The following paper was originally presented at the
Ninth System Administration Conference (LISA ’95)
Monterey, California, September 18-22, 1995

Finding a Needle in a Virtual Haystack: Whois++ and the Whois++ Client Library

Jeff R. Allen
Harvey Mudd College

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

Finding a Needle in a Virtual Haystack: Whois++ and the Whois++ Client Library

Jeff R. Allen – Harvey Mudd College

ABSTRACT

Powerful Directory Services are imperative in large networks to help keep users connected to the people and resources available on the net. This paper surveys previous work to build Internet Directory Services, and presents a set of requirements for the next generation of Directory Service technology. Next, the paper presents an overview of a new standards-track protocol named Whois++. Finally, client software written by the author is presented, and freely available server software is reviewed.

Introduction

It is no secret that the Internet is growing at an incredible pace. As a matter of fact, much of a system administrator's job is trying to keep up with this growth in all the diverse ways that it affects the organizations for which we work. With this expansion comes growing pains, as technology falls increasingly short of the demands placed on it. One technology that has fallen drastically behind in this rush of growth is Network Directory Services; the job of finding people, machines, and services on the network.

There is a loosely organized body of work meant to correct this problem, ranging from the Finger protocol (first documented in 1977) to the entire X.500 effort, dating from before 1985 to as recently as 1993. Still, we are faced with the reality that finding people on the Internet is one of those things best left to a network guru, one who knows all the right nooks and crannies into which to delve. Worse yet, the word is out to the "customers", the new class of users who are flooding onto the net, that the Internet's Directory Services aren't up to par. In *Newsweek's* Cyberscope column, the editors made the following observation: "The Internet provides myriad opportunities for procrastination. One of the best ways to avoid real work is trying to find someone's Internet address." [NW94] They go on to recommend a service named **netfind**, which works acceptably well, but falls short of the kind of ease of use required for a truly "good" solution.

With support from the IETF (as part of the WNILS working group and later, the ASID working group), another generation of researchers have attacked the problem, this time revamping the Whois system used by Internic (and previously, SRI's NIC) into a fully distributed, client/server system called Whois++. It is the author's belief that Whois++, while not perfect, will prove useful in the struggle to bring the Directory Service problem under control.

This paper introduces System Administrators to the concepts and technology of Whois++ so that they will be ready to adopt it if and when their users call for it, or when they recognize a problem in their organization that could be solved using Whois++. In particular, the paper will discuss an API and library that the author has developed to ease the task of writing innovative Whois++ clients.

The Directory Services Problem Explained

The Directory Service Problem is about connecting people to other people. There are many people on the network, and it is hard to keep all of the information about all of the people accessible to all of the rest of the people in an easy to use, easy to search directory system. Consider, on top of that, the tremendous rate of change of both the number of people, and the information about them, and the problem seems almost impossible. At all times, the problems of scaling in the system must be confronted head-on. Only through careful engineering, including application of client/server database concepts and distributed indexing technology, can a large-scale Directory Service system succeed. (Truth be told, it takes hard work, good politics, and a little luck too!)

For readers who want a more rigorous justification of the scaling problems the following may suffice: The fundamental problem is that the need for Directory Services grows as n^2 when the community increases by n members. This is due to the fact that in a group of n people, there are n^2 possible acquaintances, and if we assume the need for Directory Services among this group is roughly proportional to the number of acquaintances, then the need grows at the square of the rate of population growth. The constant of proportionality is anyone's guess, but the fact remains that the growth in the number of acquaintances is not strictly linear. Any system will have to take this characteristic of the problem into account, most likely by providing ample scalability in the design.

In addition to concerns of raw scale, there is another reason for the demand for Directory Services. The Internet is becoming a competitive market of consumers and producers, much like the US long-distance carrier market did in the last decade. To a professional who uses the network for business, the change in address necessitated by a switch to a lower-cost provider might mean lost contacts. Directory Services aren't just an interesting research challenge anymore. Users are in need of a quick solution to their problem: they want every contact to be able to find them quickly and reliably, no matter where they "reside" on the global Internet.

Due to the growing administrative complexity of managing names and addresses in the rapidly expanding Internet, a decentralized system of naming authorities has been created. These groups (like Internic and RIPE) all possess useful information about network entities, but it is hard for users to access, since it is spread across the network. This is a case where a strong distributed network Directory Service would serve users well. In fact, a protocol called RWhois is being developed to meet the immediate need of scaling the existing Whois system up to handle multiple naming authorities. RWhois, however, is tightly wed to the current Whois system. [RFC1714]

Finally, coming up with a scalable, extensible directory system may give other researchers the tools they need to solve other resource discovery problems. Effort is already being put forward by the folks at Bunyip Information Systems to merge some of the capabilities of Archie with the distributed characteristics of Whois++. Those working on a key distribution facility for a public key cryptographic system may also want to look into Whois++. As with any well designed tool, the eventual uses cannot even be imagined by the present day users.

Existing Systems

There are a number of systems currently in place to assist in the demand for Directory Services, but none of them have the scalability and ease of use required to solve the problem in both the short and long term.

Perhaps the earliest attempt of all at solving the Directory Service problem in the Internet was the Finger protocol, defined in RFC 742 in late 1977. This simple protocol was easy to design, easy to implement, and most importantly, solved the problem at hand nicely: it allowed the researchers on a handful of machines to find out who was logged into a handful of other machines on the net. From there, it evolved into a quick and easy way for people to distribute information about themselves to others. It remains one of the primary ways PGP keys are exchanged. With regard to solving what we now understand as the very complicated problem of Directory Services, Finger is a complete failure. In its time, it was a nice little application of the evolving network.

Why doesn't Finger fit the bill for a network-wide Directory Service? The biggest problem is that there is no cross-indexing in the system of servers. There are literally millions of servers out there, each holding a little bit of useful information. The problem is getting the right server, and retrieving the information of interest. Because the results of a Finger query can't be reliably parsed by a computer program, the arduous task of searching the global Finger database can't even be automated. It has to be done by hand by an experienced network user, one who knows how to find the information they are after.

Like Finger, Whois was a protocol designed to fill an immediate, pressing need. The Network Information Center (NIC) at SRI was building a database of useful information about Internet users. To share this information, a stateless, one-shot TCP based protocol was defined. It works just like Finger, except that a more advanced syntax for searching was established. To this day, the Whois servers at **nic.ddn.mil** and **rs.internic.net** get thousands of queries a day. (The InterNIC alone estimates they received 70,000 hits a day during the month of June, 1995.) [INIC]

There were several problems with Whois. First, the protocol was never really documented as an official Internet Standard. Instead, RFC 954 reads like an instruction manual for using the server. Since no reference implementation of the server was ever released for network-wide use, several incompatible versions of servers that implemented Whois-like services sprung up. That there was no reference server is understandable, since the NIC database is stored in a commercial database, and any code released to the public would be essentially useless without the same commercial server and database configuration. Even if a compatible group of servers had been installed as a result of the growth of Whois' popularity, there was still no cross-indexing in the system, hobbling its effectiveness for large-scale searches.

One feature that Whois introduced to the Directory Services field was the concept of handles. In database terminology these are "primary keys" for the server's database. Handles are alphanumeric identifiers that are unique within a given Whois server. They provide an easy way to come back to data retrieved earlier. In some cases, the rules used to make handles are so predictable that a search can be formulated in the form of a handle lookup, yielding a quick, focused search. This is an important feature that can be seen in all contemporary Directory Service schemes.

The ISO/OSI solution to the Directory Services problem came in the form of X.500. The X.500 development effort was spawned from the work on X.400, the OSI Messaging standard. It became clear to the X.400 developers that to make a user friendly mail system, a strong directory service would be required. This fact remains true today: behind most successful LAN e-mail systems lies a proprietary Directory Service system of some type. Typically, they are small and hard to manage, which makes them unsuitable as candidates for an Internet Directory Service. In e-mail systems without an integral Directory Service, like UNIX mail, one of the biggest problems users face is finding the right address to put on their e-mail.

When the X.500 effort got up to speed, people finally realized what a hard problem wide-scale Directory Services is and threw the heavy artillery at it. Many man-months of work went into writing the first X.500 specification. The result was a system that seemed to cover all the bases, dotting all the i's and crossing all the t's. The cost was complexity: the X.500 specification is hard to understand in its entirety and even harder to implement completely and correctly. Writing clients for the system requires understanding several layers of the OSI protocol, and mastering the TCP/IP interface used to bridge the gap between the Internet and OSI worlds.

Many organizations around the world use X.500 and/or systems derived from it to handle their Directory Service needs. It is by no means dead, and should certainly not be discounted. With that said, X.500 has had a very low acceptance within the networking community. In the opinion of the author, this failure to gain market share is due to three factors: complexity, politics, and search performance.

Due in part to the complexity of the protocol, there have been few servers made available in the public domain that support X.500. Quipu, the X.500 server that was distributed with ISODE, was poorly supported and hard to use as a result of its status as a research project; there were simply no resources to make it user friendly. Some commercial enterprises have invested in producing X.500 systems, but even so, there has been little growth in the use of X.500. One large company that has publicly endorsed the standard is Novell. However, the NetWare Directory Services system, which is based on X.500, operates over non-standard transport layers, and is not being deployed into a global infrastructure. Thus, even a high-profile player like Novell has not been able to make an impact in the use of X.500 on the public data networks.

Sadly, in the international standards process, sometimes political problems overshadow the technical ones. X.500 was unfortunately caught up in the heated Internet/OSI wars of the late 1980's, and had a slow start out of the gate as a result. That immense amounts of effort were lost is regrettable, but we must push forward, learn from the past, and try to launch a new Directory Service under more favorable political circumstances.

By far, the biggest failing of X.500 is its inability to deal effectively with large searches and multiple directory organizations. This is the kind of problem that could only be discovered through the limited real-world use X.500 has seen in the last few years. Because there is no shared information between servers, queries must be flooded out to all servers in the tree in an very inefficient manner. This causes unreasonable delays and large network cost for even fairly simple requests. Thus, at the highest levels of the tree, possibly where it was needed most, searching had to be curtailed or even turned off. [WEI95]

One other surprising development has come on the Directory Services scene in the last two years. The World Wide Web seems to be capable of some of the same features that we might demand in a new Directory Service. Data (in the form of user home pages) is stored all around the net in a highly distributed fashion. It is cross-indexed in many ways by many different servers, including Lycos, Yahoo, and Open Market's Commercial Sites Index. [LYCOS, YAHOO, OM] Perhaps the best thing about using the Web as a user directory is that the users are in total control of the data. This means that data is more likely to be quickly updated to reflect changing circumstances. The down side, though, is that the data will likely be completely unintelligible to intelligent clients, thus making some the of the very interesting features of a directory service system inaccessible. For instance, it won't be possible to make a user interface in which you double-click on a user's e-mail address to begin writing a letter to them. The browser simply won't know which bytes are the e-mail address, and which bytes are the user's favorite quote by Frank Zappa.

The ideal reconciliation of the two systems (Web based dissemination of user information, and structured, searchable Directory Services) will be to make one of the attributes stored by the Directory Service a URL pointing to the Web-based information about the user. This way, a structured search can be made for users, and once the desired person's record is found, one click of the mouse might take you to their homepage. A different click of the mouse might address a waiting e-mail message.

A Perfect Directory Service

Judging by the shortcomings of the existing systems, there are four characteristics that the next-generation system must have:

1. It must organize data into collections of attribute/value pairs, so that machines can parse the information automatically.
2. A new system must be distributed at all levels. Data storage and indexing need not be

separated, but they both must be distributed across the network to withstand heavy loading, and to provide uninterrupted service.

3. The new system must support fast and efficient searching at all levels. Without large-scale distributed indexing, this goal will be unattainable in a huge network like the Internet.
4. The organization of the data and indices in the system must be able to change over time as demands change. Indexes that cater to special interests should be possible.

The designers of Whois++ obviously had a set of goals like this in mind, since Whois++ fulfills each one nicely. This should not be surprising, of course; Whois++ is a next-generation directory service, meant to incorporate the lessons learned from the previous body of work.

The Whois++ System

As hinted above, there are really two distinct problems designers face when trying to create a Directory Service. First, they must deal with the raw data, defining protocols to transmit it while retaining automatically-parseable attribute/value pairs. Second, they must develop indexing and searching protocols to allow users to quickly find the data of interest. The designers of Whois++ divided the design into two conceptual pieces, one to serve data, and one to index it. In reality, these pieces can be implemented in the same server, so that a given server can serve local data and index the data of remote servers too.

The design of the database server is relatively straightforward. It is the part of Whois++ most reminiscent of the original Whois service. The main idea is that the database server will return templates, which are collections of attribute/value pairs identified by a template handle. Each server in the system has a server handle, which will eventually be assigned by the Internet Assigned Numbers Authority (IANA).[1] These handles are unique among all servers in use on the Internet. Within an individual server, each template handle must be unique. This makes it possible to uniquely identify any template, anywhere on the Internet, using just a server handle and a template handle.

Within a template, attributes are distinguished by attribute names. Since they are transmitted in full ASCII text, and are often stored in the server the same way, they are arbitrarily extensible. The server administrator can add attribute names to the server's templates as they are required. This begs the question of who controls the definitions of attribute names, and how do they impose their will on the various server administrators? The solution offered by Whois++ is typical of the Internet community: there is no schema administration authority. Various IETF working groups will likely publish advisory RFC's to help new administrators choose reasonable attribute names. It is the author's opinion that ultimately, the Whois++ client developers will have control over the schema. After all, what use is a fancy new attribute name if no Whois++ clients will recognize it and display it usefully?

From the point of view of the client, retrieving all that data is all fine and dandy, but the important thing is to be able to succinctly search for records in the database. The searching syntax is based on the

[1] Currently server handles are being registered by Patrik Faltstrom, <paf@bunyip.com>. Discussions are underway with IANA to find a way to assign Whois++ server handles without significantly impacting their existing workload.

```
C: <connect to muddcs.cs.hmc.edu, port 5050>
S: % 220-This is muddcs running Bunyip-Whois++: DIGGER 1.0.2
S: % 220 Ready to go!
C: handle=jeff
S: % 200 Search is executing
S: # FULL USER CSHMCEDU0 JEFF
S:  NAME: Jeff R. Allen
S:  EMAIL: jeff@hmc.edu
S:  ORGANIZATION-NAME: Harvey Mudd College
S:  DESCRIPTION-URI: http://www.cs.hmc.edu/~jallen
S: # END
S:
S: % 226 Transaction complete
S: % 203 Bye, bye
S: <disconnect>
```

Figure 1: This is a transcript of the retrieval of a single template from a Whois++ server. The server handle for this particular server is ''CSHMCEDU0''. Lines preceded with ''S'' come from the server. Lines preceded by ''C'' come from the client.

original Whois protocol. The syntax is specified in exacting detail in the protocol specification, so it would be pointless to cover it completely again here. [DEU95] Basically, a search string is composed of tokens from the template(s) that you'd like to match. The keywords **and**, **or**, and **not** can be used to modify the search. To further constrain where in the template a token can match, attribute identifiers can be used. Thus, a search for "**Name=Smith**" will not match a record in which the only "**Smith**" token is in the "**Postal-Address**" attribute. Finally, a specific template can be retrieved by using a handle search, assuming the user knows the handle. The form for this type of search is predictable: "**Handle=** *handlename*". Unless specifically requested to be case-sensitive, all matches are case-insensitive. Attribute matching is always done case-insensitively.

When a search is too broad, it may return many more hits than are actually useful to the user. In some cases, searches can be devised to return virtually every record stored by the server. To prevent simple overloading by broad searches, and malicious attempts to download the entire database, Whois++ servers enforce several constraints on the searches. The most important is "Max-Hits". An absolute limit is set on Max-Hits by the server administrator. No client can ever receive more than this number of templates in response to a single request.

Without additional cross-indexing technology, however, Whois++ is not much better than Whois, or Finger for that matter. The cross-indexing capabilities of the protocol are what make it so special, and may in the long term, allow Whois++-based systems to solve problems not directly related to Directory Service. The cross-indexing takes the form of centroid passing. In physics, the centroid of an object is the center of all mass, a kind of balancing point. In the Whois++ world, it's a list of tokens that represents all of the words known by a server. More precisely, the centroid of a particular template type in a server is the collection of all tokens occurring within all templates of that type. The example in Figure 2 may make the definition clearer.

```
If the server       Record 1             Record 2             Record 3
contains these      Template: Person     Template: Person     Template: Person
templates:          First-Name: John     First-Name: Joe      First-Name: John
                    Last-Name: Smith     Last-Name: Smith     Last-Name: Jones

Then the centroid       Template:       Person
will look like          First-Name:     Joe, John
this:                   Last-Name:      Smith, Jones
```

Figure 2: The centroid for the three records shows that all the tokens originally present are accounted for, even though the centroid is much smaller.

```
C: <connect to muddcs.cs.hmc.edu, port 5050>
S: % 220-This is muddcs running Bunyip-Whois++: DIGGER 1.0.2
S: % 220 Ready to go!
C: smith
S: % 200 Search is executing
S: # FULL USER CSHMCEDU0 SMITH
S:  NAME: Robert Smith
S:  EMAIL: Robert_Smith@hmc.edu
S:  ORGANIZATION-NAME: Harvey Mudd College
S: # END
S: # SERVER-TO-ASK CSHMCEDU0
S:  Server-Handle: CSHMCEDU5
S:  Host-Name: MUDDCS.CS.HMC.EDU
S:  Host-Port: 5055
S: # END
S: % 226 Transaction complete
S: % 203 Bye, bye
S: <disconnect>
```

Figure 3: This transaction shows a Whois++ client/server interaction in which both a template and a referral are returned. It is the client's responsibility to carry out the additional query suggested by the server named ‘‘CSHMCEDU0’’ on the server named ‘‘CSHMCEDU5’’.

A centroid represents the set of knowledge the server has about a its domain of the distributed database. Let's call this little server with the three templates above "Server A". If its centroid were passed to another server (call the receiver "Server B") responsible for indexing all the Whois++ servers on the network, it would be immediately obvious to Server B that Server A can't help with a query like, "Last-Name=Schwartz". This is because "Schwartz" doesn't appear under the "Last-Name" attribute in the centroid that Server B received from Server A.

As soon as servers start passing centroids, a kind of order, or hierarchy, develops. Those servers with more substantial centroids (gathered from several subservient servers) are more likely to be able to match a query. However, when they match a query based on data from a remote server, there is no way for them to reconstruct the template to be able to present it to the client. Nor does the master server even have the authority to do so; for all it knows, the template may have changed in the subservient database since the centroid was received. Instead of attempting to return all templates that match a query (an impossible feat, given the information available in a centroid), servers are allowed to return referrals to other network servers that may be able to fulfill the query. Figure 3 shows a referral from a master server for Harvey Mudd to a subservient server, also at Mudd.

Directory Services in the past (notably X.500) have suffered because their indexing structures were fixed by the design. The Whois++ design attempts to get around the problem by easing the restrictions on server-to-server connections. Because servers can pass centroids in virtually any configuration, multiple indexing-server configurations are possible. Since the client is responsible for tracking cross-references within the global database, it can detect loops in the references it receives. Thus, there is no need to protect the system's hierarchy from loops. Instead of constructing a strict server tree, administrators will create a server mesh.

Currently, those servers which are running are configured as a strict tree, with the server at **services.bunyip.com**, port 63 as root. However, it isn't hard to see how a parallel mesh might be useful, one that only indexes commercial entities, for instance, or one which will specialize in templates which represent files available for anonymous FTP.

We have reviewed both the database server and the index server. The only piece of the system left to explore is the client. Whois++ clients will likely come in all shapes and sizes, as opposed to the very limited clients available for Whois today. They will also likely be hidden deep in other applications, which will benefit from using the protocol. Whois++ clients will be able to make use of the data returned from a Whois++ server in ways Whois clients were never able to. For instance, an e-mail application might have a built-in Whois++ client. At the "To" prompt, the user will request help finding a user's name. By making a Whois++ query, they will find the name they are looking for. The client software will be able to scan the attribute/value pairs that are returned and find the one for "E-mail address". With a double-click (or a drag-and-drop, or whatever) the user can add the recipient to the message. This type of feature is something Microsoft Mail and Lotus cc:Mail users have had all along, but they have never had the entire Internet indexed via an Internet standard protocol.

What's involved in writing a client? A client needs network control code, to make and break connections to servers. It needs to parse the slightly more complicated messages Whois++ servers return. Finally, it needs to manage the search, so that server loops are avoided, and so that searches get expanded in a sensible way to make sure the requested information is found somewhere in the mesh.

Because so much of the intelligence required to conduct a distributed query has been designed out of the server and into the client, it will be somewhat harder to write Whois++ clients than it was to write clients for previous services. However, with a general purpose, easy to use API (and its implementation, a Client Library), writing clients could become easy. The details of Whois++ server interaction, loop detection, and query management can be left to the library, while the programmer concentrates on a good user interface, or on the useful application of the retrieved information. Whois++ clients may not always have user interfaces, either. Any program that uses the Whois++ protocol may profit from use of the library. For instance, an X.500 to Whois++ gateway daemon might make use of the library.

The Whois++ Client Library (WCL)

The Whois++ Client API specifies a set of data types and function calls used to interact with Whois++ servers. The implementation (written in C with an interface to Perl 5) makes it easy to write Whois++ clients. The library has the following features:

- A server cache, to amortize high TCP startup time across multiple queries.
- Easy-to-use exception handling using callbacks.
- A full implementation of the Whois++ mesh traversal algorithm. [FAL95]
- No hard-coded limits on template, attribute, or value size.

The Whois++ Client Library (WCL) comes with a text-based client for use for both testing the library, and as a ready-to-use Whois++ client. It also functions in one-shot command-line mode for use in Perl 4 and shell scripts. A prototype HTTP-to-Whois++ gateway geared to making user lookups possible via a friendly Web-browser interface is also included with the distribution, demonstrating just how easy it is to put a nice user interface on the pre-existing library code.

The library compiles on SunOS 4.1.3, Solaris 2.x, and Irix 5.x. It is POSIX- and ANSI-compliant source, which should integrate easily with most development environments. See the section named "Software Availability" below for more information about how to get the package.

The Library in Action

To whet the reader's appetite for the library, two example uses of WCL will be described here. The first is the Whois++ to HTTP gateway mentioned above. This type of application is certain to make users happy, but how can an overworked system administrator benefit from Whois++ technology? The second example shows a subroutine that could be added to a Perl 5 user creation script to derive the new user's vital statistics, given a handle in the existing Whois++ server.

The first thing a user of the HTTP to Whois++ gateway sees is a forms-based representation of a Whois++ query. Upon submitting the form, the Whois++ query takes place. If there are several matches, an intermediate page requesting a selection pops up. Once the user has narrowed the query to a single template, the system returns and displays a page describing the user. If the user has made the required information available, a hypertext link to their homepage, a picture, and a ''mailto:'' link are all included by the gateway.

For the hard-core sysadmin, who prefers not to use a GUI, here's a more useful tool: a Perl 5 subroutine which can automate the information-gathering part of a user creation script. This example assumes that a corporate database representing all users is already available via a Whois++ server. (See the section on "Whois++ Servers" below for an idea on how this might be accomplished.) Furthermore, it assumes that the handles in the database are employee identification numbers. System Administrators who work for academic institutions may want to think of these assumptions in terms of a "student database" and "student numbers". Figure 4 shows a rough sketch of what the relevant parts of the script might look like. The first line imports code needed to make sure that the functions will be autoloaded at the appropriate time. After that, a prototypical subroutine call is shown. This call would likely come near the beginning of the script, after the employee number has been read from user input, or from a file. Finally, the subroutine is shown. In this case, the library is only being called upon to offer server management and template parsing services. Since we know that the employee number must map to a template on the local server if it is valid, there is no need to go off probing other servers in the mesh. Finally, the name and uid are returned. In the case of an invalid employee number, a list consisting of a pair of empty strings will be returned. The calling program can then take appropriate action.

Whois++ Servers

Like all protocols in the Internet suite, a reference server of one sort or another has existed throughout the development of the protocol to help make sure that the grand ideas were actually implementable. The reference server (currently the only one available, though that's likely to change in the coming months) was written by Patrik Faltstrom of

```
use WCL;
[... user code goes here ...]
($uid, $name) = getUserInfo($employee_num);
[... rest of script goes here ...]

sub getUserInfo {
  my ($emp) = @_;
  my $slot, @res, %av;

  $sid = wclMakeServid($serverhost, $serverport) if (! defined($sid));
  $slot = wclGetServer($sid);
  @res = wclParse(wclCommand($slot, "handle=$emp"));

  # 4th element of result is a list of the a/v pairs. This conversion
  # implicitly loses ordering and repeated keys, which are defined by the
  # protocol to be significant. We choose to ignore them for this example.
  %av = $res[3];
  return ($av{"UID"}, $av{"Name"});
};
```

Figure 4: A subroutine to retrieve user account information from an existing corporate database for use in an account creation script.

Bunyip Information Services, Inc. Bunyip has generously allowed the Internet community free use of the server (named 'Digger'), although its code is copyrighted code and may not be redistributed. Availability of the most recent version is discussed in the section named "Software Availability", below.

Digger is written in C and uses an SQL database for the back end, where the actual data and centroids are stored. Digger currently supports two database backends (Oracle and mSQL), and can be ported relatively easily to other database systems. mSQL is a relatively new publically redistributable shareware SQL server written by David J. Hughes of Bond University.[2] Digger ships with the newest version of mSQL, though it is also available separately from Bond University. [BUNYIP, MSQL]

Installing Digger is a breeze, thanks to Patrik's use of GNU autoconf and a clever INSTALL script. Installing mSQL is just as easy. Because all of the code is written to the POSIX specification, it should be easy to get the servers running on other fairly modern machines. As usual, both authors welcome e-mail describing any minor difficulties encountered during the installation process. Both packages are known to build correctly on SunOS 4.1.x, Solaris 2.x, HP-UX, Linux and OSF/1.

The biggest job a Whois++ server administrator faces is acquiring and formatting data into a form suitable for Digger's template insertion program. There are both technical and political problems here that sysadins will need to solve locally. Bunyip has promised a set of scripts to aid template conversion (send e-mail to digger-info@bunyip.com for more information about these scripts). Most administrators will want to set up a system which provides periodic updates from a central database, since it is important to keep the Whois++ server's data up-to-date with respect to the main database.

A truly adventurous SQL hacker may like to link Digger directly to an existing user database using the internal interfaces to SQL that Digger provides. Though there are no known examples of this type of project underway, only small pieces of code should theoretically be required, and the benefit (no time-delay induced inaccuracy between the databases) should outweigh the investment. Perhaps a future LISA paper will describe a project like this.

Administrative and Legal Challenges

Because of the perceived potential for abuse, developers of electronic Directories have often faced opposition from non-technical, but nonetheless interested parties, thus compounding the difficulty of the Directory Services Problem. This is a part of the problem that needs to be faced, perhaps even more urgently than the technical aspects.

Various groups, including privacy-advocacy lawyers and law makers, corporate executives, and academic administrations have all gotten involved in the fray at one time or another, each pushing essentially the same argument: electronic directories are an infringement on a person's right to privacy, and must therefore be unconditionally blocked.

The fundamental problem with the argument that electronic directories invade people's right to privacy is that less visible, but highly invasive directories already exist at the disposal of the privileged few. The entire Direct Marketing industry revolves around managing, trading, processing, and building lists of names. These directories are not made available to the public for useful ends. Instead, literally millions of tons of unsolicited mail is sent to the "lucky" members of these mailing lists. Other examples of privacy invasions from electronic directories surround us: credit reporting agencies, credit-card records used to create consumer spending histories, etc. Finally, it's helpful to ask oneself, "what's the difference between the Directory Service offered by a telephone company and one offered over a computer network?" If the telephone company can provide such a useful service, network providers certainly should be able to.

It all comes down to the rights of the person whose privacy will allegedly be invaded. All of the potential good that a public access electronic Directory can do, in this author's opinion, is worth the risk that a little bit of electronic privacy might be lost. But that's just the point: this tradeoff is an individual decision, and every person who is listed in every directory has a right to control how much information is available, and to whom. In addition, every user always has the right to submit changes to the data, and have their records promptly updated. These rights are respected by the other maintainers of lists. The Direct Marketing Association provides a registry of people who want to be excluded from the industry's "service".[3] Telephone companies provide unlisted numbers as a regular service to subscribers. Electronic Directory providers can and should provide the same types of service.

These principles of user control and notification are discussed in RFC 1355 and RFC 1295. [RFC1355, RFC1295] Administrators would be well

[2] mSQL is a very important, very useful piece of software which the Internet has needed for some time. David Hughes has earned the right to license his software for a small fee. Please read the license which comes with his software carefully and comply with it, if you choose to use his software.

[3] Call the Mail Preference Service at (212) 768-7277 and ask to be added to their Suppression File. For more information about protecting your privacy, see the FAQ on junk mail posted occasionally by Chris Hibbert <hibbert@netcom.com> to misc.consumers.

advised to review and share these particular documents with management personnel before attempting to put a large database online for public use. Taking the time to put together a coherent policy on how users relate to the data being published about them may reduce future problems.

It's Not Just For People Anymore...

With a service as general and powerful as Whois++, data of virtually any type can be indexed and served via the Whois++ protocol. Just some of the possible applications include the items below.

Public-key encryption system key-servers

While it will likely be impossible to securely offer full key escrowing services via a Whois++ server, the protocol will be useful to handle insecure key exchanges like those used by PGP. In this case, the Whois++ client/server system would simply automate what is already commonplace on the Internet today. Key exchanges are usually manually carried out via the Finger or HTTP protocols.

URN-to-URL translators

Some plans to supplant Uniform Resource Locators (URLs) with more general Uniform Resource Names (URNs) call for an infrastructure of servers to translate the various tags. These servers must come up with the closest, fastest, or cheapest URL for a given URN, and may be called upon to provide reverse mappings too. Whois++ should be up to the challenge, even though the translation database will be widely distributed and quickly changing.

Interpedia search engines and/or SOAP managers

Plans for an Internet-wide, publically authored encyclopedia representing the knowledge of all the Internet's users call for careful indexing and very fine-grain data distribution. In addition, the management of SOAPs, or Seals of Approval may be a problem suited to Whois++ technology. [RHINE]

Conclusion

It is the author's opinion that the Whois++ architecture will be a useful step forward in Directory Service technologies, enough so that it is worthwhile to develop clients for it to spur the market for Whois++ servers. With the use of the Whois++ Client Library, it should be easy to produce powerful, interesting Whois++ clients to solve the problems of a growing Internet. And with powerful, easy to use clients, the market for servers (both free and commercial) will develop.

Whois++ and WCL will be up to each of the challenges above. All that's required is a little imagination and some hard work. WCL eases the burden, leaving the programmer free to work on the challenging problem of managing and presenting the data in a useful way.

Software Availability

All of the relevant URLs are provided in Figure 5. The version numbers are correct at the time of this printing, but they will change over time. With the exception of mSQL, all of these products are free, copyrighted works. There is a small shareware fee for mSQL.

Author Information

Jeff R. Allen is a full-time student in his final semester at Harvey Mudd College in Claremont, CA. His computer-related interests include stupid Perl tricks, innovative user support, and single-handedly solving the Directory Service Problem, though graduating must take priority to all others at this time. When he is away from computers, Jeff likes to read, unicycle, and plan pranks against CalTech (though they seldom actually see fruition). E-mail messages, including job offers, are gladly accepted at: jeff@hmc.edu.

References

[ALLEN95] Allen, Jeff R. "Whois++ Client API v2.0a" (work in progress) *http://www.cs.hmc.edu/~jallen/wppcl*

[BUNYIP] Bunyip Information Services, Inc. Digger home page. *http://services.bunyip.com:8000/products/digger/digger-main.html*

[DEU95] Deutsch, Peter, Rickard Schoultz, Patrik Faltstrom, Chris Weider. "Architecture of the WHOIS++ Service" (work in progress) *ftp://ftp.internic.net/internet-drafts/draft-ietf-asid-whois-arch-03.txt*

[FAL95] Faltstrom, P., R. Schoultz, C. Weider. "How to interact with a Whois++ mesh". (work-in-progress) *ftp://ftp.internic.net/internet-drafts/draft-ietf-asid-whois-mesh-01.txt*

[INIC] Personal correspondence with Internic engineers at: <action@internic.net>, July 1995.

API:	http://www.cs.hmc.edu/~jallen/wppcl
	ftp://ftp.hmc.edu/pub/research/wppcl/api.ps.Z
WCL:	ftp://ftp.hmc.edu/pub/research/wppcl/wcl-2.0a.tar.Z
Digger:	ftp://ftp.bunyip.com/pub/digger/software/digger-1.0.4.tar.gz
mSQL:	ftp://ftp.bond.edu.au/pub/Minerva/msql/msql-1.0.7.tar.gz

Figure 5: Where to find the software

[LYCOS] Mauldin, Michael L. "Lycos, The Catalog of the Internet." *http://lycos.cs.cmu.edu*

[MSQL] Hughes, David J. mSQL Information and Distribution. *ftp://ftp.bond.edu.au/pub/Minerva/msql*

[NW94] *Newsweek*, December 5, 1994, page 10.

[OM] Open Market Inc. "Open Market's Commercial Sites Index" *http://www.directory.net*

[RFC742] Harrenstien, K. RFC 742, NAME/FINGER. December 30, 1977. *ftp://ftp.internic.net/rfc/rfc742.txt*

[RFC954] NICNAME/WHOIS. K. Harrenstien, M. K. Stahl, E. J. Feinler. October 1, 1985 *ftp://ftp.internic.net/rfc/rfc954.txt*

[RFC1295] The North American Directory Forum. User Bill of Rights. January 1992. *ftp://ftp.internic.net/rfc/rfc1295.txt*

[RFC1355] Curran, J., Marine, A. Privacy and Accuracy Issues in Network Information Center Databases, August 1992. *ftp://ftp.internic.net/rfc/rfc1355.txt*

[RFC1714] Williamson, S., Kosters, M. Referral Whois Protocol (RWhois). November 1994. *ftp://ftp.internic.net/rfc/rfc1714.txt*

[RFC1758] NADF Standing Documents: A Brief Overview. The North American Directory Forum. February 1995. *ftp://ftp.internic.net/rfc/rfc1758.txt*

[RHINE] Rhine, Jared. Interpedia Research information. *http://www.math.hmc.edu:8088/interpedia*

[WEI95] Weider, Chris, Jim Fullton, Simon Spero. "Architecture of the Whois++ Index Service". (work in progress) *ftp://ftp.internic.net/internet-drafts/draft-ietf-wnils-whois-05.txt*

[YAHOO] Filo David, Jerry Yang. "Yahoo." *http://www.yahoo.com* EM