USENIX Association

Proceedings of the 14th Systems Administration Conference (LISA 2000)

New Orleans, Louisiana, USA
December 3–8, 2000

Network Information Management and Distribution in a Heterogeneous and Decentralized Enterprise Environment

Alexander Kent & James Clifford – Los Alamos National Laboratory

ABSTRACT

To promote enterprise-wide information and resource sharing, we have implemented a network information management and distribution system that gives subscribing systems real-time access to relevant information changes. Participating systems need little more than a small, single application to receive the updates. User interaction and data administration require only a web browser. The resulting system is timely, reliable, secure, easy to support and maintain, and extensible.

Introduction

In Los Alamos National Laboratory's distributed and heterogeneous computing environment of more than 12,000 users and more than 20,000 computers, the task of maintaining accurate network services information is a complex undertaking. Information such as network accounts, e-mail addresses, login names, aliases, home-page pointers, privileges, permissions, and so on, is constantly changing as people come and go or change job assignments. Matters are further complicated when the input data comes from different sources and must then be distributed across multiple server systems. To simplify the system administrator's job, we designed a system to make it easier for network services to get the current and accurate information they need to do their job. We call it A Real Time Information Management and Update System (ARTIMUS).

If you have attended past USENIX LISA conferences or read the proceedings, the problem and solution may sound vaguely familiar. The general requirements for an account management system have been nearly the same for over ten years, including: [2]

- Vendor and service independence
- Data flexibility
- Minimal changes to existing software
- Automated account installation
- User access to data

This paper focuses on the unique aspects of the Los Alamos National Laboratory (LANL) approach. Specifically, these areas include the database design, manipulation of data within the central database, propagation of that data to end-hosts and services, and security. Recent systems differ on whether they use a "big hammer" relational database or not. ARTIMUS, [1], [4], and [11] do; [2], [7], [9], and [10] do not.

Motivation

Our organization, LANL's Network Engineering Group, is the Lab's Internet service provider. We provide the usual services: LAN and dialup connectivity to the Internet, DNS, white pages directory information, authentication, e-mail accounts, web page publishing, and so on. Other LANL organizations offer compute, storage, print, personnel, training, library, and many more network services to internal and remote customers. In addition, organizations throughout the Laboratory run their own network servers to satisfy local needs. Each organization, and even individuals, choose the type and configuration of computing system(s) that best meets their needs. While this looks like a recipe for chaos, it also presents opportunities.

The group needed a sane way to manage people, accounts, and other network information for our ISP business. With the right design, we saw that we had a chance to help the other network service providers too. Subscribers could participate in the same centralized account and information management system that we use on their own servers. These are the requirements we felt we had to meet to have a viable system:

- Each user should be able to make changes to his or her own network information. A third party, like a system administrator, should not be needed to make the change.
- The end user interface should be intuitive and consistent. Managing your network profile will not be a frequent or familiar activity. The goal here is to allow changes to be entered quickly and accurately without a call to the help desk.
- Existing subscriber system software should not require modification. Programs, either shrink wrap or open software, will still run using existing data formats and access methods.
- Additional subscriber systems should be easy to integrate. We will not own or manage many of these clients. Changes to subscriber systems must be few and minor in nature.
- Subscriber systems should not be affected by changes to our data structures and business rules.

- Subscriber systems should not be affected by failures in the information management system.
- Subscriber systems should be notified of data changes in near real-time so users can view and test changes immediately after making them.
- Subscriber systems should get only accurate, consistent, and current data.
- System administrators should retain control over their systems. For example, they should be able to control who gets access and privileges on their systems. There should also be provisions for system administrators to make local changes outside of ARTIMUS.

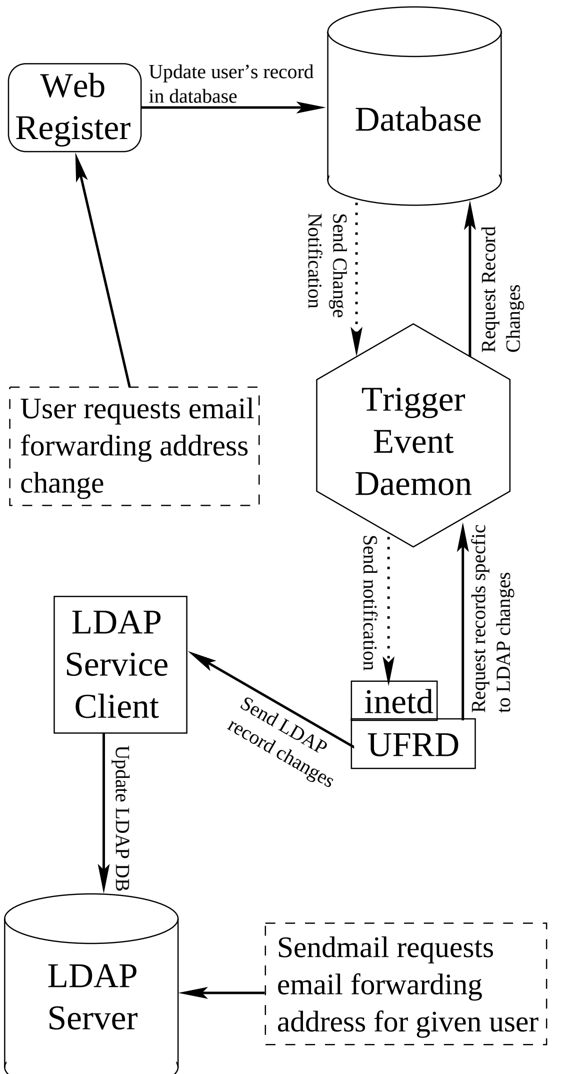


Figure 1: Example data flow for a user setting a new e-mail forwarding address for a given name, and sendmail requesting the forwarding address.

- System administrators should see their workload drop after participating in ARTIMUS.
- ARTIMUS should be extensible. The difficulty in adding new services and operating systems should be in step with their complexity.
- ARTIMUS should have high availability.
- ARTIMUS should import corporate data to eliminate redundant data entry and promote consistency. For example, employee data from HR could be used to build organizational e-mail lists.
- ARTIMUS should be secure. Information should be readable and modifiable only by authorized persons.
- ARTIMUS, not subscriber systems, should implement site policies and business rules.

Overview and Example

Here is a simplified example to demonstrate ARTIMUS' data flow, which can be seen in Figure 1. Suppose Amy wants mail that is sent to the institutional address help@lanl.gov to be delivered to her departmental server account amy@cic-mail.lanl.gov. Here is how it is done: Amy uses her web browser to go to the Network Registry service and logs in to identify herself. She creates help as a new name which the Network Registry inserts into the database assigning ownership to Amy. Then she sets the forwarding address for help and the web server completes the task by updating the database. The change to the table containing names triggers a database procedure that makes a copy of the updated record and notifies the Trigger Event Daemon (TED) process (see the section "End Service Propagation"). TED notifies the subscriber services affected by the forwarding address change so those systems can update their local copies of e-mail forwarding information.

System Design

ARTIMUS is a three-tier design of web-based user interface to central database to end-system propagation. The system is functionally divided into these three distinct segments to provide maximum security, flexibility and simplicity.

Web Interface

Users view and change information with ARTIMUS' web interface called the Network Registry. All user modifiable information is updated in this manner, traditional applications are not used (e.g., chfn). This GUI interface is designed to be friendlier than native system interfaces. In addition, the SSL web interface encrypts the data between the desktop and the web server thus protecting the registration information from capture and modification.

User access to ARTIMUS data is controlled through the web interface. The user must first authenticate to the Network Registry with an employee identifier and password. The web interface then enforces access controls that restrict what the user can change within the database. Normally users may only change information they "own." However, system administrators may be granted access to change other users' information. We considered pushing the authentication and access control into the database itself but Sybase, which we currently use, does not support any appropriate external authentication systems like RADIUS or Kerberos.

In the previous example of Amy changing the forwarding address of help@lanl.gov, Amy goes to the URL for the Network Registry and authenticates with her employee identifier and password. Selecting the name help she then enters the forwarding address as seen in Figure 2. The Network Registry then does a SMTP VRFY on the given forwarding address and submits an update to the database name table if the address is valid.

In addition to e-mail forwarding, the Network Registry is used to create and remove user accounts on UNIX and Microsoft-based systems, change passwords, and manipulate several other user network attributes. Additional features and systems are constantly being added to the Network Registry.

The web interface is built upon whiz [8], a Python-based sequential CGI forms tool that provides a simple, stateful, and authenticated method of managing the registry segments and adding additional features. Communication to the database is handled through a locally developed system called sqld which wraps database traffic with SSL-based encryption between the Network Registry and the database. Other, more standard database access libraries could be substituted for sqld.

The Database

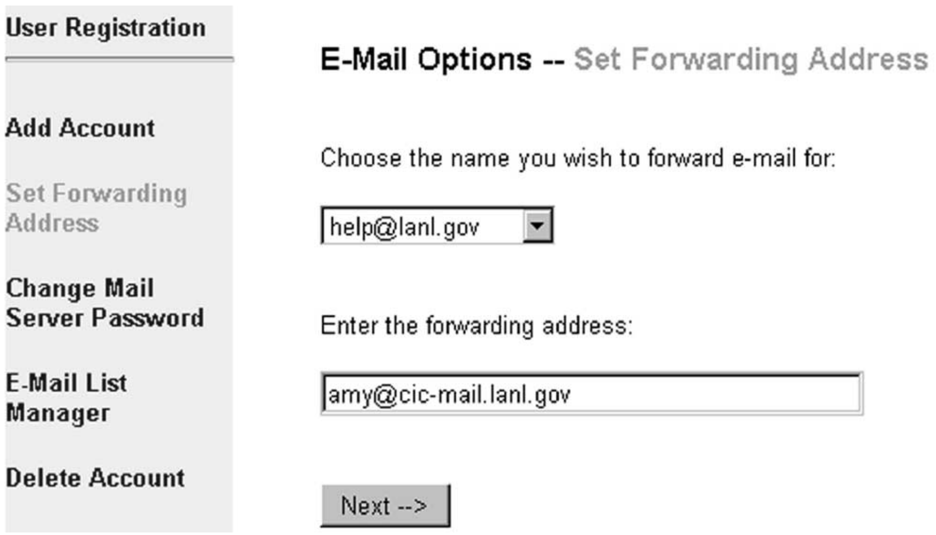
The authoritative copy of nearly all network service information is kept in a relational database. The Network Registry and various corporate repositories, like human resources' employee data, are ARTIMUS' information sources. The database enforces syntax, consistency, various business rules, and serves as a central information clearinghouse.

If the information is accurate in the database, it will be accurate on the servers. And if it is wrong in the database, it will be wrong everywhere. Because data integrity is so important, we use the database's features such as syntax checking on table fields, inter-table consistency enforcement, and ''triggers'' that call stored procedures to validate record changes. Both Sybase and Oracle products support these features. Previously, we tried maintaining consistency by doing the checking in the web server and other applications that modified the database. It worked but changes to table structures and field definitions were more difficult because several programmers had to modify code simultaneously. This explains our move to a ''big hammer'' relational database.

The database tables are divided into three primary categories: one set for *person* data, one for *name* data, and one for *auth*entication/authorization data. *Person* data is needed for ownership; i.e., some *person* owns each table entry in the database. *Person* data is also needed for white-pages publishing. A *name* has an owner and may have attributes such as as a forwarding address, a URL, a UNIX UID, sharing group members, and mail list members. *Auth*entication and authorization data include a *person's* passwords, authentication tokens, accounts, and permissions or authorizations.

A portion of the network information database is shown in Figure 3. Designing the database tables was a non-trivial, iterative process. Subscriber requirements for data, syntax, and business rules were gathered and database design principles were applied [6]. The resulting design was reviewed by peers and customers. Then the process was repeated as more services were added and mistakes were uncovered.

Here is how the syntax, integrity, and business rules are applied in Amy's e-mail forwarding address example. Before help is added to the name table, the



E-Mail Options -- Set Forwarding Address

Choose the name you wish to forward e-mail for:

help@lanl.gov

Enter the forwarding address:

amy@cic-mail.lanl.gov

Figure 2: Network Registry page for adding/changing e-mail forwarding addresses.

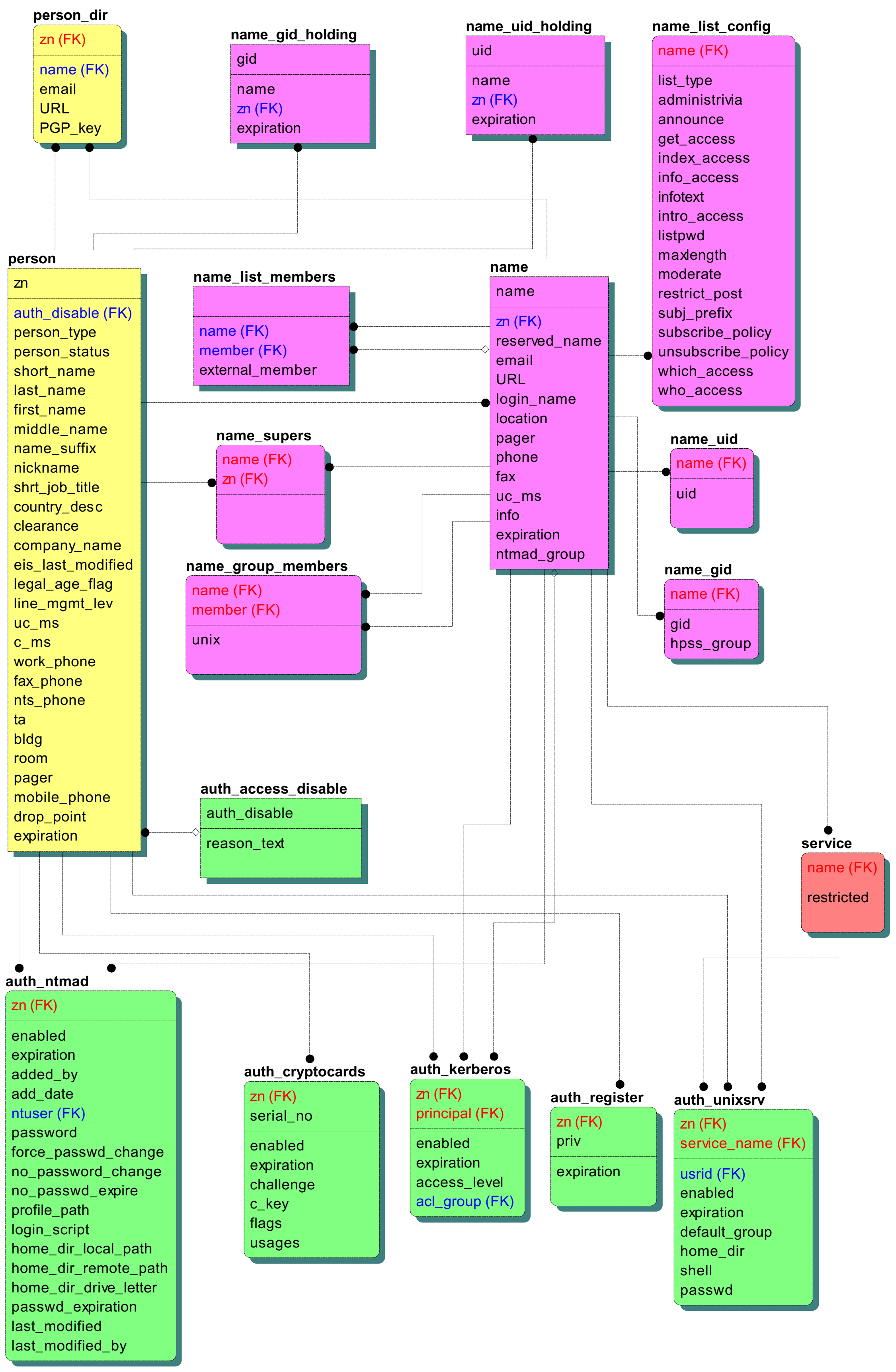


Figure 3: Major tables and dependencies within the database.

length and characters are checked. Names with @'s or spaces are rejected.[1] Also, names that already exist are rejected to enforce uniqueness.

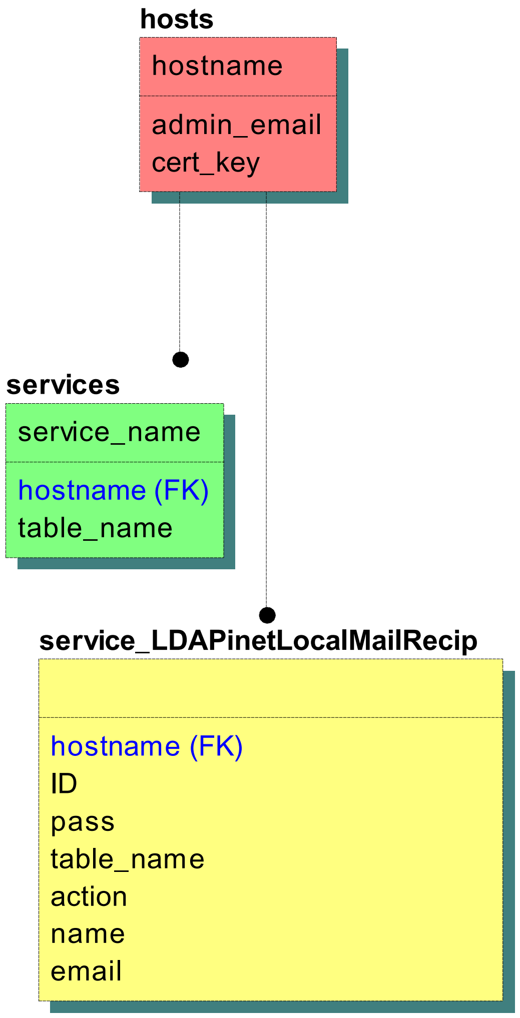


Figure 4: TED support tables within the database.

Similarly, the e-mail forwarding address, amy@cic-mail.lanl.gov, is checked before it is added to the database. This time an @ is required.

The effects of applying the concepts of business rules and data integrity are more striking in the example of an employee leaving. Our first rule is to wait for seven days after someone's personnel record disappears before deleting any data. Data entry mistakes happen and resurrections are simple.[2] When a personnel record disappears, that individual's expiration date is set to today plus seven days in the `person` table. Later, a cron job initiates deleting expired people. Before a person is deleted, the database deletes all of his or her names. The SQL command `delete from name where zn='Amy'`[3] deletes all of Amy's names. But before a name can be deleted, the database also cleans up anything that depends on that name. For example, when `help` is deleted, e-mail lists are cleaned up with `delete from name_list_members where name='help'`. This cascaded cleaning keeps the database internally consistent and, even better, reduces bounced e-mail to list owners.

End Service Propagation

Propagating information from the database to the subscriber systems is a critical component of ARTIMUS. The in-house written subsystem that performs this function is affectionately called *TED* and *UFRD*.

The Trigger Event Daemon (TED) feeds database changes to the Update Fields Real-time Daemons (UFRD) running on subscriber systems. UFRD passes the changes to local programs that then process the updates. See Figure 1.

The TED-UFRD subsystem is somewhat similar to the Project Athena Zephyr Notification System [3] in terms of providing immediate, reliable, and high fan-out information propagation. However, the TED-UFRD design is simpler and specific to transmitting information from a central source, the database, to end systems, the subscribers. Data transmission security and subscriber system data access control are built in.

The TED-UFRD system disseminates information changes in near real-time from database tables to a list of subscriber hosts affected by the changes. Instead of requiring services to wait for cron jobs to run or other polling mechanisms, the end services are immediately notified of information updates. Oracle has a similar notification facility but requires Oracle software on each subscriber system [5].

Currently LANL updates the following services through TED-UFRD with an average of 1500 information events per day:

- RADIUS dial-in passwords for the institutional modem pool (1150+ users)
- CRYTPOCard one-time password token card accounts (7200+ users)
- Microsoft master accounts domain for the institution (4400+ users and global groups)
- LDAP white-pages system (19,000+ record objects)

Additional services will begin using TED and UFRD over the next few months, including UNIX account and group management, Kerberos, Entrust, and DCE/DFS systems.

Trigger Event Daemon

Moving data from the database system to TED starts with triggers on tables containing information needed by the subscriber services. In the e-mail forwarding address example, the trigger is on the `name` table. A trigger fires (i.e., calls a stored procedure in

[1]Names cannot have a @ because an implicit @lanl.gov is appended when associated with a forwarding address.

[2]While not actually removing any information, all authentication and authorization (UNIX, NT, etc.) accounts for the person are disabled immediately.

[3]zn is a unique person identifier, usually an employee number. *Amy* is used here for simplicity.

the database) when a table is changed. The database allows one trigger per action (add, update, and delete) per table so TED triggers must coexist with a table's integrity triggers.

To remember table changes, TED triggers create new records in TED service tables with the type of modification (add, delete, delete-update, or add-update), the name of the table being modified, a

```
create trigger name_trg
    on name for insert
as
-- Begin TED update
    declare @tbl_name           varchar(40)
           ,@name               typ_name
           ,@zn                 typ_zn
           ,@reserved_name      typ_flag
           ,@email              typ_email
           ,@URL                varchar(255)
           ,@login_name         typ_flag
           ,@location           varchar(16)
           ,@pager              typ_pager
           ,@phone              typ_phone
           ,@fax                typ_fax
           ,@uc_ms              varchar(4)
           ,@info               varchar(255)
           ,@expiration         datetime
           ,@ntmad_group        typ_flag
    select @tlb_name = "lan1..name"
    declare ted_crsr cursor
        for select * from inserted
    open ted_crsr
    ted_loop:
        fetch ted_crsr into @name,@zn,@reserved_name,@email,@URL,
            @login_name,@location,@pager,@phone,@fax,@uc_ms,
            @info,@expiration,@ntmad_group
        if @@sqlstatus = 0 begin
            exec TED..send_service_LDAPmail_fwd(@tbl_name,
                "ADD",@name,@email)
            goto ted_loop
        end
    close ted_crsr
-- End TED update
```

Figure 5: Insert trigger attached to name table for updating e-mail addresses.

```
create proc send_service_LDAPmail_fwd
    @table             varchar(40)
   ,@action            typ_action
   ,@name              typ_name
   ,@fwd_addr          typ_email
as
    declare host_crsr cursor
        for select hostname from TED..services
            where service_name=@service_name
        open host_crsr
        host_loop:
            fetch host_crsr into @host
            if @@sqlstatus = 0 begin
                insert TED..service_unixsrv values(
                    @host,0,@table,@action,@name,@fwd_addr)
                goto host_loop
            end
       close host_crsr
       exec TED..notify("LDAPinetLocalMailRecip")
       return 0
```

Figure 6: Stored procedure called by triggers attached to name table to move records to TED tables and notify TED daemon.

subscriber host name, and the modified fields. A separate record is inserted into the service table for each subscriber host needing the information. The service table for e-mail forwarding and two other TED support tables are shown in Figure 4. The trigger sends a UDP packet to TED with the name of the service table where the inserts were placed. To simplify the coding, TED's triggers actually call a stored procedure to insert the records into TED service tables. The trigger for e-mail forwarding attached to the name table can be seen in Figure 5. The stored procedure called by the trigger is in Figure 6.

Upon receiving the UDP packet, TED reads the indicated service table from the database.[4] Then TED sends a UDP packet to each subscriber host it finds in the service table records. The UDP packet signals the host(s) to start UFRD. The host's UFRD makes a TCP

[4]TED will work with multiple database servers. As a security measure, it will only connect to database system(s) in its configuration file.

```
#!/usr/bin/perl
use Net::LDAPapi;
use Sys::Syslog;
# Update inetMailRecipient object.
# Changes read from stdin look like:
#   table|action|name|fwd_address
$local_domain = "lanl.gov"
$sep=chr(255);  # Separate fields with char 255
$ROOTDN = "cn=root, o=Los Alamos National Laboratory, c=US";
$ROOTPW = "oob";
$ldap_server = "ldap.lanl.gov";
&openlog("ufrd.LDAPmailRecipient",'pid','daemon');
&Sys::Syslog::setlogsock('unix');
$ld = new Net::LDAPapi($ldap_server);
if ($ld == -1){
   &syslog("err","Connection to $ldap_server failed");
   exit 1;
}
if ($ld->bind_s($ROOTDN,$ROOTPW) != LDAP_SUCCESS){
   &syslog("err","LDAP bind error: $ld->errstring");
   exit 1;
}
while (<>){
  chomp;
  ($table,$action,$name,$fwd_addr)=split($sep);
  $ENTRYDN="cn=$name, objectClass=inetLocalMailRecipient";
  # Treat update add/deletes same as regular add/deletes
  if ($action =~ "DELETE" || $action =~ "UPDEL"){
    if ($ld->delete_s($ENTRYDN) != LDAP_SUCCESS){
      &syslog("err","LDAP delete error: $ld->errstring");
      exit 1;
    }
  } elsif ($action =~ "ADD" || $action =~ "UPADD"){
    %ldapdata = ("cn" => $name,
                 "objectClass" => "inetLocalMailRecipient",
                 "mailLocalAddress" => "$name==[@]==$local_domain",
                 "mailRoutingAddress" => $fwd_addr);
                 "
    if ($ld->add_s($ENTRYDN,%ldapdata) != LDAP_SUCCESS){
      &syslog("err","LDAP add error: $ld->errstring");
      exit 1;
    }
  }
}
$ld->unbind;
exit 0;
```

Figure 7: UFRD client that creates/modifies/deletes LDAP inetLocalMailRecipient objects.

connection to TED and reads its information updates. When a change is successfully processed, UFRD sends TED an acknowledgement and the matching service table record for that host is removed. A TED service table record is kept until an acknowledgement is received so failures with UFRD can be retried.

To handle subscriber hosts that do not request their information updates, TED periodically reads all service tables' unprocessed data records. For each system with pending updates, TED sends another UDP notification since the subscriber host may have been down or unreachable. To summarize, information change notifications are sent to subscriber systems in near real-time and resent until successfully processed and acknowledged.

Because the information is stored in the database, state is maintained even if TED or the UFRD clients should terminate for unexpected reasons. TED will send an e-mail notification to the administrator listed in the TED's hosts table if a subscriber has unprocessed requests over two hours old.

Update Fields Real-time Daemon

A UDP packet from TED starts a UFRD on a subscriber host; on UNIX systems UFRD is started by inetd. The packet contains a port number to connect back to on the TED system. UFRD first reads its configuration file to find the encryption key to use with the TED system that sent the UDP packet. UFRD then makes a TCP connection to the TED server and sets up DES3 encryption using the key it found. UFRD receives the pending information updates and, based on the table name field, executes the appropriate command to process the data. The information update line(s) are passed to the command as standard input under UNIX and message passing under Windows NT. If the command exits successfully, UFRD indicates to TED to remove the corresponding record(s) from the TED service table.

The commands called by UFRD to update information may be written in the most convenient method: C, C++, Perl, Python, shell script, or some specialized interface for the service. The Perl code to process e-mail forwarding address changes is in Figure 7.

UFRD has been ported to several operating systems including Linux, Solaris, UNICOS, Irix, and Windows NT. Its simple functionality relying on running native commands to update local data allows quick porting to new platforms.

Future Plans

The group has several future improvements planned for the ARTIMUS system.

Extending ARTIMUS to other network information systems is foremost. Generating dynamic DNS updates is a goal for the next year. Our host information is already maintained through a web interface and kept in a Sybase database but does not use ARTIMUS. The difficult part will be reconciling the names' owners.

Offering the service to other Laboratory organizations has begun and will be expanded. Given the flexible and secure ARTIMUS framework, it is easy to provide central account management services to network servers throughout the Laboratory. Adding a Network Registry module, a few database tables, constraints, and triggers, as well as writing an update command for UFRD service to call, is not difficult.

The Network Registry continues to grow. Currently, we are redesigning much of the look and feel in attempt to make it easier and more intuitive for the end-users; user interface design is non-trivial. Various business and integrity checks continue to be added, both within the Network Registry and the database.

ARTIMUS needs additional rules to prevent inappropriate bulk changes from happening. Occasionally, we receive a large, but bogus, data feed from the corporate data warehouse that must be detected and rejected.

We would like to get real-time updates propagated from the Lab's corporate sources to pass along to end customers. Such real-time linkage would allow new employees to request accounts upon arrival (and have them created in real-time as well, thanks to ARTIMUS), plus automatically disable authentication and authorization accounts the moment they are terminated in the human resource system.

We would like to examine the use of individual user accounts within the database and direct authentication to the database. The tradeoffs of security benefits versus complexity may or may not make this practical. For various security issues we are considering moving from Sybase to Oracle and using Oracle's Kerberos and RADIUS authentication. Oracle's DBMS ALERT package may also be superior to the current UDP-based database notification mechanism for TED.

Additional issues, ideas, and plans will come up as we add new services and think up new methods to use the system.

Conclusions

The ARTIMUS design, including its web interface, relational database with built in data integrity, real-time update service, and data protection provides a solid base for eventually managing nearly all of the Laboratory's account and network information data. We are optimistic the implementation will meet or exceed the requirements listed in the Motivation section and discussed below. Ultimately, we will measure the system's success by the number of subscribers it supports.

User Interface Requirements

Users can manage their own network information through the Network Registry. Web based applications with authentication are commonplace at Los Alamos. Most customers will be able to easily maintain their own network profiles. Keeping the navigation simple, intuitive and consistent as features are added will be a challenge.

Subscriber System Requirements

Adding subscriber systems to ARTIMUS requires new entries in the TED tables, a UFRD daemon, a UFRD service application, a configuration file, and a shared DES key. The total installation time is usually under an hour.

Currently, 99 percent of the changes to the database are propagated to the subscriber systems in under two seconds via TED-UFRD. We expect the mean response time and standard deviation to both shrink when we retire some frequently run ad hoc database query applications.

System Administrator Requirements

There are a few system administrator features in ARTIMUS that are outside the main theme of this paper. We will briefly mention them here. Common system names like "root" are reserved, as are UNIX UIDs and GIDs under 1000. OS, third party, and local software that define users and groups can coexist with ARTIMUS. System administrators can define their subscriber systems as open or closed. Users can add accounts to open systems but only system administrators or their designees may add accounts to closed systems. Finally, we have defined organization administrators who can manage most information for anyone in an organization, and name administrators who, in addition to the owner, can change any of a name's attributes.

ARTIMUS Requirements

ARTIMUS already supports UFRD on a variety of operating systems (Figure 8) and services (Figure 9) but not all services are supported on all OSs. The first UFRD for a UNIX system took about a month to write and debug. The first UFRD for Windows NT also took about a month. Bringing up subsequent UNIX UFRDs is proportional to the time it takes to locate the necessary system include files and libraries.

UFRD agent scripts as seen in the LDAP e-mail address example (Figure 7) are usually simple and straightforward to write. The approach is: read standard input; split the line into an array (or list); and execute some existing application to add, delete, or modify some local data file. A Perl program to manage Linux user accounts by calling adduser, deluser, and moduser is 36 lines long.

Microsoft Windows NT	Microsoft Windows 2000*
Linux	Sun Solaris
SGI IRIX	Compaq Tru64 UNIX*
IBM AIX	Cray/SGI UNICOS

* under development

Figure 8: Operating systems where UFRD runs.

The current ARTIMUS implementation is reliable. We do health checks on our services every few minutes throughout the day. When a service is down for five minutes or more, an on-call person is paged. The last six months of web, database, and TED error logs show only four transient failures. Connecting to the web server failed twice and connecting to the database also failed twice. There were no prolonged failures resulting in an alarm page.

Since subscriber systems depend on ARTIMUS for updates, downtime is only noticeable to users wanting to make changes. Clearly, the main thing to avoid is losing the database. Backups and mirroring are both done.

Oddly, overall network service availability can actually be improved by adding a central database. LDAP and RADIUS are important services to the Laboratory. ARTIMUS makes replicating these servers as easy as deploying a system and adding a hostname to the TED host and service tables.

Data integrity is a key feature of ARTIMUS. It is also the most complex to implement. But, when done correctly, it promises the biggest rewards from the system. Our most difficult and time consuming problems are usually caused by incorrect, incomplete, or inconsistent data on one or more server systems. ARTIMUS

E-mail forwarding addresses (aliases)	E-mail lists*
URL's	Directories for web and FTP publishing*
UNIX accounts	Sharing group membership*
UNIX UID and GID's	Windows accounts and sharing groups
LDAP white pages	LDAP yellow pages*
LDAP POSIX users and groups*	LDAP mail recipients
CRYPTOCard token cards	RADIUS accounts and passwords
Kerberos accounts*	DCE accounts*
Authorizations*	DNS resource records*
SecurID token cards*	Entrust PKI certificates*

* under development

Figure 9: Applications that currently have UFRD service interfaces.

prevents these problems by simply rejecting bad information before it goes into the database.

Finally, security is an essential part of the ARTIMUS design. This system would not be deployed at Los Alamos if it did not protect users' network information.

Availability

Due to U. S. encryption export regulations, release of the source for the TED-UFRD propagation system must be controlled. Those interested in obtaining the source code may contact the authors directly to determine the necessary arrangements. Licensing restrictions beyond the export issue are otherwise liberal.

The web pages and database currently include many LANL idiosyncrasies. They would probably require significant changes to work elsewhere and therefore are not available at this time.

Acknowledgements

The authors would like to thank Los Alamos' Network Services Team. ARTIMUS is a product of the entire team's hard work and expertise. Particular recognition goes to Mary Gentry for her extensive work on the database design and implementation and her extraordinary commitment to detail. Susan Buchroeder was responsible for the Network Registry design. Amy Meilander provided the excellent database figures and lent us the use of her name(s) in the examples throughout the paper. Thanks to Jerome Heckenkamp, Elise Lee, and Giridhar Raichur for their constructive criticisms and thoughtful editorial comments. Finally, we are grateful to our supportive manager, Kyran Kemper.

Author Information

Alexander (Alex) Kent is a Systems Software Engineer for the Network Engineering Group at Los Alamos National Laboratory. His primary development projects include Laboratory-wide authentication and user account systems, network information propagation, and the Los Alamos firewall system. In addition, he is a full time graduate student at the University of New Mexico nearing completion of an MBA. Alex has a BS and MS in CS from New Mexico Tech. He may be reached via e-mail: alex@lanl.gov or snail mail: MS B255, Los Alamos, NM 87545.

James (Jim) Clifford is the Network Services Team Leader and a Systems Software Engineer for the Network Engineering Group at Los Alamos National Laboratory. His interests include Internet technology, Linux, and practical computer security. Jim has a BS from the University of Michigan. He may be reached via e-mail: jrc@lanl.gov or snail mail: MS B255, Los Alamos, NM 87545.

References

[1] Bob Arnold, "Accountworks: Users Create Accounts on SQL, Notes, NT, and UNIX," *12th Systems Administration Conference (LISA'98)*, pages 50-61. USENIX, 1998.

[2] David Curry, Samuel D. Kimery, Kent C. De La Croix, and Jeffery R. Schwab, "ACMAINT: An Account Creation and Maintenance System for Distributed UNIX Systems," *Workshop on Large Installation System Administration*. USENIX, 1990.

[3] C. A. DellaFera, M. W. Eichin, R. S. French, D. C. Jedlinsky, J. T. Kohl, and W. E. Sommerfeld, "The Zephyr Notification System," *Usenix Conference Proceedings*, USENIX, 1988.

[4] Jon Finke, "Invited Talk: Manage People, Not Userids," *10th Systems Administration Conference (LISA'96)*, USENIX, 1996.

[5] Jon Finke, "Oracle tricks and techniques in supporting systems administration," *System Administrator and Network Security Institute (SANS)*, 2000.

[6] Candace C. Fleming and Barabara von Halle, *Handbook of Relational Database Design,* Addison Wesley, 1989.

[7] J. Archer Harris and Gregory Gingerich, '1The Design and Implementation of a Network Account Management System," *Usenix Conference Proceedings*, USENIX, 1996.

[8] Neale Pickett, "Whiz," http://starship.python.net/crew/neale/src/whiz/ .

[9] Paul Riddle, Paul Danckaert, and Matt Metaferia, "AGUS: An Automatic Multi-Platform Account Generation System," *9th Systems Administration Conference (LISA'95)*, pages 171-180, USENIX, 1995.

[10] Henry Spencer, "Shuse: Multi-Host Account Administration," *Usenix Conference Proceedings*, USENIX, 1996.

[11] Gregory S. Thomas, James O. Schroeder, Merrilee E. Orcutt, Desiree C. Johnson, Jeffrey T. Simmelink, and John P. Moore, "UNIX Host Administration in a Heterogeneous Distributed Computing Environment," *Usenix Conference Proceedings*, USENIX, 1996.