CITI Technical Report 95–7

A Scalable, Deployable Directory Service Framework for the Internet

Timothy A Howes
tim@umich.edu

Mark C. Smith
mcs@umich.edu

ABSTRACT

This paper describes a directory service framework for the Internet that fits within the approach outlined in the IETF's RFC 1588. This framework consists of a global directory service that enables virtually any local directory service to operate under it. We also include an optimized local directory service, thereby providing a complete solution for Internet directory service. Our approach uses proven Internet technology (e.g., the Domain Name System and Uniform Resource Locators) and successful or promising pieces of other services (e.g., X.500 and WHOIS++). Previous attempts to create a unified Internet directory service, such as X.500, LDAP, WHOIS++, and SOLO, have not been fully accepted because of difficulties in implementation and deployment. Therefore, we designed our approach with ease of implementation and deployment in mind. To that end, our approach attempts to co-opt the installed base making a switch to the new service as seamless as possible.

July 11, 1995

Center for Information Technology Integration
University of Michigan
519 West William Street
Ann Arbor, MI 48103-4943

A Scalable, Deployable, Directory Service Framework for the Internet

Timothy A. Howes and Mark C. Smith

July 11, 1995

1. Introduction

Despite many efforts to create a comprehensive directory service for the Internet, no such service currently exists. Though most sites run a local service, there is no global framework capable of tying them together into a uniform service, useful throughout the Internet. Early efforts to mandate a solution by encouraging sites to switch from their local directory service to a standard one (X.500) have largely failed, despite pockets of successful X.500 sites and a slowly growing global X.500 service.

In Fall of 1993, the Internet Engineering Task Force (IETF) adopted a more pragmatic approach toward developing an Internet directory service, one that takes into account the diverse installed base and allows development of alternative global systems. The goal of this egalitarian approach, documented in RFC 1588 [7], remains to arrive at a global service, but to do so in a way that does not favor one candidate service over another, allowing a natural process of selection to take place.

We believe ease of implementation and deployment are qualities that will ultimately select the directory service winner on the Internet, and these goals have driven the design of our service. This paper describes a global directory service framework and an example local directory service that fit within the approach outlined in RFC 1588. Our approach uses proven Internet technology (e.g., the Domain Name System and Uniform Resource Locators) and successful or promising pieces of other services (e.g., X.500 and WHOIS++).

The framework is general enough to support sites running virtually any local protocol, while encouraging migration to our more functional local service. Making the migration optional enables sites with set local services to still participate in the global service. The local service component of the framework is based on a stand-alone version of the Lightweight Directory Access Protocol (LDAP) [3,4]. LDAP was formerly used exclusively as a front-end to the X.500 directory. This choice makes our service highly compatible with X.500, but free of its shortcomings.

The following sections describe various aspects of the global framework and our optimal local service. We finish with some examples, a discussion of our implementation of the system, and prospects for deployment throughout the Internet.

2. The Global Service Framework

The global directory service framework we define is described in the following sections. We start with the namespace and then give an overview of navigation, or how one gets from a name to a server holding data on that name. This section forms the basis for how our system solves the directory service "read" problem. The co-opting of the installed base is tackled next. Our solution to the directory service "search" problem is then presented, providing the basis for information retrieval when the name of an object in the directory is not known.

2.1 Namespace

A namespace plays a key role in a directory service, allowing efficient reference and retrieval of collections of related information. Our service piggy-backs on existing DNS infrastructure to build a directory service namespace for the Internet similar to what is already used for e-mail. This approach makes implementation and deployment easier by using familiar and efficient names and avoiding the need for cumbersome registration procedures.

In some services, e.g., X.500 [1,2] and the Domain Name System (DNS), the namespace is explicit and hierarchical. A hierarchical namespace is characteristic of most global systems. It allows information to be subdivided easily, authority to be distributed, new information to be added with little disruption of the rest of the system, and straightforward navigation through the namespace. Unfortunately, such a namespace can require relatively complicated management, and a site must understand and endure some registration process before it can "get connected." These are barriers to deployment.

A hierarchy is not the only option. Services such as WHOIS++ [9] and SOLO [8] favor a more loosely organized "mesh" structure. In this model, information is referred to by its content. An entry in the mesh may have as many "names" as it has collections of attributes that uniquely identify it. New sites come online by announcing their presence to one or more "neighbors," or to a more formal registration authority. In this way, the mesh may grow from the bottom up, or "out," as the case may be. The barriers to getting connected are low, but without careful administration this method can lead to the same scaling problems and tangled information structure that characterize the World Wide Web (WWW). In a directory system meant for organizing and finding information, this lack of organization is a bad thing.

User acceptance is another barrier to deployment that any namespace faces. Internet users have to deal with many different namespaces: an IP address namespace, a host namespace, and several electronic mail and information resource namespaces, to name a few. To succeed, another namespace must bring with it enough benefit to out-weigh the inconvenience that it poses to users. Large and unwieldy names, such as those proposed by X.500, fight an uphill battle for acceptance. Names that users cannot easily grasp or put on a business card, such as those proposed by WHOIS++, are also at a disadvantage. Names that are hard to remember, or have no obvious relationship to the objects they name will be difficult for users to accept.

To ensure user acceptance, our proposal relies heavily on the DNS, which is already an integral part of the Internet. The DNS is scalable, familiar to users and administrators, and has well-established registration procedures. Just as RFC 822 piggy-backs on the DNS to define an e-mail namespace for the Internet, we build on the DNS to define an Internet directory service namespace. Our namespace has the same l*ocal@domain* syntax as an Internet e-mail address, and similar semantics. The *domain* part identifies the server holding the information, while the *local* part has meaning only to the local directory server.

Aside from having an aesthetic appeal (for example, our directory names in this scheme are *tim@umich.edu* and *mcs@umich.edu*), this scheme has the advantage of being familiar to both users and administrators. Having gone without a true directory service for so long, Internet users have gotten used to dealing directly with e-mail addresses, to which our directory names bear a striking resemblance. The names are sufficiently compact and efficient to be carried easily in protocol, while retaining enough mnemonic value to please users.

Furthermore, because we piggy-back on the existing DNS infrastructure, the registration problem is neatly averted. A site must register with the DNS to get connected to the Internet in the first place, and by doing so it simultaneously (and unwittingly) registers in our directory service namespace as well.

2.2 Navigation: The Directory Service Read Problem

Navigation is the process of finding which servers contain the data corresponding to a name. This process is the essential component in any solution to the directory service read problem—given the full name, return the information associated with that name. Directory lookups of this kind must be fast and efficient, without requiring any user interaction. Often a program does the lookup, perhaps in the course of delivering a piece of mail or accessing a resource.

In a read operation, the client specifies the full name of the object. Contrast this with the less constrained search problem, which we discuss in detail later. A search may begin with partially specified criteria, contact several servers, and require subsequent interaction with a user or other agent before the search is finished. Most directory services treat the read and search operations as substantially the same. We feel they have different enough requirements to warrant different technical solutions.

When it comes to returning information associated with a name, few systems compare in scale or efficiency to the Internet's DNS. The DNS has a variety of uses, from host name to address translation, to the reverse mapping, to e-mail service location for a domain name. We use an approach analogous to this latter function for our directory service. Among the records associated with a name in the DNS can be a mail exchange, or MX, record. The purpose of this record is to provide a mapping from a domain name to the name of one or more hosts performing mail service for that domain. Because our goal is to provide a mapping from a domain name to a host performing directory service for a domain, the analogy is direct.

Our service defines a directory exchange, or DX, record. The DX record is functionally similar to an MX record, except that it points to a host performing directory service for a domain instead of e-mail service. As with an MX record, multiple hosts can be specified, with different preferences. In fact, as the next section describes, a DX record actually specifies a protocol for retrieving directory information in addition to a host name.

A directory client begins with a name of the form *localpart@domain*. It looks up *domain* in the DNS, asking for DX or address records. If no DX records are found, it assumes an LDAP service is running on the standard port at the given address. If one or more DX records are found, the client chooses among them, based on priority and protocol. It contacts the corresponding server using the appropriate protocol and retrieves the information.

Using DX records has several advantages over other schemes:

- The technology on which they are based is familiar and scalable, proven by years of use in the Internet community.
- The concepts are well-understood by system administrators who already interact with the DNS to use and provide other Internet services.
- All that is necessary to "get connected" to our directory service, using any local service a site wants, is to add a simple record to the DNS. If a site chooses to run our version of stand-alone LDAP as their local service even that is not necessary. (See the section 5, Implementation and Deployment, for a discussion of our

implementation of DX records using TXT records.)

2.3 Installed Base

Any new service that does not take into account a large installed base faces an uphill battle for acceptance and ultimate deployment. For a successful Internet directory service, we strive for as easy and seamless an upgrade path as possible. Co-opting the installed base so that it actually becomes part of the new service is an even bigger win.

The World Wide Web is an example of such an approach in action. The Web incorporated Gopher, FTP, and other protocols, immediately making their information available through its own model, increasing its own value. We strive for the same level of success by including existing directory services in our own. To do so, we employ the same tool used successfully by the Web: the Uniform Resource Locator, or URL.

The models we have described for namespace and navigation could be easily applied to a number of directory services. Our goal is to broaden them to apply to all services at once. To do this, we extend the DX record concept introduced above. We allow a DX record to be multivalued and expand its content to allow a partial URL, rather than a simple hostname. A client looking up DX records for a directory name is returned one or more partial URLs, pointing to the directory service(s) for the domain. Intelligent clients can then choose the access method they prefer. This method also supports redundant replicated servers.

The URLs returned are "partial"; they contain only enough information to identify the desired server and the protocol used for accessing it. The rest of the URL, enabling the client to retrieve the information it desires, is contained in the *localpart* of the directory name. How the full URL is constructed and used depends on each protocol. For example, if the local directory service is a CSO nameserver, the *localpart* of the name might be the desired entry's unique identifier. If the local directory service is a WHOIS or WHOIS++ server, the *localpart* of the name might be the record's handle. We do not define the exact mappings here, but the approach is clear.

The advantages of this scheme are several:

- By using URLs, we employ technology familiar to administrators.
- It is relatively easy to add new protocols to the mix—all that's needed is the definition of a new URL format.
- In addition to incorporating existing directory services, URLs have the potential to make our service easily available to the wide variety of WWW clients already familiar to users.

2.4 The Search Problem: Dealing with Multiple Servers

So far, the framework we've described addresses part of the directory service problem—it enables quick and efficient information retrieval from a single server, given a name uniquely identifying it. Although this function is vital to the success of any service, the ability to search for and retrieve information from multiple servers poses a more interesting and challenging problem. The goal is, given some search criteria, to determine which servers contain the requested information, and then to retrieve it. Our solution is based on the standard LDAP model for searching, with the addition of centroids to help prune the search space of servers.

Efficient resource consumption is the primary constraint during the search process. For searches with a naturally restricted scope, this is a minor problem. If only one or a few servers are involved, it is not unreasonable to contact them all with the search request. For wider-area searches, this approach is not feasible.

For example, keeping in mind the namespace model described above, suppose a user wants to search for a friend named

"Babs Jensen." The problem is that our user met Babs while on spring break and knows only that she is in college "somewhere up north." A simple-minded approach would be to extend the search to every directory server in the *edu* domain. (At last count, this domain contained over 1,000,000 hosts.) Contacting them all would leave our user waiting quite a while for the results of his query. If more than a few users produced such queries, the network and server resources consumed would soon bring the Internet itself to its knees. Clearly, a better method of handling such searches is needed.

In our system, a client performing the search described above would begin by looking up DX records for the name *edu* in the usual way (or perhaps through cached knowledge of the "root" information, as is currently done in the DNS). Depending on its preferred choice of protocol, the client picks one of the returned URLs and connects to the indicated service to perform the search. Our only requirement on this service is that it return *DX referrals* to the client, indicating likely places for the client to continue the search.

A DX referral is similar to the partial URL defined in Section 2.2. It provides a level of indirection to the information it references, allowing control over the protocol(s) used to access the information to remain in the hands of the site providing the service. For example, the search for Babs in the *edu* domain might return two DX referrals, *dx://umich.edu* and *dx://cornell.edu*. The actual protocols and directories that contain the information are discovered by the client in the normal way: by looking up DX records for the names *umich.edu* and *cornell.edu*, respectively. If a site does not need the indirection provided by a DX referral, it may elect to return a protocol-specific partial URL directly.

The efficiency with which searches are conducted depends on how well the high-level servers initially contacted prune the searchspace. These servers return DX referrals only to those directories which, when searched, will return positive results to the query. Return too few referrals and some information vital to the user may be lost. Return too many, and we revert to the "contact every server" approach.

Because we allow many high-level protocols to provide the searching service in our scheme, it is not really for us to say how they might make this happen. But in our implementation of LDAP, described more fully in the next section, we make use of centroid indexing [6] for this purpose. Originally adapted for use in WHOIS++ as a means of both searching and navigating through the mesh of servers, we adopt the technique as a searchspace-pruning device in LDAP.

Our solution to the searching problem has several advantages:

- It allows any protocol that can support the namespace we define to participate as wide-area-search servers. In contrast, other proposals that mandate a particular protocol for searching fix the query language, the information model, etc.
- Our approach is based on DX referrals, which introduce a useful level of indirection, again leading to more flexibility and control by local sites.
- While we propose the use of centroids as a method of pruning the searchspace, particularly in our implementation of LDAP, our scheme is not tied to centroid technology. Should another distributed indexing mechanism become available, it can be incorporated into our scheme easily. The details of searchspace pruning are hidden from the user, who sees only a familiar and consistent view of the namespace.

3. The Local Service: Stand-Alone LDAP

Having described the global framework into which any end-user directory can fit, we now turn our attention to the design of one such end-user system. Our system is based on the Lightweight Directory Access Protocol (LDAP), with a few simple extensions to make it fit better into the our global framework. LDAP was designed as a lightweight access method for X.500. In this mode, clients interact with a single LDAP server that interacts with one or more X.500 servers on the client's behalf. The LDAP server holds no data itself. All data is contained in the X.500 servers (Figure 1).

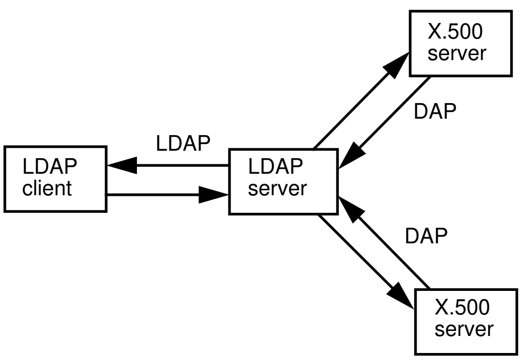

Figure 1. LDAP as a front end to X.500. The LDAP server chases referrals, merges results, etc. from multiple X.500 servers.

LDAP retains the basic X.500 model, supporting most X.500 operations, and the same namespace and information model. The protocol itself runs directly over TCP, leaving out many of the more esoteric and less often used X.500 features, with most data items encoded in simple character string formats. The simplified feature set, transport, and encoding, allow LDAP clients to be more easily developed and run using far fewer resources than their full X.500 counterparts.

We make two key extensions to LDAP, enabling it to function as a stand-alone directory service and to participate fully in the framework outlined above, both as a local service and as a wide-area search service. Both extensions are designed to be transparent to existing LDAP clients, making the new version compatible with the existing one. Old clients will not know enough to take advantage of the new features we introduce, but they will not break. We took a lesson from the developers of MIME here, who transparently enabled new features in RFC 822 e-mail, simply by defining new formats for message bodies that fit within the existing protocol. Our extensions are similar in philosophy.

First, we extend the LDAP Distinguished Name (DN) format to include the Internet-style names described earlier. Since these names are carried in LDAP as simple strings, there are no protocol changes necessary to support this. Second, LDAP allows the server to send back an arbitrary text message to the client as part of a search result. We impose some structure on this message, defining a format for the return of referrals within it. Clients that don't understand this format will not be able to follow the referrals, but they will not be bothered by their presence either.

The resulting interaction model is shown in Figure 2. In this configuration, data is held by the LDAP servers themselves, along with referrals pointing to different LDAP (or other protocol) servers.

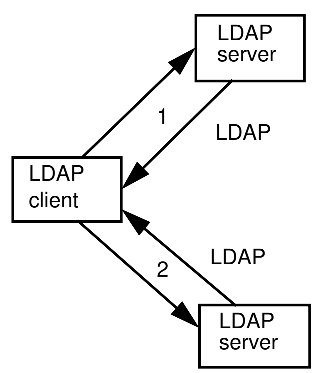

Figure 2. Stand-alone LDAP interaction. A client makes a request in transaction 1 and follows the returned referral in 2.

4. Example Interactions

This section describes in detail two typical interactions between an Internet directory client and the global and local services defined in this paper.

4.1 Example of the Search Capability

Our first example illustrates the search capability of the system. In this example, our goal is to discover the phone number of a user named Babs Jensen, a college student somewhere in Michigan.

1. A client looks up *edu* in the DNS, asking for DX records. The client may receive several records, but let's assume it chooses the one providing an LDAP service for the *edu* domain, *ldap://some.host*.

   The client contacts the LDAP port on *some.host* and sends it an LDAP subtree search of the *edu* domain for an entry with an ObjectClass of person, a locality of Michigan, and a name of Babs Jensen.

   The LDAP server consults the centroid index it has collected and finds that there are two possible services where the client should continue the search, one at the University of Michigan, and another at Michigan State University.

2. The corresponding DX referrals, which the LDAP server returns to the client, are *dx://umich.edu* and *dx://msu.edu*.
3. The client may present these referrals to the user, or it may follow them automatically. Choosing the latter, it looks up *umich.edu* and *msu.edu* in the DNS, asking for DX records.
4. For each set of DX records returned, the client chooses one corresponding to a protocol it understands and follows that record. In our example, the chosen records might be *ldap://umich.edu* and *cso://msu.edu*.
5. The client contacts each directory via the appropriate protocol and continues its search, retrieving the results it desires.

The five steps in this process correspond to the client transactions shown below in Figure 3 and Figure 4.

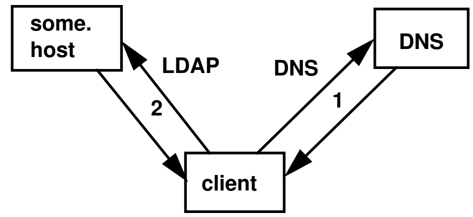

Figure 3. In transaction 1, the client retrieves DX records for the edu domain. In transaction 2, the client retrieves DX referrals to servers it should contact.

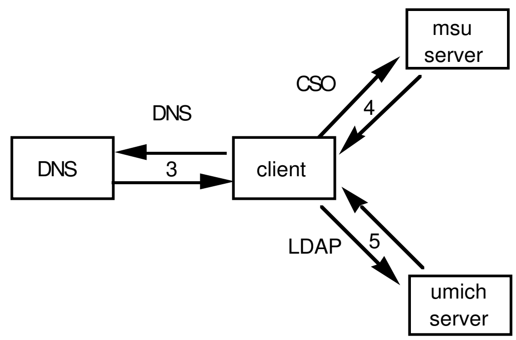

Figure 4. n transaction 3, the client retrieves DX records for each DX referral. In transactions 4 and 5, the client contacts the indicated servers via CSO and LDAP respectively, retrieving results.

4.2 Example of E-mail Processing

Our second example is of an e-mail user agent processing a piece of mail addressed to the directory name *babs@umich.edu*. The goal is to find the e-mail address to which to forward the message, illustrating the read capability of the system. This interaction is shown in Figure 5..

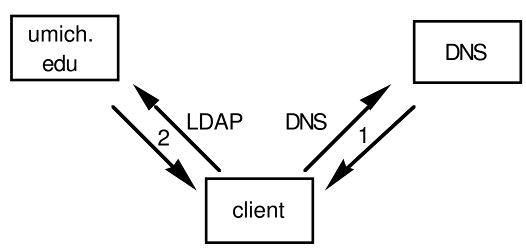

Figure 5. In transaction 1, the client retrieves DX records for umich.edu. In transaction 2, the client retrieves the desired information.

The client starts by looking up *umich.edu* in the DNS, asking for DX records. Among the DX records returned to it is one pointing to an LDAP service for the domain, *ldap://umich.edu*. The client contacts the LDAP server on the *umich.edu* host, giving it the name *babs@umich.edu* and asking for the e-mail address associated with the entry

5. Implementation and Deployment

As of this writing, we have implemented most of the global and local services described in this paper and are beginning to gain experience with deployment. More specifically:

- Our implementation of DX records in the DNS is built using TXT records, already supported in most nameservers.
- We have developed a stand-alone LDAP daemon, called *slapd*, that supports a variety of database backends, including one capable of holding and searching centroid indexes. How centroid data gets into the backend is still a problem, one we hope to solve by using an existing common indexing protocol server.
- We have modified our LDAP client library to handle the referrals returned by this server, either automatically or only when directed by the client program.
- Work is underway to make use of the new client library features in maX.500, waX.500 and xax500, our LDAP clients for the Macintosh, MS Windows, and X Windows, respectively.
- We also have plans to develop a forms-based Web interface capable of following multiprotocol referrals.
- Finally, we are working on support for replication in *slapd*, described in detail below.

Slapd has been designed with high performance, ease of configuration and use, and flexibility in mind. Implemented on a variety of UNIX systems, it is based on a single highly-threaded process. A new thread is created for each operation, maximizing concurrency. We developed our own threads "glue" library, based on draft 4 of the Posix standard. The library provides a mapping onto a number of underlying threads packages (we currently support three and have plans to add more). Support is provided for both preemptive and non-preemptive threads.

Slapd supports multiple database backends, allowing great flexibility when adapting to a site's environment. Communication with a backend is through a well-defined API, making it easy to add new or customized databases. We have implemented three backends so far: an */etc/passwd* backend; a "shell" backend; and a high-performance, fully-indexed hash or btree-based backend called *ldbm*. The relationship between *slapd* and its backends is shown in Figure 6. Multiple instances of each backend can be enabled by editing a simple tailor file. This allows a single *slapd* process to serve several portions of the namespace.

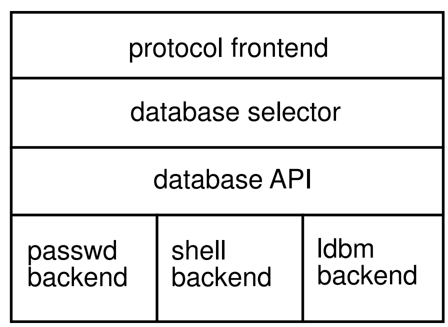

Figure 6. ***Slapd* architecture.** Supports multiple simultaneous database backends through a common API.

The shell backend allows the execution of arbitrary administrator-defined commands in response to queries. *Slapd* executes the command, feeding it a text representation of the request on standard input and reading a text representation of the results from standard output. Although not appropriate for high-performance environments, this method allows easy integration with virtually any system (e.g., via shell scripts).

The *ldbm* backend is based on technology we originally developed for use in our X.500 server. The underlying indexes are implemented by one of a number of dbm or btree packages. As with the threads support, we have developed a library of "glue" routines allowing the use of several different packages and easy addition of more (we support three so far). The *ldbm* backend can support efficient processing of all types of LDAP searches, including substring, approximate, and range queries. It is targeted at medium to large-scale databases of up to a few hundred thousand entries. It is not meant to handle millions of entries but should be sufficient for the needs of most sites.

We are in the process of adding replication support to *slapd*. Our approach is to have *slapd* log changes to a file which is then read by a separate process. The stand-alone LDAP update replication daemon (*slurpd*) is responsible for distributing the changes to the appropriate replicas (*slapd* records where each change should go in the replication log file). Changes are made to a replica using the normal LDAP protocol; no special replication protocol is required. *Slurpd* authenticates itself to each replica as a special entity allowed to make changes. This leads to a master and slave replication scheme. The interaction is depicted in Figure 7.

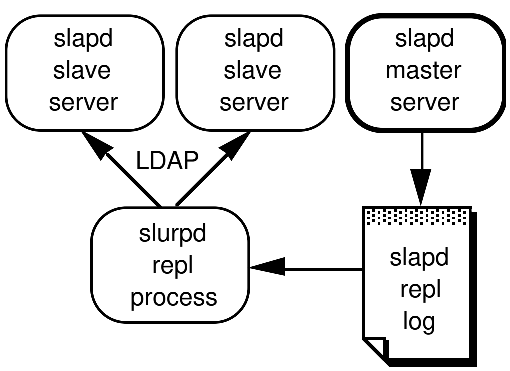

Figure 7. **Master and slave *slapd* replication. Interaction with *slurpd*.**

6. Availability

Our implementation is freely available using the URL *ftp://d.rs.itd.umich.edu/ldap/ldap.tar.Z*. We invite anyone interested to retrieve the software and participate on either a local or global level with our pilot. Sites wanting to participate strictly at a local level can just bring up one or more stand-alone LDAP daemons and point their local clients at them.

If the site wants to be visible to the outside world, it need only add a simple TXT/DX record to its local DNS server. If the site wants to participate in the global search service, it needs to run centroid generation and updating software. The software will automatically find and connect to the correct indexing server. If a site wants to participate with a directory service protocol other than LDAP, it can do so by adding a TXT/DX record to the DNS pointing to that service.

Deployment on a wider scale will begin as soon as we have gained more experience with the system, worked out the remaining bugs, and improved the documentation.

7. Summary

This paper defines a global framework and local directory service within that framework which follows the approach outlined in RFC 1588. The scheme described here removes the barriers to implementation and deployment encountered by other directory services, while incorporating existing local directory services impartially. The registration problem is sidestepped by using the existing DNS namespace. The likelihood of user acceptability is enhanced through the use of familiar names. System administrators likely already have the knowledge necessary to hook their site into the global system; all that's necessary is to add a simple record to the DNS. Migration from existing services is not required and can take place at leisure. Most of the system has been implemented, and work is ongoing on the remaining pieces.

8. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. NCR-9416667. Thanks also to Peter Honeyman for his valuable review of this paper.

References

1. The Directory: Overview of Concepts, Models and Service. CCITT Recommendation X.500, 1988.
2. Information Processing Systems – Open Systems Interconnection – The Directory: Overview of Concepts, Models and Service. ISO/ IEC JTC 1/SC21; International Standard 9594-1, 1988
3. W. Yeong, T. Howes, S. Kille, Lightweight Directory Access Protocol. Request For Comments (RFC) 1777, March, 1995.
4. T. Howes, S. Kille, W. Yeong, C. Robbins, The String Representation of Standard Attribute Syntaxes. RFC 1778, March 1995.
5. Tim Howes, Mark Smith, An LDAP URL Format. Internet Draft draft-ietf-asid-ldap-format-00.txt, March, 1995.
6. Chris Weider, The Common Indexing Protocol, Internet Draft draft-weider-comindex-00.txt, March 1995.
7. J. Postel, C. Anderson, White Pages Meeting Report. RFC 1588, February, 1994.
8. C. Huitema, P.-A. Pays, A. Zahm, A. Woermann, Simple Object Look-up Protocol (SOLO), Internet Draft draft-huitema-solo-02.txt
9. Peter Deutsch, Rickard Schoultz, Patrik Falstrom, Chris Weider, Architecture of the WHOIS++ Service. Internet Draft draft-ietf-wnils-whois-arch-03.txt, March 1995.

Author Information

Tim Howes is a Senior Systems Research Programmer for the University of Michigan's Information Technology Division. He received a B.S.E. in Aerospace Engineering, a M.S.E. in Computer Engineering from U-M, and is completing a Ph.D. in Computer Science. He is currently project director and principal investigator for the NSF-sponsored WINX project, and in charge of campus-wide directory service development and deployment at U-M. He is the primary architect and implementor of the U-M LDAP system, the GDA X.500 DSA, and a major developer of the QUIPU X.500 implementation. He is author or co-author of several RFCs including RFC 1777 and RFC 1778 defining the LDAP protocol. He is chair of the IETF Access, Searching, and Indexing of Directories working group, and an active member of the ACM and IEEE.

Mark Smith is a Project Leader and Systems Research Programmer for the University of Michigan's Information Technology Division, specializing in Directory Service, Unix, Macintosh, and AppleTalk and TCP/IP network programming. He received his B.S.E. in Computer Engineering from U-M and is completing an M.S.E. in Computer Engineering. He is currently leading X.500 Yellow Pages development and deployment at U-M. He is the architect and implementor of the popular maX.500 Directory User Agent, and is a major developer of the U-M LDAP implementation, and has made important contributions to several other Internet software packages for Unix and Macintosh. He is co-author of RFC 1249 describing the DIXIE protocol and an active member of the IETF.