keys to access files for the purposes of reading and writing those files across a network. There have been attempts to create Internet-scale distributed file systems, but most file systems are highly specialized for efficient reading and writing of large amounts of file-oriented data on high-bandwidth local area networks or as part of a storage area network. Network file systems are not typically intended for use as general-purpose directory services distributed or replicated across wide-area networks, although there are various attempts underway to define Internet-scale file systems.

Various ad hoc directory services have also been constructed using custom database systems (e.g., using sequentially accessed flat files or keyed record access) or relational database management systems. For example, a common application of a directory service is a simple “white pages” address book, which allows various lookup queries based on search criteria ranging from people’s names, email addresses or other searchable identity attributes. Given that RDBMS products are widely deployed and provide robust information storage, retrieval and transactional update services, it is straightforward to implement a basic white pages directory service on top of an RDBMS. However, there can be complications due to limitations imposed by the fact that most directory services are defined in terms of a hierarchical data model and the mapping of this hierarchical data model into a relational data model is often less than satisfactory for many network based directory-enabled applications that require ultra fast search access and cannot tolerate the inherent overhead of mapping between the two data models. In addition, network based authentication and access control models may require an extra abstraction layer on top of the RDBMS system. This mapping may result in less functionality in these areas as well as potential inefficiencies.

Finally, numerous “yellow pages” directories have arisen on the Internet that provide a way to find information. While called “directories” because of the way information is organized hierarchically and presented via a web browser, such directories are more properly text-based information search and retrieval systems that utilize web spiders and some form of semantic qualification to group textual information into categories that meet various search criteria. These web directories are highly useful for searching and accessing very large information spaces that are otherwise difficult to navigate and search without the assistance of some form of browsable navigation directory (e.g., Yahoo and Google). While powerful, we distinguish this type of directory from LDAP-based directories that rely on an extensible, but consistent data model that facilitates a more highly structured search mechanism. Some yellow pages directories actually utilize LDAP directories to augment their generalized text search mechanisms, by creating structured repositories of meta information, which is used to guide future searches based on stored

topological information and/or historical search results.

2 THE EVOLUTION OF LDAP

The LDAPv3 protocol and modern directory servers are the result of many years of research, international standardization activities, commercial product developments, Internet pilot projects, and an international community of software engineers and network administrators operating a globally interconnected directory infrastructure. LDAP directories originated from projects and products that originally implemented directories based on the 1988 and 1993 International Telecommunications Union (ITU) X.500 series of international standards [ITU X.500, 1993], [ITU X.501, 1993], [ITU X.511, 1993], [ITU X.518-X.521, 1993], [ITU X.525, 1993]under the assumption that electronic messaging based on the ITU X.400 series of standards would eventually become the dominant e-mail system in the international business community. For several technical, economic, and sociopolitical reasons, X.400 has not become the dominant email transport system that was envisioned, but X.500 has survived and many LDAP protocol features and the schema and data model are derived from the X.500 specifications, and the early work on deploying practical Internet directories by the IETF OSI-DS working group [RFC1274] .

Before the emergence of the Internet as a ubiquitous world-wide public network, there was an assumption that the world's data networks, and the messaging and directory infrastructures deployed on them, would be operated much like the world's voice communications networks were managed in the 1980s. These networks were managed by large telecommunications companies that had international bilateral agreements for handling such international communication services. The emergence of the worldwide Internet as a viable commercial network undercut many of the underlying service model assumptions that had gone into designing these technologies by participants in the International Telecommunications Union standards organizations. As universities, industrial R&D organizations, and technology companies increasingly embraced the Internet as a model for doing business they created technologies that effectively created a value-added network on top of the bandwidth leased from the Telcos. This new style of networking created new demand for innovations that would better fit the Internet style of networked computing, where there is a very high degree of local autonomy with regard to deploying and managing network services as part of an enterprise's core IT function, rather than leasing expensive application services from a telecommunications company, as opposed to simply leasing bandwidth.

2.1 The Past, Present and Future Generations of LDAP Directories

Since the mid-1990's, directory servers based on LDAP have become a significant part of the network infrastructure of corporate intranets, business extranets, and service providers in support of many different kinds of mission-critical networked applications. The success of LDAP within the infrastructure is due to the gradual adoption of directory servers based on the LDAPv3 protocol. The use of LDAP as a client access protocol to X.500 servers via an LDAP-to-X.500 gateway, has been replaced by pure LDAPv3 directory servers. In this section, we briefly review the technological evolution that led to the current adoption of LDAPv3 as part of the core network infrastructure.

During the 1980s, both X.400 and X.500 technologies were under active development by computer technology vendors, as well as by researchers working in networking and distributed systems. Experimental X.400/X.500 pilots were being run on the public research Internet based on open source technology called the ISO Development Environment (ISODE) , which utilized a *convergence protocol* allowing OSI layer 7 applications to operate over TCP/IP networks through a clever mapping of the OSI class 0 transport protocol (TP0) onto TCP/IP. TP0 was originally designed for use with a reliable, connection-oriented *network* service, such as that provided by the X.25 protocol. However, the principal inventor of ISODE, Marshall Rose (who was to go on to develop SNMP), recognized that the reliable, connection-oriented *transport* service provided by TCP/IP could effectively (and surprisingly efficiently) masquerade as a reliable connection-oriented network service underneath TP0. By defining the simple (5 byte header) convergence protocol specified in [RFC 1006], and implementing an embedding of IP addresses into OSI presentation addresses, the OSI session and presentation layers could be directly mapped onto TCP. Hence any OSI layer 7 application requiring the upper-layer OSI protocols could instantly be made available on the public Internet. Given the lack of public OSI networks at the time, this pragmatic innovation enabled the rapid evolution of X.500-based directory technology through real-world deployments across the

Internet at universities, industrial and government R&D labs, and forward looking companies working to commercially exploit emerging Internet technologies. The lack of any real competitive technology led to the rapid adoption of network-based directory services on the Internet (unlike X.400, which failed to displace the already well-established SMTP as the primary Internet protocol for email).

LDAP arose primarily in response to the need to enable a fast and simple way to write directory client applications for use on desktop computers with limited memory capacity (<16MB) and processing power (<100 MHz). The emergence of desktop workstations and the PC were driving demand for increasingly sophisticated client applications. In order for X.500 to succeed independently of X.400, client applications were needed that could run on these rather limited desktop machines by today's standards. A server is only as good as the service offered to client applications, and one of the major inhibitors to the adoption of X.500 as a server technology outside of its use as an address routing service for X.400, was the lack of sophisticated client applications. The X.500 client protocol, DAP , was used principally by X.400 to access information required to route X.400 messages. Since X.400 was another server application based on the full OSI stack, the complexity of using the DAP protocol for access to X.500 was "in the noise" given the overall complexity and computing resources required by a commercial grade X.400 system. Many early X.400/X.500 vendors failed to grasp that simply specified protocols and the "good enough" services they enable were a key driver in the growth of the Internet and the resulting market demand for server software. As a result, X.400 and X.500 infrastructures were not as widely deployed commercially as anticipated, except in selected markets (e.g., some military and governmental organizations) where a high degree of complexity is not necessarily an inherent disadvantage.

The simplest application for a directory is as a network-based white pages service, which requires a rather simple client to issue a search using a string corresponding to a person's name or a substring expression that will match a set of names, and return the list of entries in the directory that match the name(s). This type of simple white pages client application was the original motivation for defining LDAP as a "lightweight" version of DAP. Some people like to claim that that the "L" in LDAP no longer stands for "lightweight" because LDAP is now used in servers to implement a full-blown distributed directory service, not just a simple client access protocol. However, the original motivation for making LDAP a lightweight version of DAP was to eliminate the requirement for the OSI association control service element (ACSE) , the remote operations service element (ROSE) , the presentation service, and the rather complicated session protocol over an OSI transport service (e.g., TP4). Even with the convergence protocol defined in [RFC 1006], the upper layer OSI protocols required a rather large in-memory code and data footprint to execute efficiently. LDAP was originally considered "lightweight" precisely because it operated directly over TCP (eliminating all of the OSI layers except for use of ASN.1 and BER [ITU X.681, 1993] [ITU X.690, 1993]), had a much smaller binary footprint (which was critical for clients on small memory desktop PCs of the time), and had a much simpler API than DAP. In this context, the use of the term lightweight meant a small memory footprint for client application, fewer bits on the wire, and a simpler programming model, not lightweight functionality.

The LDAPv2 specification ([RFC 1487] and [RFC 1777]) was the first published version of the lightweight client directory access protocol. While supporting DAP-like search and update operations, the interface to LDAPv2 was greatly simplified in terms of the information required to establish an association with an X.500 server, via an LDAP-to-DAP gateway. The introduction of this application level protocol gateway mapped the client operations to the full DAP and OSI protocol stack. So in this sense, LDAPv2 was a proper subset of the services offered by DAP, and no changes were required to an X.500 server to support these lightweight client applications, such as address book client services as part of a desktop email application. LDAPv2 enabled rapid development of client applications that could then take advantage of what was expected to be a global X.500-based directory system. As client applications began to be developed with LDAPv2, some operational shortcomings manifested themselves. The most notable, was the lack of a strong authentication mechanism. LDAPv2 only supports anonymous and simple password based authentication (note: RFC 1777 predated the emergence of SSL/TLS). Such security concerns, the mapping of the X.500 geopolitical naming model to the Internet domain names, the need for referrals to support distributed directory data, the need for an Internet standard schema (e.g., `inetOrgPerson`) and the desire for a mechanism for defining extensions, led to the formation of IETF LDAPEXT working group that began defining a richer set of services based on LDAPv2, which became a series of specification for

LDAPv3 (RFCs 2251-2256).

This process of gradually realizing that a simpler solution will likely satisfy the majority of user and application requirements is a recurring theme in today's technology markets. In addition, the emergence of business critical web-based network services and the wide spread adoption of LDAPv3 based technologies as part of the infrastructure enabling web services has enabled a sustainable market for LDAP directories and ensured that someday in the future, LDAP directories will be considered entrenched legacy systems to be coped with by some future infrastructure initiatives.

2.2 First and Second Generation Directory Services

As just discussed, LDAP originated as a simplification of the X.500 DAP protocol to facilitate client development on small machines. The first LDAPv2 client applications were used with an X.500 server called "Quipu," which was developed as a collaboration among various European and American universities, research organizations, and academic network providers and administrators.[1] Quipu was based on the 1988 X.500 standards specifications and as such, implemented DAP as its client access protocol, requiring either a full OSI protocol stack or, as discussed previously, the OSI upper layers over [RFC 1006]. Quipu was deployed on the research Internet in the late 1980's and gained substantial exposure as an early directory service on the Internet at relatively small scale (e.g., 100k directory entries was considered a large directory at the time). Quipu was deployed at a number of universities as part of the Paradise directory project, which was administered from University College London where Quipu was primarily developed. In cooperation with researchers at the University of Michigan (i.e., Tim Howes), individuals at the Internet service provider PSI (i.e., Marshall Rose and Wengyik Yeong), researchers at University College London (e.g., Steve Kille), and other individuals, LDAPv2 emerged and an application layer gateway called the LDAP daemon (ldapd) was developed at the University of Michigan that mapped LDAPv2 operations to DAP operations that were then forwarded onto an X.500 server, such as Quipu. As a result of LDAPv2 and this LDAP-to-DAP gateway, lightweight client applications were rapidly developed that could run on Windows and Macintosh PCs.

The success of Quipu as an early prototype X.500 directory on the Internet and LDAPv2 as a client led to further innovation. One of the main advantages of Quipu was that it was extremely fast in responding to search operations. This was due to its internal task switching (co-routine) architecture, which predated POSIX threads on most UNIX® systems, and the fact that on startup, it cached all of the directory entries into memory. This feature also severely limited the scalability of Quipu because of the expense and limitations of physical memory on 32-bit server machines and the potentially long startup time required to build the in-memory cache. Work was begun in 1992, both at the University of Michigan and at the ISODE Consortium, to produce a more scalable and robust directory server. The ISODE Consortium was an early open source organization that was a spinout of University College London and the Microelectronics and Computer Technology Corporation (MCC) in Austin, Texas.

The University of Michigan team first developed a server that exploited X.500 chaining to create an alternative back-end server process for Quipu that utilized a disk-based database built from the UNIX® dbm package as the underlying data store. Client requests were first sent to the main Quipu server instance that maintained topology information in it cache that allowed it to chain the requests to the back-end server. Effectively, Quipu was turned into a "routing proxy" and scalability was achieved with one or more back-end servers hosting the data in it disk-based database, using caching for fast access. This approach proved the viability of a disk-based database for Quipu, but without integrating the disk-based database into the core of the Quipu server. This approach was taken for simplicity, and also because POSIX threads were finally viable in UNIX® and the new back-end server was based on a POSIX threading model instead of a task-based co-routine model. However it suffered from the drawback of now having two separate server processes between the LDAP client and the actual data. An LDAP client request had to first go through the LDAP-to-DAP gateway, then through the Quipu server, then to the back-end server over the X.500 DSP protocol [ITU X.525, 1993], then back to the client.

[1] A Quipu (pronounced *key-poo)* is a series of colored strings attached to a base rope and knotted so as to encode information. This device was used by peoples of the ancient Inca empire to store encoded information.

During this same period, the ISODE Consortium began work on a new X.500 server that was essentially a rewrite of much of Quipu. Based on the promising performance and scalability results from the University of Michigan's back-end server implementation, the goal was to integrate the disk-based backend and POSIX threading model into a new, single process directory server that could scale well and deliver search performance comparable to that of Quipu. This result was achieved with a new directory server from ISODE in 1995, based on the 1993 X.500 ITU standards. The protocol front-end was also re-designed so that additional protocol planes could be added as needed to accommodate additional server protocols. At the time, there was serious consideration given to implementing a DNS protocol plane, so that the directory server could be used to provide a DNS service in additional to both the LDAP and X.500 directory services. However, this work was never done. Instead, work was done to provide an integrated LDAPv3 protocol plane alongside of the X.500 DAP and DSP protocols, resulting in the first dual protocol X.500+LDAP directory server.

The work at the University of Michigan continued in parallel and it became obvious that one could do away with the LDAP-to-DAP protocol gateway and the routing Quipu server, and simply map LDAPv2 directly to the new disk-based back-end server. All that was required was to implement the LDAPv2 protocol as a part of the back-end server and eliminate the X.500 DSP protocol. This was the key observation that led to the first pure directory server based on LDAPv2 with some extensions, called the standalone LDAP daemon (slapd) .

These two separate architectural efforts led to the definition of the LDAPv3 protocol within the IETF, which was jointly defined and authored by individuals at the University of Michigan (Tim Howes and Mark Smith) and at the ISODE Consortium (Steve Kille and Mark Wahl). Both slapd and the ISODE server were the first directory servers to implement LDAPv3 as a native protocol directly, and validated its utility for implementing a directory service, not just as a client access protocol to an X.500 directory. In addition, both servers adopted the Berkeley DB b-tree library package as the basis for the disk-based backend, which added to the scalability, performance, and eventual robustness of both servers. In 1996, Netscape hired the principal inventors of the University of Michigan slapd server, which became the Netscape Directory Server that was widely adopted as an enterprise-scale LDAPv3 directory server. In 1995, the ISODE Consortium converted from a not-for-profit open-source organization to a for-profit OEM technology licensing company and shipped an integrated X.500 and LDAPv3 server. Also in 1996, Critical Angle was formed by former ISODE Consortium engineers and they developed a carrier-grade LDAPv3 directory server for telecommunications providers and ISPs. This was the first pure LDAPv3 server to implement chaining via LDAPv3, and also the first server to have multi-master replication using LDAPv3. In addition, Critical Angle developed the first LDAP Proxy Server, which provided automatic referral following for LDAPv2 and LDAPv3 clients, as well as LDAP firewall, load balancing, and failover capability, which are critical features for large-scale directory service deployments.

With the emerging market success of LDAPv3 as both a client and a server technology on the Internet and corporate intranets, vendors of previously proprietary LAN-based directory servers launched LDAPv3 based products, most notably Microsoft Active Directory and Novell eDirectory. Microsoft had originally committed to X.500 as the basis for its Windows™ directory server, but adopted LDAP as part of its Internet strategy.

IBM built its SecureWay LDAP directory product using the University of Michigan slapd open source code and designed a mapping onto DB2 as the database backend. Like IBM, Oracle implemented an LDAP gateway onto its relational database product. However, it is generally the case that mapping of the hierarchical data model of LDAP and X.500 onto a relational data model has inherent limitations. For some directory service deployments, the overhead inherent in the mapping is a hindrance in terms of performance.

Most X.500 vendors continue to provide an LDAP-to-DAP gateway as part of their product offerings, but their marketing does not usually mention either the gateway or X.500, and instead calls the X.500 server an LDAP server.

The Critical Angle LDAP products were acquired in 1998 by Innosoft International . In 1999, AOL

acquired Netscape. Sun Microsystems and AOL/Netscape entered into a joint marketing and technology alliance called iPlanet shortly thereafter. In March 2000, Sun acquired Innosoft, and consolidated its directory server expertise by combining the Netscape technology, the Innosoft/Critical Angle technology, and its own Solaris LDAP-based directory technology initiatives into a single directory product line under the Sun ONE™ brand.

2.3 Next-Generation Directory Services

Now that LDAPv3 directory servers are widely deployed, whether native LDAP implementations or using LDAP gateways mapping operations onto X.500 servers or relational database systems, new types of directory-based services are being deployed. The most recent of these are identity management systems, which provide authentication, authorization, and policy based services using information stored in the directory. The most common type of identity management service is web single sign-on. With the proliferation of network based systems, the need for a common authentication mechanism has dictated that a higher-level service be deployed that abstracts from the various login and authentication mechanisms of different web-based services. Identity servers built on top of directory services are providing this functionality. At present, such services are primarily being deployed within an enterprise, but there are efforts underway to define standards for federating identity information across the Internet. It will take some time before these standards activities and the technologies that implement them are deployed, but the foundation on which most of them are being built is a directory service based on LDAPv3.

Another area where LDAP directories are gaining wide spread usage is among wireless carriers and service providers. Next generation wireless services are providing more sophisticated hand-held devices with the opportunity to interact with a more functional server infrastructure. Directories are being deployed to provide a network-based personal address book and calendars for mobile phones and PDAs that can be synchronized with the handheld devices, a laptop computer and a desktop computer. Directory services are being deployed as part of the ubiquitous network infrastructure that is supporting the management of personal contact and scheduling information for hundreds of millions of subscribers. Fortunately, LDAP directory technology has matured to the point at which it is capable of providing the performance, scalability and reliability required to support this “always on” service in a globally connected world.

In order to simplify the integration of directory data access into web service development environments, new directory access protocols building on LDAP are being defined. Instead of ASN.1 and TCP, these protocols use XML [Bray et. al., 2000] as the encoding syntax and Simple Object Access Protocol (SOAP) [Gudgin, 2003] as a session protocol overlying HTTP or persistent message bus protocols based on Java Messaging Service (JMS) APIs. The standards body where these protocols are being developed is OASIS, the Organization for the Advancement of Structured Information Standards.

One group within OASIS in particular, the Directory Services working group, has published version 2 of the Directory Services Markup Language (DSMLv2), which leverages the directory semantics from LDAP: a hierarchical arrangement of entries consisting of attributes, but expresses the LDAP operations in SOAP. Just as X.500 servers added support for LDAP either natively or through an LDAP to X.500 gateway, there are implementations of DSMLv2 both as native protocol responders within an LDAP server, or a DSMLv2 to LDAP gateway.

Other working groups have already or are in the process of defining protocols for more specialized directory access, such as Universal Description, Discovery and Integration of Web Services (UDDI) [Bellwood, 2003], ebXML Registry Services [OASIS, 2002], and Service Provisioning Markup Language (SPML) [Rolls, 2003]. In the future, market dynamics may favor the adoption of one or more of these XML-based protocols to augment and eventually supplant LDAP as the primary client access protocol for directory repositories in the web services environment.

3 THE LDAP NAMING MODEL

The primary contents of most directory services are entries that represent people, but entries may also represent organizations, groups, facilities, devices, applications, access control rules, and any other information object. The directory service requires every entry have a unique name be assigned when the

entry is created, and most services have names for users which are based on attributes which do not change frequently, and are human readable.

Entries in the directory are arranged in a single-rooted hierarchy. The Distinguished Name (DN) of an entry consists of the list of one or more distinguished attribute values chosen from the entry itself, followed by attributes from that entry's parent, and so on up the tree to the root. In most deployments, only a single attribute value is chosen from each entry.

The most widely used naming attributes are defined in the following table.

dc	domainComponent: one element of a DNS domain name, e.g. dc=sun, dc=com
uid	userid: a person's account name, e.g. uid=jbloggs
cn	commonName: the full name of a person, group, device, etc. e.g. cn=Joe Bloggs
l	localityName: the name of a geographic region, e.g. l=Europe
st	stateOrProvinceName: used in the United States and Canada
o	organizationName: the name of an organization, e.g. o=Sun Microsystems
ou	organizationalUnitName: the name of a part of an organization: ou=Engineering
c	countryName: the two letter ISO 3166 code for a country, e.g. c=US

The hierarchy of entries allows for delegated naming models, in which the organization managing the name space near the root of the tree agrees on the name for a particular entry with an organization to manage that entry, and delegates to that organization the ability to construct additional entries below that one. Several naming models have been proposed for LDAP.

3.1 The X.500 Naming Model

The original X.500 specifications assumed a single, global directory service, operating based on interconnections between national service providers. In the X.500 naming model, the top levels of the hierarchy were to have been structured along political and geographic boundaries. Immediately below the root would have been one entry for each country, and the entries below each country entry would have been managed by the telecommunications operator for that country. (In countries where there were multiple operators, the operators would have been required to coordinate the management of this level of the tree.)

As there is no one international telecommunications operator, an additional set of protocol definitions was necessary to define how the country entries at the very top of the tree were to be managed. The telecommunications operator for each country would be able to define the structure of entries below the country entry. The X.500 documents suggested that organizations that had a national registration could be represented by organization entries immediately below the country entry. All other organizations would be located below entries that represented that country's internal administrative regions, based on where that organization had been chartered or registered. In the United States and Canada, for example, there would have been intermediate entries for each state and province, as states and provinces operate as registrars for corporations. For example, a corporation that had been chartered as "A Inc." in the state of California in the United States, might have been represented in the X.500 naming model as an entry with the name o=A Inc.,st=California,c=US, where o is the attribute for the organization name that was registered within the state, st for state or province name within the country, and c the attribute for the country code. It should be noted that some certificate authorities which register organizations in order to issue them X.509 public key certificates, e.g. for use with SSL or secure email, assume this model for naming organizations.

The entries for people who were categorized as part of an organization (e.g. that organization's employees) would be represented as entries below the organization's entry. However X.500 did not suggest a naming

model for where entries representing residential subscribers would be represented.

3.2 Limitations of the X.500 Naming Model

The first limitation is that there is no well-defined place in the X.500 model to represent international and multi-national organizations. Organizations such as NATO and agencies of the United Nations and the European Community, as well as multi-national corporations, were some of the first to attempt to pilot standards-based directory services, yet ran into difficulties as there was no obvious place for the organization's entry to be located in a name space that requires a root based on a national geopolitical naming structure.

A related problem is that a corporation typically operates in additional locales beyond the one in where they are legally incorporated or registered. In the United States for example, many corporations are registered in Delaware for legal and tax reasons, but may have no operating business presence in that state beyond a proxy address. A naming structure that has the organization based in Delaware may hinder users searching the directory, who might not anticipate this location as the most likely place to find the directory information for the corporation.

In some cases, organizations preferred to have an entry created for them in a logically appropriate place in the X.500 hierarchy, yet the telecommunications operator implied by the naming model as being authoritative for that region of the directory tree may have had no plans to operate X.500. Conversely, some parts of the directory tree had conflicting registration authorities as a result of political turf wars and legal disputes, not unlike those that plagued the Internet Assigned Numbers Authority (IANA) and the administrators of the root DNS servers. For use within the United States and Canada, the North American Directory Forum (NADF) proposed a set of attribute and server extensions to address the problems of overlapping registration authorities creating entries for individual and business subscribers in entries, however these extensions were not implemented, and no X.500 service saw significant deployment in these countries.

3.3 Early Alternatives to the X.500 Naming Model

Many LAN-based directory services predating LDAP suggested a simpler naming model. Unlike the complete interconnection in a single, global directory service, these models assumed that interconnection only occurred between pairs or small groups of cooperating organizations, and that relatively few organizations worldwide would interconnect. Instead of the geographic divisions of X.500, this naming model based on a single registration authority that would assign names to organizations immediately below the root of the tree, e.g. `o=Example`, resulting in a flattened namespace. This model did not readily accommodate conflicts in names between organizations, and relied on one registration authority to ensure uniqueness.

As the Internet became more widely used by organizations for electronic mail, a variant of the flat namespace model was to register the organization's Internet domain name as the value for the `organizationName` attribute, e.g. `o=example.com`. By relying on an external naming authority for managing the actual assignment of names to organizations, potential conflicts would be resolved before they reached the directory name registration authority, and the use of the hierarchical domain name space would allow for multiple organizations with the same name that were registered in different countries or regions, e.g. `o=example.austin.tx.us` and `o=example.ca`.

3.4 Internet Domain-based Naming

The single-component organization naming model described above addresses the difficulty that organizations have when faced with getting started using the X.500 model, but this approach suffers from a serious limitation. While domain names themselves are hierarchical, placing the full domain name as a string into the `organizationName` attribute prevented the hierarchical structure of the directory from being used. In particular, it was not defined in that approach how an organization that was itself structured and represented internally with multiple domain names, e.g. `east.example.com` and `west.example.com`, would be able to manage these as part of a hierarchy below the organization entry.

These limitations were removed in the mapping defined in [RFC 2247] between Internet domain names and LDAP distinguished names. In this approach, each domain name component is mapped into its own value of the `dc` attribute to form the distinguished name of the entry (typically an organization) to which the domain corresponds. For example, the domain name `cs.ucl.ac.uk` would be transformed into `dc=cs,dc=ucl,dc=ac,dc=uk`, and the domain name `example.com` into `dc=example,dc=com`.

Follow-on documents have proposed how DNS SRV records can be used to specify the public directory servers that an organization provides, in a similar manner to the MX records for specifying the mail servers for that organization. In combination with RFC 2247 naming, a directory client that knows only a domain name can use these techniques to locale the LDAP server to contact, and construct the search base to use for the entry which corresponds to that domain.

RFC 2247 assumes that when performing operations on the directory entries for an organization the organization's domain name is already known to the client so that it can be automatically translated into a distinguished name to be used as an LDAP search base. RFC 2247 does not address how to programmatically locate an organization when the organization's domain name is not known; this is currently an unsolved problem in the Internet.

3.5 Naming Entries Within an Organization

There are currently no Internet standards that are widely adopted for naming entries representing people within an organization. Initial deployments of LDAP made extensive use of the `organizationalUnit` entries to construct a directory tree which mirrored the internal divisions of the organization, and use the cn attribute as the distinguished attribute for the person's entry, as in the following:

```
cn=Joe Bloggs, ou=Western, ou=Sales, ou=People, dc=example, dc=com
```

That approach however resulted in organizations needing to frequently restructure their directory tree as the organization's internal structure changed, and even placing users within organizational units did not eliminate the potential for name conflicts between entries representing people with the same full name. Currently the most common approach for directory deployments, in particular those used to enable authentication services, is to minimize the use of `organizationalUnit` entries, and to name users by their login name in the `uid` attribute.

Many deployments now have only a single `organizationalUnit` entry: `ou=People`. Some multi-national organizations use an `organizationalUnit` for each internal geographic or operating division, in particular when there are different provisioning systems in use for each division, or it is necessary to partition the directory along geographic lines in order to comply with privacy regulations. For example, an organization that has two operating subsidiaries X and Y might have entries in their directory named as follows:

```
uid=jbloggs, ou=X, ou=People,dc=example, dc=com
uid=jsmith, ou=France, ou=Y, ou=People, dc=example, dc=com
```

For service provider directories or directories that offer a hosted directory service for different organization entities, the DNS domain name component naming is most often used to organize information in the directory naming tree as follows:

```
uid=jbloggs, ou=X, dc=companyA, dc=com
uid=jsmith, ou=Y, dc=companyA, dc=com
uid=jwilliams, ou=A, dc=companyB, dc=com
uid=mjones, ou=B, dc=companyB, dc=com
```

In a typical service provider or hosted directory environment, the directory data for different organizations

is physically partitioned and an LDAP proxy server is used to direct queries to the appropriate directory server that holds the data for the appropriate naming context.

4 THE LDAP SCHEMA MODEL

Directory servers based on LDAP implement an extensible object-oriented schema model that is derived from the X.500 schema model, but with a number of simplifications as specified in [RFC 2256]. Two additional schema definition documents are [RFC 2798], which defines the `inetOrgPerson` object class, and [RFC 2247], which defines the `dcObject` and domain object classes. The schema model as implemented in most LDAP servers consists of two types of schema elements: attribute types and object classes. Object classes govern the number and type of attributes that an entry stored in the directory may contain, and the attribute types govern the type of values that an attribute may have. Unlike many database schema models, LDAP schema has the notion of multi-valued attributes that allows a given attribute to have multiple values. The types of the values that may be associated with a given attribute are defined by the attribute type definition. The most common attribute type is a UTF-8 string, but many other types occur, such as integer, international telephone number, email address, URL, and a reference type that contains one or more distinguished names, representing a pointer to another entry in the directory.

A directory entry may have multiple object classes that define the attributes that are required to be present, or may optionally be present. Directory servers publish their internal schema as an entry in the directory. It can be retrieved by LDAP clients performing a `baseObject` search on the a special entry that is defined by the directory server to publish schema information (e.g., `cn=schema`), with the attributes `attributeTypes` and `objectClasses` specified as part of the search criteria. This schema entry maintains the schema definitions that are active for a given directory server instance. The format of these two attributes is defined in [RFC 2252].

4.1 Attribute Type Definitions

An attribute type definition specifies the syntax of values of the attribute, and whether the attribute is restricted to having at most one value, and the rules that the server will use for comparing values. Most directory attributes have the Directory String syntax, allowing any UTF-8 encoded Unicode character, and use matching rules that ignore letter case and duplicate white space characters.

User attributes can have any legal value which the client provides in the add or modify request, but a few attributes are defined as operational, in which the attributes are managed or used by the directory server itself, and may not be added or changed by most LDAP clients directly. Examples of operational attributes include `createTimestamp` and `modifyTimestamp`.

4.2 Object Class Definitions

Each entry has one or more object classes, which specifies the real-world or information object that the entry represents, as well as the mandatory and permitted attributes defined in the entry. Object classes come in one of three kinds: *abstract*, *structural* or *auxiliary*.

There are only two abstract classes. The `top` class is present in every entry, and it requires the `objectClass` attribute be present. The other abstract class is `alias`, and it requires the `aliasedObjectName` attribute be present.

Structural object classes define what the entry represents, and every entry must contain at least one structural object class; for example: `organization`, `device`, or `person`. A structural object class inherits either from top or from another structural object class, and all the structural object classes in an entry must form a single 'chain' leading back to `top`. For example, the object class `organizationalPerson` inherits from `person`, and permits additional attributes to be present in the user's entry that describe the person within an organization, such as `title`. It is permitted for an entry to be of object classes `top`, `person` and `organizationalPerson`, but an entry cannot be of object classes `top`, `person` and `device`, since `device` does not inherit from `person`, nor `person` from `device`.

Auxiliary classes allow additional attributes to be present in a user's entry, but do not imply a change in what the entry represents. For example, the object class `strongAuthenticationUser` allows the attribute `userCertificate;binary` be present, but this class could be used in an entry with object class `person`, `device` or some other structural class.

4.3 Object Classes for Entries Representing People

The person structural class requires the attributes `cn` (short for `commonName`) and `sn` (short for `surname`). This class is sub-classed by the `organizationalPerson` class, and the `organizationalPerson` class is sub-classed by the `inetOrgPerson` class. Most directory servers for enterprise and service provider applications use `inetOrgPerson`, or a private sub-class of this class, as the structural class for representing users. In addition to the mandatory attributes `cn`, `sn` and `objectClass`, the following attributes are typically used in entries of the `inetOrgPerson` object class:

`departmentNumber`	a numeric or alphanumeric code
`description`	a single line description of the person within the organization
`displayName`	name of the user as it should be displayed by applications
`employeeNumber`	unique employee number
`employeeType`	a descriptive text string, such as "Employee" or "Contractor"
`facsimileTelephoneNumber`	fax number, in international dialing format (e.g., +1 999 222 5555)
`givenName`	first or given name
`homePhone`	home phone number, in international dialing format
`homePostalAddress`	home mailing address, with '$' inserted between lines
`jpegPhoto`	photograph in JPEG format
`labeledURI`	the URI for a web home page
`mail`	Internet email address
`manager`	distinguished name of the entry for a manager
`mobile`	mobile phone number in international dialing format (e.g., +1 999 222 4444)
`ou`	organizational Unit, if different from department
`pager`	pager phone number and codes, if any
`postalAddress`	mailing address, with '$' inserted between lines
`roomNumber`	office room number
`secretary`	distinguished name of the entry for a secretary
`surname (sn)`	Last or surname
`telephoneNumber`	telephone number, in international dialing format
`title`	Job title
`uid`	user id, typically part of the person's distinguished name
`userPassword`	user password compared against during LDAP authentication

For example, the following definition is an LDAP Interchange Format (LDIF) text representation of a typical user entry in an LDAP directory, as specified in [RFC 2849]. The order of appearance of a most attributes and value pairs does not imply any specific storage requirements, but it is convention to present the `objectClass` attribute first, after the distinguished name. Several additional attributes are permitted in entries of this object class, but are no longer widely used. For further details consult [RFC 2256] and [RFC 2798].

```
dn: uid=jbloggs,ou=people,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
cn: Joe Bloggs
sn: Bloggs
```

```
departmentNumber: 15
description: an employee
displayName: Joe Bloggs
employeeNumber: 655321
employeeType: EMPLOYEE
facsimileTelephoneNumber: +1 408 555 1212
givenName: Joe
homePhone: +1 408 555 1212
homePostalAddress: Joe Bloggs $ 1 Mulberry Street $ Anytown AN 12345
labeledURI: http://eng.example.com/~jbloggs
mail: jbloggs@example.com
manager: uid=jsmith,ou=people,dc=example,dc=com
mobile: +1 408 555 1212
ou: Internet Server Engineering
pager: +1 408 555 1212
postalAddress: Joe Bloggs $ 1 Main Street $ Anytown AN 12345
roomNumber: 2114
telephoneNumber +1 408 555 1212 x12
title: Engineering Manager
uid: jbloggs
userPassword: secret
```

4.4 Other Typical Object Classes

The `organization` structural class requires the `o` attribute (short for `organizationName`) to be present in the entry, and permits many attributes to optionally also be present, such as `telephoneNumber`, `facsimileTelephoneNumber`, `postalAddress` and `description`. This class is normally used to representcorporations but could also represent other organizations that have employees, participants or members and a registered name.

The `organizationalUnit` structural class requires the `ou` (short for `organizationalUnitName`) be present, and permits the same list of optional attributes as the `organization` class. This class is normally used to represent internal structures of an organization, such as departments, divisions, or major groupings of entries (e.g., `ou=People`).

The `domain` structural class requires the `dc` (`domainComponent`) attribute be present. This class is used to represent objects that have been created with domain component naming, but no other information about them is known. The `dcObject` auxiliary class is used to permit the `dc` attribute to be present in entries of the `organization` or `organizationalUnit` classes, typically so that the `dc` attribute can be used for naming the entry, although the `o` or `ou` attribute is still required to be present.

The `groupOfNames` structural class uses `cn` for naming the group, which is represented by the attribute `member`. This requires one or more attribute values each containing the distinguished name of another entry that is a member of the group. A similar and more widely used class, `groupOfUniqueNames`, uses the `uniqueMember` attribute.

5 LDAP DIRECTORY SERVICES

There are several architectural applications of LDAP in today's Internet: email address book services, web-based white pages lookup services, web authentication/authorization services, email server routing and address list expansion services, and literally hundreds of uses that are generally categorized as a network-based repository for application-specific information (e.g., application configuration information, directory-enabled networking such as router tables, and policy-based user authentication and access authorization rules). In other words, basic LDAP directory services have become a critical part of the network infrastructure for many applications, just as DNS, FTP, SMTP and HTTP are core infrastructure services. LDAP is very often there behind the scenes of many end-user applications and embedded in a number of

other services that are not end-user visible.

One of the most common questions that arises in corporate directory services deployments is the following: "why not just use a relational database system rather than a new kind of database?" The answer to this question is often as varied as the application services in support of which an LDAP directory is being considered. In some cases, such as a simple web-based white pages service, there is no real compelling advantage over using a RDBMS that may already contain all the information about people and their contact details. Directories are a distinct type of distributed database and are best suited to a new generation of network-based applications whose data access and service availability requirements do not require a relational data model or a SQL-based transactional database.

An important distinction that networking system novices make is to distinguish a protocol and the service implemented in terms of the protocol. A protocol defines the communication pattern and the data exchanges between two end points in a distributed system. Typically, one end point is a client and the other a server, but both end points could be peers. The semantics offered by a server often extends beyond the information exchange rules that are specified by the protocol. In other words, the server may require additional features to implement a reliable, maintainable, highly available service that transcend the basic information exchange implied by a protocol. For example, LDAPv3 has the concept of extended operations and special controls, some of which are standardized, and some of which are not. The result is that vendors have created extensions to the core protocol specifications to enable additional services, such as configuration and management of the server over LDAP without ever having to shutdown the server to ensure high availability. This is not necessarily a bad thing, since extended operations and controls are useful from an administrative perspective, enabling network-based management of an LDAP service using the LDAP protocol or special server-to-server communication enabling a distributed directory service. In a competitive market, where technology vendors compete with one another by enabling proprietary client visible features, complete interoperability between clients and servers may be broken. This situation is typically avoided by having periodic interoperability testing forums where competing vendors demonstrate interoperability. As long as the core protocol and basic service model is not violated then client interoperability is maintained.

Another confusing aspect of standards based network services is the difference between standards conformance versus demonstrated interoperability. Within some standards organizations, the emphasis has been on demonstration of static conformance to a written specification. Static conformance is often achieved through the successful demonstration that a server passes some set of conformance tests. Conformance testing is useful to the vendor, but what is most useful to a user is interoperability testing, i.e., does the server from vendor A work with clients from vendors B, C, and D? Conformance is easier to achieve than interoperability. LDAPv3 has been shown to be a highly interoperable protocol and most clients work without complication with most servers. Vendors of LDAPv3 client and server products regularly meet to perform interoperability testing forums, sponsored by The Open Group, to ensure that products work together. In the remainder of this section we discuss the basic and advanced modes of operations for common LDAP-based directory services.

5.1 Basic Directory Services

The LDAP service model is a basic request/response client-server model. A client issues either a bind, search, update, unbind, or abort request to the server. All protocol operations may either be initiated directly over a TCP connection or encrypted via an SSL/TLS session.

The bind operation is used to pass authentication credentials to the server, if required. LDAP also supports anonymous search, subject to access control restrictions enforced by the server. Bind credentials consist of the user's distinguished name along with authentication credentials. A distinguished name typically identifies a user corresponding to a logical node in the directory information tree (DIT). For example: `uid=jbloggs, ou=People, dc=sun, dc=com`. Authentication credentials may be a simple clear text password (optionally over a SSL or TLS session), information obtained from a digital certificate required to strongly authenticate, or other encrypted or hashed password authentication mechanisms enabled by a particular server.

There are three principal models of distributed operations: a simple client-server model, a referral model, and a chaining model. The simple client server interaction is not depicted, but operates as one would expect: a client performs a bind operation against the server (either anonymously or with bind credentials) issues an LDAP operation (e.g., search), obtains a result (possibly empty or an error), and then either issues other operations, or unbinds from the server. In this mode of operation, an LDAP server will typically handle hundreds to thousands of operations per second on behalf of various types of LDAP-enabled clients and applications.

In some deployments, most notably those on the public Internet or in government, university and enterprise directory service environments where anonymous clients may connect and search a forest of directories, a referral model may be appropriate. The referral model assumes that either a) all clients will bind anonymously, or b) authentication information is replicated among the set of directory servers, or c) there is some mechanism for proxy authentication at one server on behalf of a network of trusted directory servers that will accept authentication credentials via a proxy [Weltman, 2003].

Figure 1 depicts the most common referral model situation, and assumes that a client is anonymous, such that it can bind and search any one of the two directory servers depicted. In the referral model, if a client request an operation against one directory server (e.g., a search operation) and that directory server does not hold the entry or entries that satisfy that operation, then a referral may be returned to the client. A referral is a list of LDAP URLs that point to other directory servers that the original server is configured to refer queries too. A referring server may have out of date information and the referral may not succeed. Referral processing is the responsibility of the client application and is most often handled transparently as part of the LDAP API from which the client application is built. The referral model is most often appropriate in directory service deployments where there are no stringent requirements on authentication since servers may be configured to accept unauthenticated anonymous operations, such as searches. In fact, one of the major disadvantages of the referral model is that it facilitates trawling of a large distributed directory service and allows a snooping client application to probe and discover a directory service's topology. This may be undesirable even in a public Internet environment, such as a university network. For this reason, LDAP proxy servers were invented to provide an additional level of control in these more open network environments.

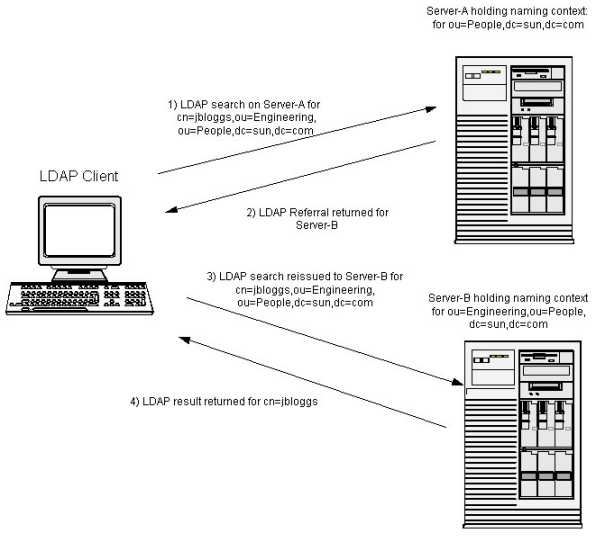

Figure 1. LDAP Referral Model

The chaining model is similar to the referral model, but provides a higher degree of security and administrative control. LDAP chaining is similar to chaining in the X.500 directory service model, but it is done without requiring an additional sever-to-server protocol as in the case of X.500. Figure 2 illustrates the chaining model. In this case, a client issues a search request to Server-A, which uses its knowledge of which server holds the subordinate naming context, and it chains the request to Server-B.

Chaining assumes an implied trust model between Server-A and Server-B, since typically Server-A will authenticate to Server-B as itself, not as the client. For efficiency in the chaining model, it is typical for Server-A to maintain a persistent open network connection to Server-B to eliminate the overhead of binding for each chained operation. In some cases, such as anonymous clients, there is no authentication information to proxy, so Server-A will often maintain a separate, non-authentication connection to Server-B for such requests. A proxied authentication model could also be used in which case the client credentials are passed along as part of the chained operation, requiring both Server-A and Server-B to authenticate, or proxy the authentication of the client.

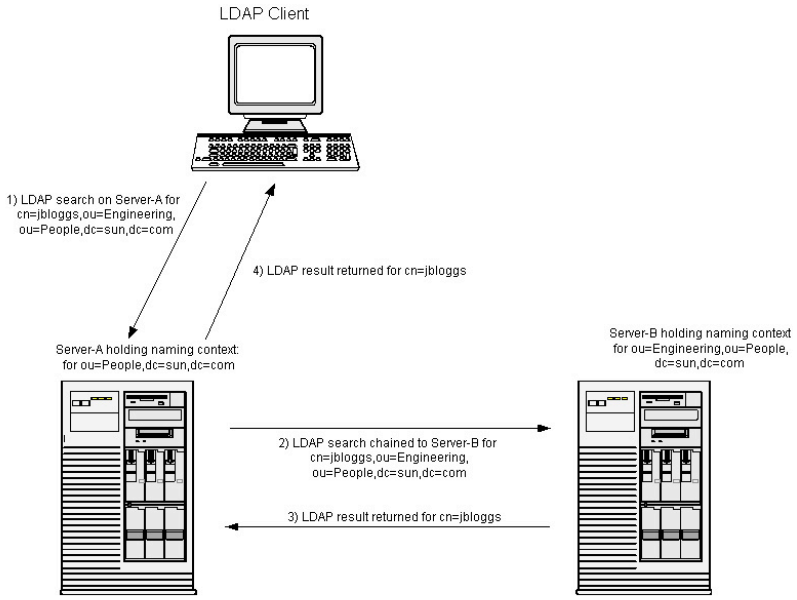

Figure 2. LDAP Chaining Model

5.2 High Availability Directory Services

First generation X.500 and LDAP directory servers focused primarily on implementing as much of the protocol specifications as possible, providing flexible and extensible schema mechanisms, and ensuring very fast search performance. During the early adoption phase of a new technology, these are the critical elements to get right and it is necessary for rapid feedback and technology evolution as the result of real world deployments. However, as LDAP server technology has become more central to the network infrastructure backing up mission-critical business operations (e.g., as a network user authentication service), security, reliability, scalability, performance, and high availability of the service are the dominant operational requirements. In remainder of this section, we briefly discuss high availability features and issues related to deployment of LDAP directory services in support of business critical applications.

5.2.1 Master-Slave Replication

The X.500 specifications define a replication protocol called DISP (Directory Information Shadowing Protocol) that provides a simple master-slave replication service. Technically, directory replication is based on a *weakly consistent* supplier-consumer model, but the master-slave terminology has become dominant. In this weakly consistent replication model, one directory server acts a supplier (master) of data to another directory server that is the consumer (slave). At any given time a replica may be inconsistent with the authoritative master, and so a client accessing data at the replica might not see the latest modifications to the data. If this situation is likely to cause problems for the client applications, there are deployment techniques (e.g., using a proxy server) that ensure that a client application will be connected with an authoritative master server so that it may obtain up-to-date information.

Within the X.500 model it is possible for either the supplier to initiate a replication data flow, or for the

consumer to request a replication data flow from the supplier. Replication can occur as soon as a change is detected in a supplier, called on-change, or according to some periodic schedule. Supplier initiated replication works best when network links between the supplier and the consumer are reliable, with low latency and reasonably high bandwidth. Consumer initiated replication is most often used in situations where the consumer might frequently become disconnected from the network, either due to the unscheduled network outages or high latency, low bandwidth networks that require potentially large replication data exchanges to be done during off-peak times as determined by the consumer (e.g., a consumer on a computer in a submarine).

Replication typically requires that not only directory entries be replicated from a supplier to a consumer, but also schema and access control information. If not done automatically via a protocol, manual configuration is required for each consumer, and there could be thousands of consumers in large distributed directory system (e.g., a directory consumer in every airport in the world holding flight schedule information). Some directory servers do not implement schema and access control such that it can be replicated, so manual configuration, or some other out-of-band technique is used to replicate this type of operational information.

While master-slave replication provides high-availability via geographic distribution of information, facilitating scalability for search operations, it does not provide high-availability for modify operations. In order to achieve write-failover, it is necessary to employ systems engineering techniques to enable either a cluster system running the single master, or enable a hot standby mode so that in the event of the failure of the single master, the hot standby server can be brought on line without delay. Another technique is to allow a slave server to become a master server through a process of *promotion*, which involves special control logic to allow a slave to begin receiving updates and to notify other slaves that it is now the authoritative master. Many directory deployments only allow write operations to the master server and route all search operations, using DNS or an LDAP Proxy server, to one of the slaves so that the load on the master server is restricted to only modify operations.

A common scenario that is employed in single master, multiple slave, directory deployments is to deploy a small set of replica hubs, each being a read-only replica from which other slaves may obtain updates. In the event a master fails, a hub is easily promoted. In this model, depicted in Figure 3, the replica hubs are both consumers and suppliers, since they consume their updates from an authoritative master server, but also supply updates to other slaves. This scenario is most useful when a large number of slave servers would put unnecessary load on a single master, and so a hierarchy of servers is established to distribute the replication update load from the master to the hubs; otherwise, the master might spend all of its time updating consumers.

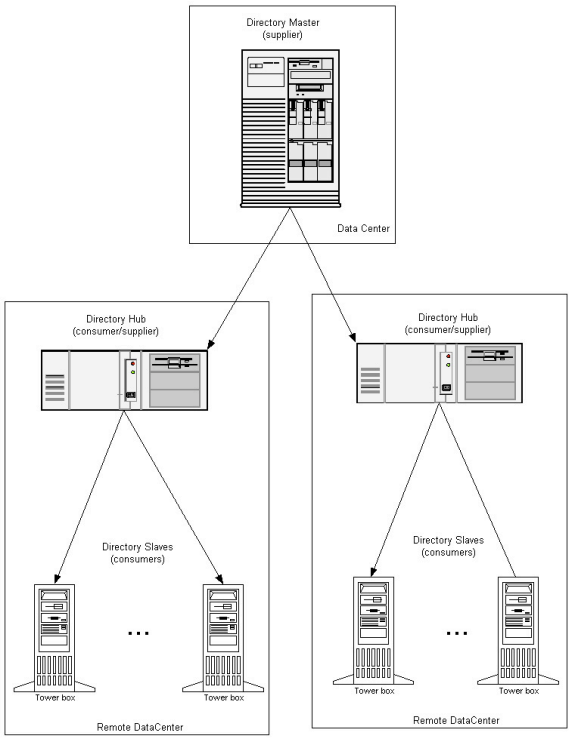

Figure 3. Master-Slave Replication

5.2.2 LDAP Proxy Server

An LDAP proxy server is an OSI layer 7 application gateway that does LDAP protocol data unit forwarding, possibly requiring examination, and possibly on-the-fly modification, of the LDAP protocol message to determine application layer semantic actions. The objective of the LDAP proxy server is to provide an administrative point of control in front of a set of deployed directories, but does not answer LDAP queries itself, instead chaining the queries when appropriate to the directory servers. The proxy allows a degree of transparency to the directory services, in conjunction with DNS maps pointing LDAP clients to one or more proxy services instead of at the actual directory servers. The LDAP proxy provides a number of useful functions that are best done outside of the core directory server. These functions include:

1. LDAP schema rewriting to map server schema to client schema in the cases where client schema is either hard-coded or for historical reasons does not match the extensible schema of the directory server. Once thousands of clients are deployed it is difficult to correct the problem, and so server applications must often adapt for the sake of seamless interoperability.

2. Automatic LDAP referral following on behalf of both LDAPv2 and LDAPv3 clients. The LDAPv2 protocol did not define a referral mechanism, but a proxy server can map an LDAPv2 client request into an LDAPv3 client request, so that referrals can be used with a mixed set of LDAPv2 and LDAPv3 clients.

3. An LDAP firewall that provides numerous control functions to detect malicious behavior on the part of clients, such as probing, trawling, and denial-of-service attacks. The firewall functions include rate limiting, host and TCP/IP-based filters similar to the TCP wrappers package, domain access control rules, and a number of LDAP-specific features, including operations blocking, size limits, time limits and attribute filters. The rate limiting feature allows a statistical back-off capability using TCP flow control so that clients attempting to overload the directory are quenched.

4. The proxy provides a control point for automatic load balancing and failover/failback capability. It may also be able to maintain state information about load on a set of directory servers and redirect

LDAP traffic based on load criteria or other semantic criteria at the LDAP protocol level. In addition, it can detect the failure of a directory server and rebalance the incoming load to other directory servers, and detect when a failed directory server rejoins the group.

5. The proxy also provides the point-of-entry advertised to clients in the DNS to the directory service thereby providing a level of indirection to client applications that facilitates maintenance, upgrades, migrations, and other server administrative tasks done on the backend directory servers, in a manner that is transparent to clients so that a highly available directory service is delivered.

An LDAP proxy is unlike an IP firewall in that it does application layer protocol security analysis. The proxy is also unlike an HTTP proxy in that it does not do caching of directory data since it is unable to apply directory access rules to clients. The LDAP proxy is also unlike an IP load balancer in that it is able to make application level load balancing decisions based on knowledge of directory server topology, query rates, query types, and load and availability metrics published by the directory servers.

Figure 4 is an illustration of a typical deployment of a pair of LDAP proxy servers that sit behind an IP firewall and accept LDAP connection requests and operations on port 389.

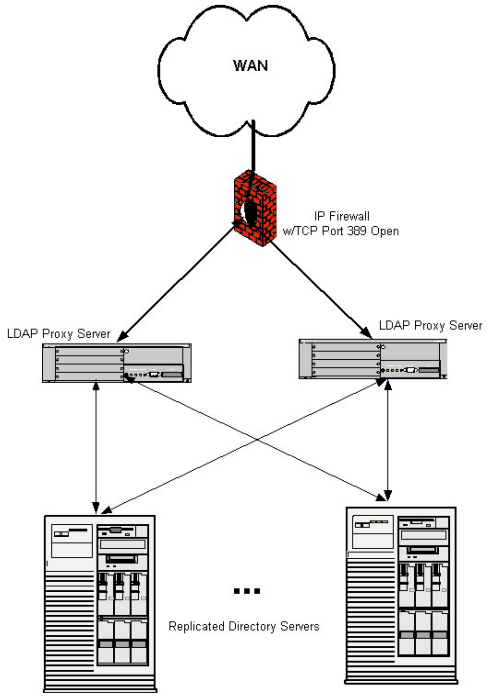

Figure 4. LDAP Proxy Servers

5.2.3 Multi-Master Replication

Multi-master replication with high-speed RAID storage, combined with multiple LDAP proxy servers and dynamic DNS provides a very high availability directory service. In some cases, a clustered operating system platform may provide additional availability guarantees. There are different techniques for implementing multi-master replication but they all share in common the goal of maintaining write availability of a distributed directory service by ensuring that more than one master server is available and reachable for modifications, that will *eventually* be synchronized with all of the other masters. As discussed previously, LDAP replication is based on a weakly-consistent distributed data model, and so any given master may be in a state of not having processed all updates seen at other master servers. This feature of an LDAP directory service is sometimes criticized as a weakness of the LDAP service model, but in well-designed directory service deployments with high bandwidth, low latency LANs and WANs, it is possible to have weak consistency and still provide a very high service level for most application environments. In

loosely coupled distributed systems over WANS, globally consistency is very difficult to achieve economically. For those environments that require total consistency at any point in time, another network based distributed data service is probably more appropriate.

Ideally, any directory server in a deployment of a directory service could be a master for some or all of the data that it holds, thereby providing n-way multi-master replication. In practice however, the masters are typically placed in a controlled environment like a data center, where the server's authoritative database can be properly administered, regularly backed up, etc. In the case of geographically dispersed data centers, then each data center may contain one or more master servers interconnected by a high-speed LAN that stay in close synchronization, while masters in another data center are connected by a slower WAN and might often be out of synchronization. Different high availability goals will dictate how the masters are deployed, how the data is partitioned, and how proxy servers are deployed to help enable client applications to get at replicas or the masters themselves.

Whatever the choice of topology, multi-master replication combined with hub and slave replicas, and proxy servers, offers a highly available directory service. In this scenario, proxy servers provide a critical piece of functionality since replicas and hubs will often be configured to return referrals to one or more master servers for a client that requests to do an update on a replica. Alternatively, if the directory server used as a replica offers chaining, then it may be able to chain the operation to the master. For modifications, it is often desirable to have the client application authenticate to the master with its own credentials for auditing purposes, rather than have a replica proxy the modification on behalf of the client applications.

5.2.4 Replication Standardization

There is no official standard for how replication is to be done with LDAP. Each LDAP directory server vendor has implemented a specialized mechanism for replication. There are many reasons why no LDAP replication standard was reached within the IETF, but the principal reason was that no consensus could be reached on standard access control mechanism. A common access control mechanism, or consistent mapping, is required before LDAP replication interoperability can be achieved between servers from different vendors. In addition, market politics inhibited the successful definition, as the access control model and other semantic features implemented as part of a directory service, independent of the LDAP directory protocol, were viewed as competitive elements of various vendor products.

The question is often asked by the LDAP standards working group did not simply adopt the X.500 access control model and replication protocol, since they were already standardized. The reasons are complicated but there is an important fact about X.500 replication that is often not often well understood. The X.500 replication protocol suffers from numerous flaws itself, and most X.500 products implement proprietary workarounds in their products to enable replication to work at scale beyond a couple of hundred thousand directory entries. Briefly, the DISP requires a replication data unit to be sent as a single protocol data unit for both the total update during initialization, as well as during incremental updates for changes. Using the OSI upper layer protocols, the DISP protocol is defined in terms of a remote procedure call service and a session layer service that was not well designed to take procedure arguments that could be on the order of several megabytes to several gigabytes. Hence, implementations of DISP that use the remote operation service element (ROSE) of the OSI stack are most often severely limited in their ability to perform replication updates of any appreciable size. The alternative is to implement DISP in terms of the reliable transfer service element (RTSE), which is used by X.400, but no two X.500 vendors convincingly demonstrated interoperability at scale. As a result, X.500 total update replication in practice is neither scalable nor interoperable as a practical matter between any two X.500 serves from different vendors. Each vendor had no choice but to make changes to the protocol as implemented in their product to achieve practical replication between their own servers, and those changes usually make all but the most basic level of interoperability unachievable in practice. Fully interoperable and scalable replication between disparate directory servers, whether X.500 or LDAP, has not yet been achieved. The replication problem remains an active area of research, especially with respect to performance and scalability over WANs, and topology and replication agreement manageability of potentially hundreds-to-thousands of master-master and master-slave replication agreements.

6 LDAP PROTOCOL AND C LANGUAGE CLIENT API

The LDAPv3 protocol is defined in [RFC 2251]. Closely related specifications are [RFC 2222] and [RFCs 2252-2256]. Many Internet application protocols, such as SMTP and HTTP, are defined as text-based request-response interactions; however, LDAPv3 (like SNMP) is defined using Abstract Syntax Notation One (ASN.1) so that application protocol data units (PDUs) are strongly typed in that structural type information is sent along with user data in the form of encoded type tags. Clients and servers implementing LDAPv3 use the Basic Encoding Rules (BER) to encode protocol data units as compact network byte order binary strings before transmission via TCP. This encoding/decoding process introduces slight computational overhead to protocol processing, but processing of LDAP operations is less computational intensive than other ASN.1 represented protocols such as DAP or XML encoded protocols, and even some text based protocols that require a lot of string parsing and end-of-message handling. This efficiency is due to the restricted use of only the basic and efficiently encoded ASN.1 data types in defining the LDAP protocol structure. Most data elements are represented as strings that easily encoded using BER. This optimization allows very compact and efficient encoders/decoders to be written.

6.1 LDAPv3 Protocol Exchange

As with typical client-server protocols, the LDAPv3 protocol exchange is initiated by a client application requesting a TCP connection to a server, typically on reserved TCP port 389. Either endpoint of the connection can send an `LDAPMessage` PDU to the other endpoint at any time, although only request forms of the `LDAPMessage` are sent by the client and response forms sent by the server. Typically, a client will initiate a *bind, operation*$_1$*, ..., operation*$_N$*, unbind* sequence, where each operation is either a search, compare, or one of a family of update operations. A client may also choose to *abandon* an operation before unbinding and closing or aborting the TCP connection or the client may choose to re-bind on an existing TCP connection with new credentials.

The LDAPMessage is defined in ASN.1 as follows:

```
LDAPMessage ::= SEQUENCE {
        messageID       MessageID,
        protocolOp      CHOICE {
                bindRequest     BindRequest,
                bindResponse    BindResponse,
                unbindRequest   UnbindRequest,
                searchRequest   SearchRequest,
                searchResEntry  SearchResultEntry,
                searchResDone   SearchResultDone,
                searchResRef    SearchResultReference,
                modifyRequest   ModifyRequest,
                modifyResponse  ModifyResponse,
                addRequest      AddRequest,
                addResponse     AddResponse,
                delRequest      DelRequest,
                delResponse     DelResponse,
                modDNRequest    ModifyDNRequest,
                modDNResponse   ModifyDNResponse,
                compareRequest  CompareRequest,
                compareResponse CompareResponse,
                abandonRequest  AbandonRequest,
                extendedReq     ExtendedRequest,
                extendedResp    ExtendedResponse },
          controls       [0] Controls OPTIONAL }
```

An `LDAPMessage` is converted to bytes using a BER encoding, and the resulting series of bytes is sent on the TCP connection. In LDAP only the BER definite length fields are used, so the receiver of a PDU knows how long the PDU will be as soon as the type tag and length of the outermost `LDAPMessage`

SEQUENCE has been read from the network. The use of definite length encodings allows LDAP PDU processing on the server side to be done very efficiently, since knowing the full length of the incoming PDU from reading the first few bytes leads to efficient memory management and minimization of data copying while reading bytes from the network.

The messageID field is an INTEGER. When the client sends a request it chooses a value for the messageID, which is distinct from that of any other message the client has recently sent on that connection, typically by incrementing a counter for each message. Any LDAPMessage PDUs returned by the server in response to that request will use the same messageID field. This enables a client to send multiple requests consecutively on the same connection, and servers that can process operations in parallel (for example if they are multi-threaded), will return the results to each operation as it is completed.

Normally the server will not send any LDAPMessage to a client except in response to one of the above requests. The only exception is the unsolicited notification, which is represented by an extendedResp form of LDAPMessage with the messageID set to 0. The notice of disconnection allows the server to inform the client that it is about to abruptly close the connection. However, not all servers implement the notice of disconnection, and it is more typical that a connection is closed due to problems with the network or the server system becoming unavailable.

The controls field allows the client to attach additional information to the request, and for the server to attach data to the response. Controls have been defined to describe server side sorting, paging and scrolling of results, and other features that are specific to particular server implementations.

In the C API, an application indicates that it wishes to establish a connection using the *ldap_init* call.

```
LDAP *ldap_init(const char *host,int port);
```

The *host* argument is either the host name of a particular server, or a space-separated list of one or more host names. The default reserved TCP port for LDAP is 389. If a space-separated list of host names is provided, the TCP port for each host can be specified for each host, separated by a colon, as in "server-a:41389 server-b:42389". The TCP connection will be established when the client makes the first request call.

6.2 General Result Handling

The result PDU for most requests that have a result (all but Abandon and Unbind) is based on the following ASN.1 data type:

```
LDAPResult ::= SEQUENCE {
        resultCode      ENUMERATED,
        matchedDN       LDAPDN,
        errorMessage    LDAPString,
        referral        [3] Referral OPTIONAL }
```

The resultCode will take the value zero for a successfully completed operation, except for the compare operation. Other values indicate that the operation could not be performed, or could only partially be performed. Only LDAP resultCode values between 0 and 79 are used in the protocol, and most indicate error conditions; for example, noSuchObject indicating that the requested object does not exist in the directory. The LDAP C API uses resultCode values between 80 and 97 to indicate errors detected by the client library (e.g., out of memory).

In the remainder of this section, the C API will be described using the LDAP synchronous calls, which block until a result, if required by the operation, is returned from the directory server. These API calls are defined by standard convention to have the suffix '_s' appended to the procedure names. Client applications that need to multiplex several operations on a single connection, or to obtain entries from a search result as they are returned asynchronously by the directory server, will use the corresponding

asynchronous API calls. The synchronous and asynchronous calls generate identical LDAP messages, and hence are indistinguishable to the server. It is up to the client application to define either a synchronous and asynchronous model of interaction with the directory server.

6.3 Bind

The first request that a client typically sends on a connection is the bind request, to authenticate the client to the directory server. The bind request is represented within the `LDAPMessage` as follows:

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
        version                 INTEGER (1 .. 127),
        name                    LDAPDN,
        authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
        simple                  [0] OCTET STRING,
        sasl                    [3] SaslCredentials }
```

There are two forms of authentication: simple password-based authentication and SASL (Simple Authentication and Security Layer). The SASL framework is defined in [RFC 2222]. A common SASL authentication mechanism is DIGEST-MD5 as defined in [RFC 2831].

Most LDAP clients use the simple authentication choice. The client provides the user's distinguished name in the name field, and the password in the `simple` field. The `SaslCredentials` field allows the client to specify a SASL security mechanism to authenticate the user to the server without revealing a password on the network, or by using a non-password-based authentication service, and optionally to authenticate the server as well.

```
SaslCredentials ::= SEQUENCE {
        mechanism               LDAPString,
        credentials             OCTET STRING OPTIONAL }
```

Some SASL mechanisms require multiple interactions between the client and the server on a connection to complete the authentication process. In these mechanisms the server will provide data back to the client in the `serverSaslCreds` field of the bind response.

```
BindResponse ::= [APPLICATION 1] SEQUENCE {
     COMPONENTS OF LDAPResult,
     serverSaslCreds    [7] OCTET STRING OPTIONAL }
```

The client will use the server's credential to compute the credentials to send to the server in a subsequent bind request.

In the C API, an application can perform a simple bind and block waiting for the result using the *ldap_simple_bind_s* call.

```
int ldap_simple_bind_s(LDAP *ld,const char *dn,const char *password);
```

6.4 Unbind

The client indicates to the server that it intends to close the connection by sending an unbind request. There is no response from the server.

```
UnbindRequest ::= [APPLICATION 2] NULL
```

In the C API, an application can send an unbind request and close the connection using the *ldap_unbind* call.

```
int ldap_unbind(LDAP *ld);
```

6.5 ExtendedRequest

The extended request enables the client to refer operations that are not part of the LDAP core protocol definition. Most extended operations are specific to a particular server's implementation.

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }

ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }
```

6.6 Searching

The search request is defined as follows:

```
SearchRequest ::= [APPLICATION 3] SEQUENCE {
        baseObject      LDAPDN,
        scope           ENUMERATED {
                baseObject              (0),
                singleLevel             (1),
                wholeSubtree            (2) },
        derefAliases    ENUMERATED,
        sizeLimit       INTEGER (0 .. maxInt),
        timeLimit       INTEGER (0 .. maxInt),
        typesOnly       BOOLEAN,
        filter          Filter,
        attributes      AttributeDescriptionList }
```

The `baseObject` DN and `scope` determine which entries will be considered to locate a match. If the scope is `baseObject`, only the entry named by the distinguished name in `baseObject` field will be searched. If the scope is `singleLevel`, only the entries immediately below the `baseObject` entry will be searched. If the scope is `wholeSubtree`, then the entry named by `baseObject` and all entries in the tree below it are searched.

The `derefAliases` specifies whether the client requests special processing when an alias entry is encountered. Alias entries contain an attribute with a DN value that is the name of another entry, similar in concept to a symbolic link in a UNIX® file system. Alias entries are not supported by all directory servers and many deployments do not contain any alias entries.

The `sizeLimit` indicates the maximum number of entries to be returned in the search result, and the `timeLimit` the number of seconds that the server should spend processing the search. The client can provide the value 0 for either to specify "no limit".

The `attributes` field contains a list of the attribute types that the client requests be included from each of the entries in the search result. If this field contains an empty list, then the server will return all attributes of general interest from the entries. The client may also request that only types be returned, and not values.

The LDAP Filter is specified in the protocol encoding using ASN.1, however most client APIs allow a simple text encoding of the filter to be used by applications. This textual encoding is defined in [RFC 2254].

In LDAP search processing a filter when tested against a particular entry can evaluate to TRUE, FALSE or Undefined. If the filter evaluates to FALSE or Undefined then that entry is not returned in the search result set. Each filter is grouped by parenthesis, and the most common types of filter specify the "present", "equalityMatch", "substrings", "and" and "or" filter predicates. The "present" filter evaluates to TRUE if an attribute of a specified type is present in the entry. It is represented by following the type of the attribute with "=*", as in `(telephoneNumber=*)`. The "equalityMatch" filter evaluates to TRUE if an attribute in the entry is of a matching type and value to that of the filter. It is represented as the type of the attribute, followed by a "=", then the value, as in `(cn=John Smith)`. The "substring" filter evaluates to TRUE by comparing the values of a specified attribute in the entry to the pattern in the filter. It is represented as the type of the attribute, followed by a "=", and then any of following, separated by "*" characters:

- a substring that must occur at the beginning of the value,
- substrings that occur anywhere in the value,
- a substring that must occur at the end of the value.

For example, a filter `(cn=John*)` would match entries which have a commonName (cn) attribute beginning with the string "John". A filter `(cn=*J*Smith)` match entries which have a value that contains the letter "J", and ends with "Smith". Many servers have restrictions on the substring searches that can be performed. It is typical for servers to restrict the minimum substring length.

An "and" filter consists of a set of included filter conditions, all of which must evaluate to TRUE if an entry is to match the "and" filter. This is represented using the character "&" followed by the set of included filters, as in `(&(objectClass=person)(sn=smith)(cn=John*))`.

An "or" filter consists of a set of included filter conditions, any of which must evaluate to TRUE if an entry is to match the "or"filter. This is represented using the character "|" followed by the set of included filters, as in `(|(sn=smith)(sn=smythe))`.

The "not" filter consists of a single included filter, whose sense is inverted: TRUE becomes FALSE, FALSE becomes TRUE, and Undefined remains as Undefined. The negation filter is represented using the character "!" followed by the included filter, as in `(!(objectClass=device))`. Note that the negation filter applies to a single search filter component, which may be compound. Most LDAP servers cannot efficiently process the "not" filter, so it should be avoided where possible.

Other filters include the `approxMatch` filter, the `greaterOrEqual`, the `lessOrEqual`, and the `extensible` filter, which are not widely used. The approximate matching filter allows for string matching based on algorithms for determining phonetic matches, such as soundex, metaphone, and others implemented by the directory server.

6.7 Search Responses

The server will respond to the search with any number of `LDAPMessage` PDUs with the `SearchResultEntry` choice, one for each entry which matched the search, as well as any number of `LDAPMessage` PDUs with the `SearchResultReference` choice, followed by an `LDAPMessage` with the `SearchResultDone` choice.

```
SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
        objectName      LDAPDN,
        attributes      PartialAttributeList }

SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL

SearchResultDone ::= [APPLICATION 5] LDAPResult
```

The `SearchResultReference` is returned by servers which do not perform chaining, to indicate to the

client that it must progress the operation itself, by contacting other servers. For example, if server S1 holds `dc=example,dc=com` , servers S2 and S3 hold `ou=People,dc=example,dc=com` and server S4 holds `ou=Groups,dc=example,dc=com`, then a `wholeSubtree` search sent to server S1 would result in the two `LDAPMessage` PDUs containing `SearchResultReference` being returned, one with the URLs:

```
ldap://S2/ou=People,dc=example,dc=com
ldap://S3/ou=People,dc=example,dc=com
```

and the other with the URL:

```
ldap://S4/ou=Groups,dc=example,dc=com
```

followed by a `SearchResultDone`.

Invoking a search request in the C API, and blocking for the results, uses the *ldap_search_s* call.

```
int ldap_search_s(
                    LDAP            *ld,
                    const char      *basedn,
                    int             scope,
                    const char      *filter,
                    char            **attrs,
                    int             attrsonly,
                    LDAPControl     **serverctrls,
                    LDAPControl     **clientctrls,
                    struct timeval  *timeout,
                    int             sizelimit,
                    LDAPMessage     **res);
```

The scope parameter can be one of *LDAP_SCOPE_BASE, LDAP_SCOPE_ONELEVEL*, or *LDAP_SCOPE_SUBTREE*.

6.8 Abandoning an Operation

While the server is processing a search operation, the client can indicate that it is no longer interested in the results by sending an abandon request, containing the `messageID` of the original search request.

```
        AbandonRequest ::= [APPLICATION 16] MessageID
```

The server does not reply to an abandon request, and no further results for the abandoned operation are sent.

In the C API, the client requests that an operation it invoked on that connection with an earlier asynchronous call be abandoned using the *ldap_abandon* call.

```
int ldap_abandon(LDAP *ld,int msgid);
```

6.9 Compare Request

The compare operation allows a client to determine whether an entry contains an attribute with a specific value. The typical server responses will be the `compareFalse` or `compareTrue` results codes, indicating that the comparison operation failed or succeeded. In practice, few client applications use the compare operation.

```
        CompareRequest ::= [APPLICATION 14] SEQUENCE {
```

```
                entry           LDAPDN,
                ava             AttributeValueAssertion}

        AttributeValueAssertion ::= SEQUENCE {
                attributeDesc    AttributeDescription,
                assertionValue  OCTET STRING }
```

In the C API, the client requests a comparison on an attribute with a string syntax using the *ldap_compare_s* call. Note that a successful comparison is expressed with the result code *LDAP_COMPARE_TRUE* rather than *LDAP_SUCCESS*.

```
int ldap_compare_s(LDAP *ld,const char *dn,
                   const char *type, const char *value);
```

6.10 Add, Delete, Modify and ModifyDN Operations

The Add, Delete, Modify and ModifyDN operations operate on individual entries in the directory tree.

```
        ModifyRequest ::= [APPLICATION 6] SEQUENCE {
                object          LDAPDN,
                modification    SEQUENCE OF SEQUENCE {
                        operation       ENUMERATED {
                                                add     (0),
                                                delete  (1),
                                                replace (2) },
                        modification    AttributeTypeAndValues } }
```

In the C API, the client can invoke the modify operation using the *ldap_modify_s* call.

```
int ldap_modify_s(LDAP *ld,const char *dn,LDAPMod **mods);

typedef struct LDAPMod {
            int mod_op;
#define LDAP_MOD_ADD            0x0
#define LDAP_MOD_DELETE         0x1
#define LDAP_MOD_REPLACE        0x2
#define LDAP_MOD_BVALUES        0x80
            char *mod_type;
            union mod_vals_u {
                  char **modv_strvals;
#define mod_values  mod_vals.modv_strvals
                  struct berval **modv_bvals;
#define mod_bvalues mod_vals.modv_bvals
            } mod_vals;
 LDAPMod;
```

The *mod_op* field is based on one of the values *LDAP_MOD_ADD*, *LDAP_MOD_DELETE* or *LDAP_MOD_REPLACE*.

The Add operation creates a new entry in the directory tree.

```
        AddRequest ::= [APPLICATION 8] SEQUENCE {
                entry           LDAPDN,
                attributes      SEQUENCE OF AttributeTypeAndValues }
```

In the C API, the client can invoke the add operation using the *ldap_add_s* call.

```
int ldap_add_s(LDAP *ld,const char *dn,LDAPMod **attrs);
```

The Delete operation removes a single entry from the directory.

```
DelRequest ::= [APPLICATION 10] LDAPDN
```

In the C API, the client can invoke the delete operation using the *ldap_delete_s* call.

```
int ldap_delete_s(LDAP *ld,const char *dn);
```

The ModifyDN operation can be used to rename or move an entry or an entire branch of the directory tree. The entry parameter specifies the DN of the entry at the base of the tree to be moved. The `newrdn` parameter specifies the new relative distinguished name (RDN) for that entry. The `deleteoldrdn` parameter controls whether the previous RDN should be removed from the entry, or just be converted by the server into ordinary attribute values. The `newSuperior` field, if present, specifies the name of the entry that should become the parent of the entry to be moved.

```
ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry           LDAPDN,
        newrdn          RelativeLDAPDN,
        deleteoldrdn    BOOLEAN,
        newSuperior     [0] LDAPDN OPTIONAL }
```

Many directory servers do not support the full range of capabilities implied by the ModifyDN operation (e.g., subtree rename) so this operation is not frequently used by clients.

7 CONCLUSION

This history of the evolution of LDAP technology is indeed a unique and fascinating case study in the evolution of a key Internet protocol and client-server technology used worldwide. There are many excellent client and server products based on LDAP available from several companies, each providing its various advantages and disadvantages. However, it is fair to say that it was through the diligent efforts of a few dedicated individuals, the emergence of the Internet as a commercially viable technology, and the financial investment of several research organizations and corporations into this technology that has made directories based on the lightweight directory access protocol critical components of the world-wide public Internet, most corporate and organizational wired and wireless intranets, and the global wireless phone network.

8 ACKNOWLEDGMENTS

The authors wish to thank K. C. Francis, Steve Kille, Scott Page, Stephen Shoaff, Kenneth Suter, Nicki Turman, and Neil Wilson for their insightful suggestions and editorial comments on earlier drafts of this article.

9 REFERENCES

Marshall T. Rose and Dwight E. Cass. *ISO Transport Services on top of TCP: Version 3*. Internet RFC 1006, May 1987.

Paul Barker and Steve Kille. *The COSINE and Internet X.500 Schema*. Internet RFC 1274, November 1991.

Wengyik Yeong, Tim Howes, and Steve Kille. *Lightweight Directory Access Protocol*. Internet RFC 1777, March 1995.

Mark Wahl, Tim Howes, and Steve Kille. *Lightweight Directory Access Protocol (v3)*. Internet RFC 2251,

December 1997.

Mark Wahl, Andrew Coulbeck, Tim Howes, and Steve Kille. *Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions*. Internet RFC 2252, December 1997.

Mark Wahl, Steve Kille, and Tim Howes. *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names*. Internet RFC 2253, December 1997.

Tim Howes. *The String Representation of LDAP Search Filters*, Internet RFC 2254, December 1997.

Tim Howes and Mark Smith. *The LDAP URL Format*. Internet RFC 2255, December 1997.

Mark Wahl. *A Summary of X.500(96) User Schema for use with LDAPv3*, Internet RFC 2256, December 1997.

Gordon Good. *The LDAP Data Interchange Format (LDIF) - Technical Specification*, Internet RFC 2849, June 2000.

Frank Yergeau. *UTF-8, a transformation format of Unicode and ISO 10646*. Internet RFC 2044, October 1996.

Mark Smith, Andrew Herron, Tin Howes, Mark Wahl and Anoop Anantha. *The C LDAP Application Program Interface.* Internet draft-ietf-ldapext-ldap-c-api-05.txt, November 2001.

John Myers. *Simple Authentication and Security Layer (SASL),* Internet RFC 2222, October 1997.

Paul Leach and Chris Newman. *Using Digest Authentication as a SASL Mechanism*. Internet RFC 2831, May 2000.

International Telecommunications Union (ITU). *Information technology - Open Systems Interconnection Recommendation X.500 - The Directory: Overview of concepts, models and services*, 1993.

International Telecommunications Union (ITU). *Information technology - Open Systems Interconnection Recommendation X.501 - The Directory: Models*, 1993.

International Telecommunications Union (ITU). *Information technology - Open Systems Interconnection Recommendation X.511 - The Directory: Abstract service definition*, 1993.

International Telecommunications Union (ITU). *Information technology - Open Systems Interconnection Recommendation X.518 - The Directory: Procedures for distributed operation*, 1993.

International Telecommunications Union (ITU). *Information technology - Open Systems Interconnection Recommendation X.519 - The Directory: Protocol specifications,* 1993.

International Telecommunications Union (ITU). *Information technology - Open Systems Interconnection Recommendation X.520 The Directory: Selected attribute types*, 1993.

International Telecommunications Union (ITU). *Information technology - Open Systems Interconnection Recommendation X.521 - The Directory: Selected object classes*, 1993.

International Telecommunications Union (ITU). *Information technology - Open Systems Interconnection Recommendation X.525 - The Directory: Replication*, 1993.

International Telecommunications Union (ITU). *Information technology Recommendation X.681 – Abstract Syntax Notation One (ASN.1): Specification of basic notation, 1994.*

International Telecommunications Union (ITU). *Information technology Recommendation X.690 - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), 1994.*

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler, (ed.). *Extensible markup language (XML) 1.0 (second edition)*. W3C, October 2000.

Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrick Nielsen. *SOAP version 1.2 part 1: Messaging framework*. W3C, June 2003.

Organization for the Advancement of Structured Information Standards. *Directory Services Markup Language v2.0*, December 2001.

Tom Bellwood (ed.). *UDDI Version 2.04 API Specification*. July 2002.

Organization for the Advancement of Structured Information Standards. *OASIS/ebXML Registry Services Specification v2.0*. April 2002.

Darren Rolls (ed.). *Service Provisioning Markup Language (SPML) Version 1.0*. OASIS Technical Committee Specification, June 2003.

Rob Weltman. *LDAP Proxied Authorization Control*. Internet draft-weltman-ldapv3-proxy-12.txt, April 2003.

10 Author Bios

Greg Lavender is currently a Director of Software Engineering and CTO for Internet directory, network identity, communications, and portal software products at Sun Microsystems. He was formerly Vice President of Technology at Innosoft International, co-founder and Chief Scientist of Critical Angle, an LDAP directory technology startup company, and co-founder and Chief Scientist of the ISODE Consortium, which was an open-source, not-for-profit, R&D consortium that pioneered both X.500 and LDAP directory technology. He is also an Adjunct Associate Professor in the Department of Computer Sciences at the University of Texas at Austin.

Mark Wahl is currently a Senior Staff Engineer and Principal Directory Architect at Sun Microsystems. He was previously Senior Directory Architect at Innosoft International, co-founder and President of Critical Angle, and lead directory server engineer at the ISODE Consortium. He was the co-chair of the IETF Working Group on LDAPv3 Extensions, and co-author and editor-in-chief of the primary LDAPv3 RFCs within the IETF.