LDAPtcl: A Tcl Interface to the Lightweight Directory Access Protocol

Randall Kunkee
Procplace.com, Inc.
Karl Lehenbauer
Procplace.com, Inc.

Abstract

This paper describes a project to create a Tcl interface to the Lightweight Directory Access Protocol (LDAP), an Internet protocol for accessing information services that has found increasingly widespread use in applications such as ISP management and user roaming. The LDAP protocol is described. Tradeoffs between different design approaches are explored and the one chosen is given grounds for. The evolution of the interface from managing LDAP interactions through a blocking, synchronous interface to one encompassing LDAP results as an event source through the Tcl notifier is chronicled. Finally, real-life experiences with the code in a production environment are detailed.

1. Directory-Oriented Internet Services

The value of serving out directory-oriented information over the Internet has long been recognized, and is reflected in a number of protocols that have evolved to provide it. Sun's Network Information System (NIS) allows groups of machines to share a common set of configuration files, permitting a system administrator to set up NIS client systems with only minimal configuration data and add, remove or modify configuration data from a single location. Likewise, Dynamic Host Configuration Protocol (DHCP) allows machines to query the network for boot information, including IP address, subnet mask, default gateway, etc., and the Remote Authentication Dial In User Service (RADIUS) provides a way for access servers to query a remote database to validate connections and to obtain configuration data for the connection.

2. What is LDAP?

LDAP defines both a directory-oriented database and a directory service protocol that runs over TCP/IP. LDAP provides a standard way to deliver the data required by the services mentioned above, plus new services, including application-specific ones.

LDAP implementations provide a C library for clients to use to make LDAP queries and get results.

LDAP version 3 is an Internet "proposed standard" and is documented by various RFCs. (See references.)

LDAP's popularity and increasing presence is summed up in an IBM Redbook*:

> LDAP is a fast growing technology for accessing common directory information. LDAP has been embraced and implemented in most network-oriented middleware. As an open, vendor-neutral standard, LDAP provides an extendable architecture for centralized storage and management of information that needs to be available for today's distributed systems and services.
>
> After a fast start, it can be assumed that LDAP has become the de facto access method for directory information, much the same as the Domain Name System (DNS) is used for IP address look-up on almost any system on an intranet and on the Internet. LDAP is currently supported in most network operating systems, groupware and even shrink-wrapped applications.

2.1. LDAP Directory Service Model

The LDAP directory service model is based on *entries*. An entry (or object) is a collection of attributes that has a name with one or more unique values. Each object is uniquely identified by a *Distinguished Name*, or DN, which is a series of name=value pairs concatenated together to describe the object's location in the directory tree. Distinguished names are the primary keys to entries in the directory. They are documented in RFC 1779. Examples of distinguished names are:

---

*Understanding LDAP. An online version is available at *http://www.redbooks.ibm.com/abstracts/sg245110.html*

CN=Marshall T. Rose, O=Dover Beach Consulting, L=Santa Clara, ST=California, C=US

device=lp1, dc=foo, dc=bar, dc=com, o=neosoft inc, c=US

Consider the following entry written in LDIF (LDAP Directory Interchange Format):

dn: CN=Marshall T. Rose, O=Dover Beach Consulting,
  L=Santa Clara, ST=California, C=US
objectClass: inetorgperson
cn: Marshall T. Rose
sn: Rose
givenname: Marshall
sn: Rose
st: California
l: Santa Clara
mail: mtrose@dbc.com
mail: marshall.t.rose@dbc.com
postalAddress: 123 Beach Street $ Santa Clara, CA
  94123
telephoneNumber: 213-123-4567
street: 123 Beach Street

Note that the elements of the distinguished name of an entry are not necessarily all defined within the entry. Attribute names are case independent. The case-sensitivity of attribute values depends on their type.

In this example, **CN** is the common name, **O** is the organization name, **L** is the locality name, **ST** is the state or province, and **C** is the country. Each of the entry's attributes has one or more values. The mail attribute above contains two values. There are a number of "operational" attributes maintained automatically by the server, such as **modifyTimeStamp** and **createTimeStamp**. The special attribute name **objectclass** contains values which determine which attributes are permissible. Since **objectclass** may also be multivalued, attributes defined by several types of objects may be held within a single entry. A well behaved directory has **objectclass** defined in all of its entries.

In LDAP, directory entities are arranged in a hierarchical structure that reflects political, geographic and/or organizational boundaries. An example LDAP directory tree is shown in Figure 1.

In the real world, your directory server will only contain a part of the universal tree. The root of your tree is your *suffix*, and all operations must be on this suffix or they are invalid. For example, the suffix for the above entry might be "O=Dover Beach Consulting, L=Santa Clara, ST=California, C=US". Unless your directory is going to be public, this could be shortened and all DNs redesigned for a suffix of "O=Dover Beach Consulting, C=US" or "O=DBC,C=US" or perhaps even "O=DBC", to simplify matters elsewhere.

The second example uses DC (domainComponent) to demonstrate an Internet-based hierarchy.

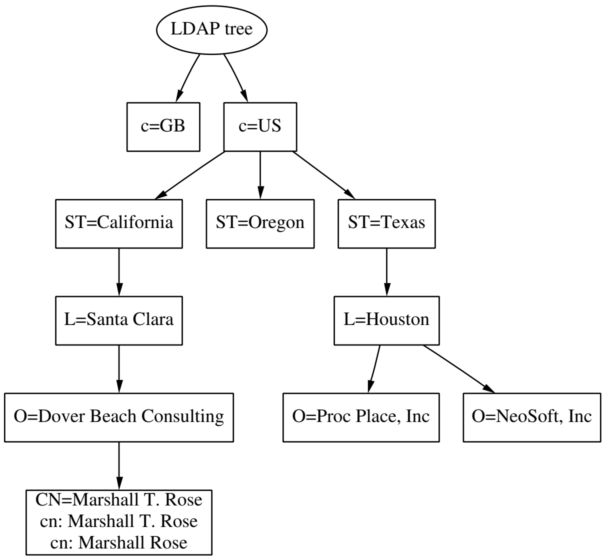

Figure 1: An example LDAP directory tree.

2.2. Open Source and Commercial LDAP Servers

The University of Michigan LDAP server is the original LDAP server and C-client and is the basis of many commercial LDAP servers (including Netscape and Novell) as well as an open source version known as OpenLDAP. The final version of the University of Michigan server, 3.3, which implements LDAP version 2, was released in 1996. It is no longer being actively maintained.

For more information on OpenLDAP, visit *http://www.openldap.org*. The site includes substantial documentation in the form of UNIX manual pages, a guide in Adobe Portable Document Format (pdf), and Internet RFCs related to LDAP services.

To find out more about the commercial iPlanet directory server, please visit *http://www.iplanet.com*. There are many commercial directory servers on the market.

3. LDAP Server Components

The LDAP server includes several programs, (names vary by product) including:

- **slapd** - the LDAP server.

- **slurpd** - Replication daemon. Slurpd synchronizes to another LDAP server by reading a replication log, written by **slapd** when updates are performed, using the LDAP protocol to update the remote server.
- **ldif2ldbm** - program used to convert a database in text LDAP Directory Interchange Format (LDIF) to an LDBM database suitable for use by **slapd**.
- **ldbmcat** - program to convert a **slapd** LDBM database to the text LDAP Directory Interchange Format (LDIF).

Along with a client API library, command-line tools to manipulate and search the database are also provided: **ldapadd**, **ldapmodify**, **ldapdelete,** and **ldapsearch**.

It is difficult to manipulate a directory using these tools, especially ldapmodify.

4. LDAP and Tcl

A Tcl interface to LDAP opens LDAP to scripting, making available Tcl's text processing, data and network access capabilities, the Tk toolkit, and all the extensions. Tcl's support for Unicode and the use of Unicode in LDAPv3 servers make a nice match.

5. LDAPtcl

Because our shop used Tcl extensively and we had a potential LDAP application (see case study below), we decided to explore creating a Tcl interface to LDAP.

5.1. LDAPtcl Implementation Decisions

Our first impulse was to try to write an LDAP client natively in Tcl. A native Tcl implementation would avoid the complexity and difficulties of writing, distributing, and maintaining a C extension. But after inspecting the Basic Encoding Rules (BER) of the Abstract Syntax Notation One (ASN.1), which LDAP uses, and the thousands of lines of C code that implements them, we began to have doubts. The University of Michigan server was already at 75,000 lines of C code, with more than 17,000 lines just for the BER and LDAP client libraries. Writing a native Tcl version would require much of this code to be translated, and debugged... a daunting task. Further, since LDAP continues to evolve, the Tcl implementation would need to be continually updated. We decided that it would be easier to write an interface to the C API and let others confront the interoperability issues.

The next question was whether to use the blocking (synchronous) or the nonblocking (asynchronous) interface. With the blocking interface, you make a request and the library returns the results, while with the nonblocking interface, you make a request and then query a library routine repeatedly for results. The advantage of the nonblocking interface is that LDAP responses could be received through the Tcl event handler, meaning that other Tcl events could be processed while waiting on, for example, the results of an LDAP search, whereas with the blocking interface, the Tcl event loop would be "frozen" while the LDAP library routines were waiting on data or transaction results. The advantage of the blocking interface is simplicity - using the blocking interface would not require the machinations of creating and debugging a Tcl event handler. For our original implementation, we chose to use the blocking interface.

6. Using LDAPtcl

6.1. Connecting To An LDAP Server

First, use `"package require Ldaptcl"` to enable the LDAPtcl package.

To create an LDAP interface entity, we use the **ldap** command.

```
ldap open foo foo.bar.com
```

This opens a connection to an LDAP server on *foo.bar.com*, and makes a new Tcl command, **foo**, which we will use to manipulate the interface and make queries to the remote LDAP server.

```
ldap init foo foo.bar.com
```

As with open, **foo** is created, but for *init*, opening the connection is deferred until we actually try to do something.

6.2. Binding a Connection

When a connection is initially made to an LDAP server, you will be known only as an anonymous user, with whatever access that user is allowed (probably read-only or no access at all). An LDAP *bind* operation must be performed in order to indentify yourself to the server. Your identity will be as a DN, i.e. an entry within the tree that represents you. Servers can also allow limited anonymous read access without requiring binding.

Both simple authentication and Kerberos authentication are available in LDAP version 2. LDAP version 3 supports many new SSL-style authentication and encryption systems.

To use LDAP version 2 and still have reasonable security in a networked, Internet/intranet environment, secure shell can be used to setup secure, encrypted connections between IP ports on client machines and on the LDAP server, and between the LDAP server and any replica or slave servers that might be used.

To perform the LDAP bind operation to authenticate a distinguished name that we'll be known as:

```
foo bind simple dn password
```

```
foo bind kerberos_ldap
```

```
foo bind kerberos_dsa
```

```
foo bind kerberos_both
```

The bind operation either returns nothing (success), or raises a Tcl error with appropriate error text upon failure. SASL, TLS, and *ldaps://* URLs are not currently supported.

Here is an example:

```
foo bind simple "cn=Manager, o=NeoSoft Inc, c=us" "secret"
```

To unbind an LDAP connection previously bound with "bind":

```
foo unbind
```

The ability of the library to call back to the client, enabling re-binding while following referrals, is not currently supported. (Experimental code has been written but is unproven.)

6.3. Modifying the Directory

In the command descriptions below, `attributePairList` is used to represent a list of attributes and their values. For example, the `attributePairList` for the example above would be:

```
set attributePairList [list \
 objectClass inetorgperson \
 cn {{Marshall T. Rose}} \
 sn {Rose} \
 givenname {Marshall} \
 sn {Rose} \
 st {California} \
 l {{Santa Clara}} \
 mail {{mtrose@dbc.com} {marshall.t.rose@dbc.com}} \
 postalAddress {{123 Beach Street \
 $ Santa Clara, CA 94123}} \
 telephoneNumber {{213-123-4567}} \
 street {{123 Beach Street}}]
```

Whenever we refer to key-value pairs, remember that the *value* part is always a *list* of values, since LDAP entry attributes can be multi-valued.

6.3.1. Adding New Objects

```
foo add dn attributePairList
```

This creates a new distinguished name and defines one or more attributes. *attributePairList* is a list of key-value pairs, the same as would be returned by `array get` if an array had been set up containing the key-value pairs. Note that, right now, the sort of lowest-level pair of the DN must also appear in the attributePairList, as in:

```
foo add "cn=karl, ou=People, o=NeoSoft Inc, c=US" {cn karl ...}
```

6.3.2. Deleting Objects With LDAPtcl

To delete an object in the LDAP database, use

```
foo delete dn
```

To rename an object to another relative distinguished name, use

```
foo rename_rdn dn rdn
```

6.3.3. Adding, Deleting, and Replacing Object Attributes

```
foo add_attributes dn attributePairList
```

This adds key-value pairs to an existing DN. If an attribute being added already exists, the new value will be appended to the list. It is an error if the value is already present for that attribute.

```
foo replace_attributes dn attributePairList
```

This replaces specified key-value pairs in an existing DN. Only the specified attributes are modified, and all previous values of the modified attributes are discarded. An LDAPv3 server will accept a null value list as meaning the attribute should be deleted, but an LDAPv2 server will consider this an error. The `replace_attributes` can be thought of as the "make it look like this" command.

```
foo delete_attributes dn attributePairList
```

This deletes attribute values from the dn. If an attribute `foo` has two values `bar` and `snap` and you perform `foo delete_attributes dn {foo bar}`, `foo` will still have an attribute value of `snap`.

If you provide an empty list for the value part of the key-value pair, the entire attribute will be deleted. To reiterate, if you provide a non-empty string for the value part, only that value will be removed from the value list.

The latest version of the interface allows these add, replace, and delete operations to be combined into a single Tcl command (and a single C API call).

6.4. Performing LDAP Searches

The Tcl interface to searching uses a *control array*, some elements of which are mandatory. It also takes a *destination array*, into which the specified attributes (or all attributes of retrieved entries if no specific attributes are specified) and values are stored.

```
foo search controlArray destArray code
```

The "code" part is executed repeatedly, once for each DN matching the search criteria.

Using data in the control array, a search is performed of the LDAP server opened when **foo** was created. Possible elements of the control array:

- **controlArray(base)** is the DN of portion of the directory to be searched. (required)
- **controlArray(filter)** contains the search criteria. (optional) For example, "(mail=*kunkee*)". Filters are always surrounded by parenthesis.
- **controlArray(scope)** sets the scope of the search. It should be "`base`" to search the object itself, "`one`" to search the object's immediate children, or "`subtree`" to search the object and all of its descendants. If not specified, scope defaults to "`subtree`".
- **controlArray(deref)** specifies how alias dereferencing is done. Deref should be one of "`never`", "`always`", "`search`" or "`find`" to specify that aliases are never dereferenced, always dereferenced, dereferenced when searching, or dereferenced only when locating the base object for the search. If not specified, deref defaults to "`never`".
- **controlArray(attributes)** is a list of attributes to be fetched. If not specified, all attributes are fetched. The RFCs state that operational attributes may not be returned unless they are explicitly requested. Kurt Zeilenga, the leader of the OpenLDAP project, has proposed that requesting the attribute name "+" represent a request for all operational attributes, and "*" requests all user-defined attributes. A request for "+ *" to an OpenLDAP version 2 server will return all attributes. Note that access limits may prevent attributes from being returned.

For each matching record, **destArray** is populated with none, some or all attribute-value pairs. **destArray(dn)** is set to the object's distinguished name.

Note that there are some additional parameters that can be set, such as how long the synchronous version of the routines should wait before timing out. Please examine the documentation in the LDAPtcl package for more information.

6.5. Caching Search Results

The University of Michigan and OpenLDAP libraries offer the client application fairly fine-grained control of caching of search results, potentially offering significant performance improvement and reduced network traffic.

By default, the cache is disabled (and is not universally supported by all SDKs).

To manage caching of data received from an LDAP connection, LDAPtcl provides:

```
foo cache enable timeout maxmem
foo cache disable
foo cache destroy
foo cache flush
foo cache uncache dn
foo cache no_errors
foo cache size_errors
foo cache all_errors
```

If an operation is performed to modify or delete a DN, an uncache request is automatically performed.

7. Version X - Nonblocking LDAPtcl

Billed as "The Tcl/LDAP Interface I wanted to write in the first place, but didn't know how", we finally did write a version that is fully asynchronous. It supports getting bind results, add results, delete results, search results, etc, as full-fledged events through the Tcl event loop, so Tcl is free to do other work while waiting for LDAP results. So for example while search results are coming back from an LDAP server, *after* events are firing, data is being received and processed from sockets and pipes, etc, and even other LDAP operations can be in progress simultaneously.

You use the command **xldap** to open a connection and create an object, as with **ldap**.

For searches, the control array is the same, but you no longer specify the result array or the code to execute over the contents.

Now the search is asynchronous, and when a result comes back, the event handler invokes a routine named **ldap_callback** in the interpreter that had the **foo** command created in it. (This is a little foul... you really should be able to specify the callback handler explicitly.)

xLDAPtcl invokes **ldap_callback** with parameters. The first argument is the command name you created when you created an LDAP connection. Next is the message ID of the search, which was returned by the `foo search` command. Next appears the word **search**, followed by a list containing the key-value attribute pairs returned by the search.

When the search is complete, this call is performed (for message "2"):

ldap_callback foo 2 ok

Other ldap commands such as add, delete, etc, also eval **ldap_callback**, as above, or call it with an LDAP error message in place of the "ok".

7.1. Abandoning A Command

The **abandon** command is used to abandon or cancel an operation in progress. The *msgid* is the message ID of an outstanding LDAP operation.

```
set msgid [foo search controlArray]
foo abandon $msgid
```

7.2. Implementation

xLDAPtcl has an *LDAP master structure* that's passed around using the *ClientData* mechanism Tcl provides for this sort of purpose. This structure contains data important to the LDAP Tcl interface. Most important of this is the *connection table*. The connection table has an entry in it corresponding to every defined LDAP connection.

For each connection, there is a *pending message hashtable* that contains entries for each action that is in progress (i.e. requested of the LDAP server where a reply hasn't been received yet).

LDAP results are processed by an *LDAP event handler* that operates as a full partner in the Tcl event handling system.*

The LDAP event handler uses the connection table to iterate over all of the open LDAP connections, calling the C *ldap_result* routine for each one in turn. When a result is received, the message ID is looked up in the connection's pending message hashtable. This entry tells us which interpreter the request was issued from, and some other stuff, and allows us to generate a callback for the received reply with enough info that the Tcl program can figure out which request is being replied to.

*This was tricky, but hey, that's half of what it's all about.

7.3. Status

xLDAPtcl has been tested, and it works. But it has not yet been used in a production environment, and should be considered beta-level code. In particular, there may be memory leaks.

7.4. Future Direction

The use of the global proc **ldap_callback** will probably be replaced by a settable callback.

Right now this interface sits entirely outside of LDAPtcl. That was great for creating it and debugging, and is somewhat nice for people who are using the old interface, since nothing changed. But it would be nice if we combined this command with the old style in a compatible way. Another possibility is to ditch the old-style command and front-end the new-style command with a proc that works with the callback to emulate the old-style interface.

8. Case Study: ISP Provisioning System

NeoSoft, Inc., a regional ISP located in Houston, Texas, had grown to tens of thousands of user accounts. New accounts were being created by an hourly "push" process that created UNIX accounts, updated RADIUS server files, edited webserver configurations, etc., in response to information that sales people entered through a Tcl/Tk-based interface.

The hourly push process, though effective, frustrated new users by delaying the creation of their accounts for up to an hour after they signed up. Finally, it was decided that the company's development of web-based online signup required new accounts to be created very rapidly. The solution also needed to be able to scale to handle many more users, access servers and remote locations.

Dialup and ISDN logins were handled at that time by a mix of Cisco AS-5200 and Ascend Max access servers. These had been successfully authenticating via multiple redundant RADIUS servers for some time.

We decided to alter the RADIUS server software to have it make requests over the network to our master database, Oracle running on a Sun enterprise server, behind a firewall. Since we didn't want the Oracle server, and the path to it, to be a single point of failure that would cause all authentication requests to fail, we decided that we would cache the data in some sort of intermediate database, possibly something based on Berkeley DB.

At this time, LDAP support was not available for either the Cisco or Ascend operating systems, but Cisco had announced their intention to add it.

We decided to Tcl-enable the RADIUS server, essentially by inserting a call to **Tcl_Eval** at the correct point, such that Tcl code would be responsible for delivering the user's RADIUS record to the server. This part was simple and never gave us any trouble.

We would then use the LDAPtcl interface to make LDAP client queries to LDAP servers, which would serve both as the intermediate database and the protocol for handling the requests. Redundant LDAP servers would be used, synchronized by slurpd.

A diagram of the configuration appears in Figure 2.

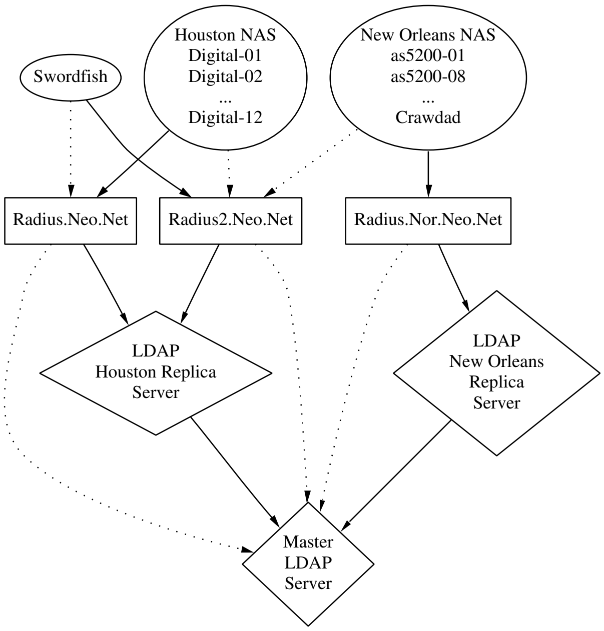

Figure 2: LDAP-enabled Access Servers. Solid lines indicate primary paths while dotted lines indicate backup paths.

Later, when the access servers directly supported LDAP, the configuration would be simplified by the removal of the RADIUS servers. The simplified configuration appears in Figure 3.

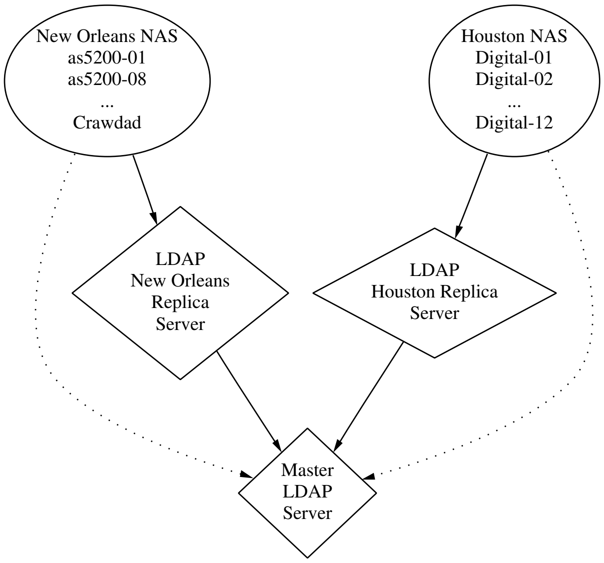

Figure 3: Simplified LDAP Configuration.

Note that this has not been implemented due to new management priorities after the sale of the company in mid-1999.

8.1. Lessons Learned

Bugs in the LDAP server and LDAP client implementations of that time (circa 1997-1998) resulted in occasional segmentation violations and mystery hangs which were very difficult to track down. A RADIUS server hanging on an LDAP client request meant no further RADIUS requests would be processed until the RADIUS server was restarted; the LDAP server would still be able to process requests despite having a hung client. Setting a client request timeout of a few seconds rather than the default of infinity helped a great deal. The packets weren't getting lost, though... bugs in the LDAP implementation caused the client and server to get out of sync. (Note that we added the ability to control the timeout from the Tcl interface.)

Over time we added the usual little bits to detect problems and restart the programs in response. Subsequent releases of the LDAP software and LDAPtcl have become much more robust and reliable. But during the early deployment phase, these problems, plus an unrelated Cisco IOS RADIUS failover bug and access server configuration errors had a negative impact on our quality of service.

There were enough problems with LDAP and they were tricky enough that it would almost certainly have been cheaper to follow a pure Tcl approach. The development of our LDAP approach cost more and took longer than we planned. A pure Tcl approach would not have been able to leverage the increasingly popular LDAP standard, however.

We ultimately did make LDAP go, and the LDAP-enabled RADIUS application is reliable, in production, running unattended 24/7. At our last check while writing this paper, the LDAPtcl-based RADIUS server had been up for over a month, performing thousands of searches per day.

9. Case Study: EcuNet.org

EcuNet has existed for 20 years, and wanted to modernize their software and hardware. Their old system had of the ability to send "personal" notes to individuals and to participate in meetings. The code to implement all of this was proprietary, and contained some interesting features. Personal and meeting notes could be downloaded and processed off-line (using a home grown protocol). Meetings have an elaborate set of rules for determining who could access them, as well as what they could do with them (ie. read-only, write-only, etc.) Participants had the ability to create private meetings for whatever purpose they might choose. It ran on Compaq hardware under SCO UNIX. EcuNet wanted to upgrade their system while at the same time leveraging the massive resources of open source tools.

The new system would have to support the old download programs for off-line processing and access control. In addition, users would be able to read their notes via a web interface, or could elect to have them forwarded to an email address. Optionally, that address could be in the EcuNet.org domain and we would provide the POP3 (and later IMAP) interfaces for them. Other than the elaborate permissions scheme (which includes inheritance), and the support for the old download protocols, this would be similar to other portals such as Yahoo. A final requirement was that the system be replicated -- they wanted to have the ability to place servers around the world where Internet service might be unreliable or infrequent. This meant the authentication and access control database would have to be distributed and in some cases allow for mobile users (eg. a user who goes on a mission to Africa for a year).

Mapping personal notes to email and meeting notes to USENET-style news articles was a no-brainer. We would support normal email and provide interfaces to the old download clients. Internet News (INN) would be used to handle the meetings. There were also a few mailing lists that needed to be supported, and we chose Mailman for that. The web server would be Apache with the NeoWebScript modules included, to allow us to embed Tcl code directly in HTML pages and generate some GIF images dynamically. Tcl would be the glue, and LDAP would provide the data.

LDAP seemed to be a fit for several reasons: It is designed for authentication, it already had replication, parts of the directory could easily be distributed, and its hierarchical (directory) data model would easily map newsgroup paths to corresponding Distinguished Names with additional information.

The hardware we selected was Alpha Server 1000s with 512MB RAM and lots of RAIDed disk (mostly because Digital was doing promotions at the time).

The primary access to the server for creating and managing accounts is via the web server. Along with authentication information for logging in (WWW, POP, IMAP), LDAP contains information on meetings aka newsgroups (subjects, and extended descriptions), lists of users invited to attend a meeting, mail forwarding information (for those who prefer to receive meeting notes via email), meeting access control, address books, and a number of user preference settings.

Another piece of information held in LDAP is .newsrc data, i.e. information on which notes/articles each user has read (or has been mailed to them). A background process, written in Tcl, maintains state information and functions as a cache of LDAP data for the web server.

A block diagram of the Ecunet service appears in Figure 4.

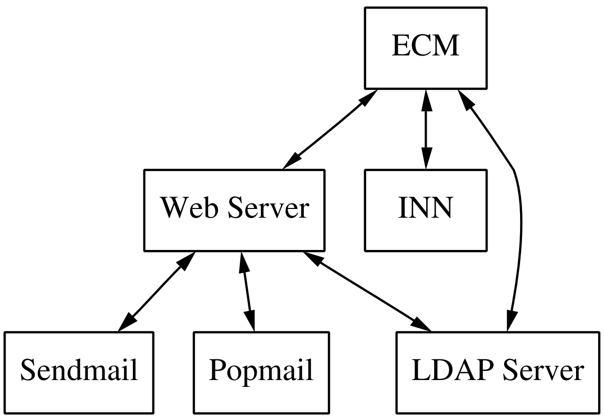

Figure 4: Ecunet Service Block Diagram

The .newsrc objects are not particularly write once, read many. They are more read once, write once (or many), which is not what LDAP is designed for. Thousands of write operations are performed on them daily.

9.1. Going Live

The new EcuNet went live in the Spring of 2000.

We soon discovered that a behavior of OpenLDAP version 1.2 servers, to create and destroy a thread for each request, caused significant leakage on our AlphaServer 1000s. This is improved with Tru64 v5.1a, and we look forward to measuring process growth with OpenLDAP v2.0, which uses a thread pool instead of creating and destroying a thread with each request. In addition, caching of a lot of data naturally grows the process to be very large. In a day of service, the LDAP server can easily exceed 300MB.

We addressed the server instability in several ways. Since the server was crashing due to a combination of high activity and the thread leak, we split the data into two separate LDAP servers. At least if the .newsrc database was busy, by isolating it into a separate server instance, we would not lose authentication capabilities retained by the primary database (it was once again write-once-read-many). Nevertheless, both servers still grew very large over time. The .newsrc database would still crash after 24+ hours.

Somewhere along the 2nd day, they would crash (after several days, for the primary server) as the thread library eventually runs out of the ability to create threads. We then instituted a policy of restarting the servers in the early hours of the morning, when the server was least active. We look forward to using OpenLDAP 2.0 in the production system, where this problem should go away due to the new thread pool management scheme.

Finally, we recently added another 512MB of RAM to the LDAP server machine, bringing it to a total of 1GB. This is enough to allow the server to cache the entire database into memory.

There was one bug in the LDAPtcl code which we also had to resolve early on. This bug fix was also turned back into the NeoSoft LDAPtcl-based RADIUS servers.

9.2. Reflections on Ecunet

Hindsight being 20-20, would we do this the same again? Assuming we had to write it again, there are several design decisions I would change.

LDAP vs. relational: We now use both, and that would have been a better approach from the beginning, especially for the write-many-read-many .newsrc objects.

There are certain reports that we do that are fairly painstaking in LDAP, which would be a breeze in SQL. Nevertheless, the future need to either distribute or replicate the database would be non-trivial in SQL, so the current design of the system won't be changing soon in this regard. (A the time of the project's inception, such an SQL server would probably have to have been commercial, a fact which is less true today) and exceeded our budget.

We choose to use a flat model with the user's UID in the DN instead of CN. We did this to avoid having to search the directory and rename every place the DN was used whenever a person changed their names. In retrospect, this was "relational" thinking, and one of our major costs is looking up the CN for a given UID (and what the Tcl servers cache). However, it does accomplish the desired goal.

As to overall performance, even though most of the system is written in Tcl, the system performs quite well. Time and again, performance problems have turned out to be easily corrected by cleaning up poorly written code. In most cases, this involved preventing the same search requests from being performed repeatedly (sometimes within the same web hit) by caching data. Not a novel concept, but less experienced programmers would do things that would cause lower layers to repeat the same searches over, for example.

9.3. Ecunet Conclusions

When we went live with Ecunet, we had some concerns about LDAP being able to hold up under the load. We selected OpenLDAP because it was open source, and felt that any problems that would arise could be dealt with quickly, even by ourselves if necessary. LDAPtcl has demonstrated extreme stability over the years, being a component in various systems that perform thousands to tens of thousands of transactions in single long-lived processes.

An [array get] is easily written into an LDAP object, and LDAPtcl searches easily return objects into a Tcl array. With LDAPv3 protocols, it is an easy matter to take the search results in an array, modify them, and set the object to the new values in the array. LDAP databases are flexible, and capable of storing any data from a Tcl application.

10. Current Status of LDAPtcl

LDAPtcl has mainly been used with the OpenLDAP client library. It also has been built successfully against the iPlanet server, although this has not been done recently.

LDAPtcl now interfaces with OpenLDAP 1.x (LDAPv2) and 2.0 (LDAPv3).

OpenLDAP and LDAPtcl are now mature technologies, and are ready to be used for your directory server needs. xLDAPtcl is ready for experimental use and further development.

11. Getting The Code

LDAPtcl is currently available via the OpenLDAP project under the contrib/ldaptcl directory. It is a standalone Tcl extension package that requires OpenLDAP and Tcl 8 to be installed on the system. A GNU autoconf-based configure script will create a Makefile to compile and install the package as a Tcl package.

12. References

12.1. RFCs

RFC2251: Lightweight Directory Access Protocol (v3)

RFC2252: LDAPv3: Attribute Syntax Definitions

RFC2253: LDAPv3: UTF-8 String Representation of Distinguished Names

RFC2254: The String Representation of LDAP Search Filters

RFC2255: The LDAP URL Format

RFC2256: A Summary of the X.500(96) User Schema for use with LDAPv3

RFC2829: Authentication Methods for LDAP

RFC2830: LDAPv3: Extension for Transport Layer Security

Copies of these documents can be obtained from *http://www.rfc-editor.org/*

The IETF (*http://www.ietf.org/*) is currently working on revising LDAPv3 specifications for publication as a "Draft Standard."

LDAP version 2 (LDAPv2) is an Internet "Draft Standard" and is documented by the following RFCs:

RFC1777: Lightweight Directory Access Protocol

RFC1778: The String Representation of Standard Attribute Syntaxes

RFC1779: A String Representation of Distinguished Names

RFC1960: A String Representation of LDAP Search Filters

12.2. Books

Implementing LDAP, by Mark Wilcox Wrox Press, ISBN 1-861002-21-1 LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol, by Howes and Smith, Macmillian Technical Publishing, ISBN 1-57870-000-0

Understanding and Deploying LDAP Directory Servers, by Howes, Smith and Good, Macmillian Technical Publishing, ISBN 1-57870-070-1

Understanding LDAP, IBM Redbook *http://www.redbooks.ibm.com/abstracts/sg244986.html*

Understanding X.500, by D. W. Chadwick, ISBN 185-0322-813. An online version is available from *http://www.salford.ac.uk/its024/Version.Web/Contents.htm*

Understanding LDAP, IBM Redbook An online version is available at *http://www.redbooks.ibm.com/abstracts/sg245110.html*