The following paper was originally published in the
Proceedings of the Eleventh Systems Administration Conference (LISA ’97)
San Diego, California, October 1997

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

Automation of Site Configuration Management

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

Although there are countless tools to track and manage the configuration of large numbers of Unix systems, there seems to be a lack of tools to manage the interaction and dependencies between systems. As our site has grown, many machines provide services that are required for the operation of other machines and applications. We have been unable to maintain accurate lists of services and servers, and even routine system upgrades have resulted in unexpected service outages.

To address this problem, we are developing a system to automatically detect many of these service dependencies, and generate up to date server listings. In addition, it provides a general framework for indexing and accessing troubleshooting, operational, installation and a number of other types of documentation. The system also assists in verifying the configuration of systems being installed, and assists with the real time testing of services.

Introduction

As the number of Unix machines being supported grows, support groups develop or install more and more tools to automate and streamline the system installation and update process. These range from moderately simple file copying tools such as rdist or package[1] to commercial products such as "Tivoli," to any number of individual development efforts such as *Config* [12], or GeNUAdmin [9] and others that can be found in the proceedings from past LISA conferences. Like these sites, we have also developed and installed tools to manage system configuration.

In addition to managing system configuration, it is often useful to collect system configuration (both hardware and software) and state information in a central location. This can be done in a number of ways, such as with syslog [14] or SNMP [11] or via the file system (see the section on *My-State*).

While we would all like to work in a problem free environment, that is often not possible. Failing that, it is desirable to at least detect system problems before our users do. Since services are often spread over many machines, tools have been developed to detect problems on these machines, either by waiting for error messages with tools such as *swatch93* [8], or by going out and testing things with tools like *pong* [10].

While all of these tools fill important roles in managing large numbers of workstations, where they are generally lacking is in dealing with the dependencies **between** the applications and servers, and with the maintenance of the configuration files. For example, while *pong* is a great tool for testing large numbers of services, it can't test servers that no one told it about. While there has been some work in the area of detecting these dependencies, these have either been limited in scope to the network configuration such as *Fremont* [15] or *Ined* [13], or have it as part of general configuration management system, and is restricted to checking just what is managed via that system [1].

An often repeated request from our Network Operation Center (NOC), was an up to date version of the *Server List*. This was a list of all of our server machines, and the services each one provided. These services ranged from domain name service, to print servers, to license servers (with multiple applications per server), authentication servers, time servers, mail servers, file servers, PC and Mac servers and so on. To make matters worse, there were often multiple services per machine, and services were moving from one machine to another machine to accommodate changes in OS level, changes in network topology, security and performance concerns. In addition, short term test services would often be set up in odd places to provide a new service, or attempt to isolate a problem with some other service, and be left in operation accidentally. This has resulted in a number of unplanned service degradations or complete failures, when one staff member takes a machine down for service or redeployment, not knowing that someone else had set up some special service on it. The most recent example, is a week after the successful upgrade of two DNS and license service machine, we remembered that the machine was also the YP master for our site. It wasn't until we created a batch of userids for a special class and no one could sign on, that we discovered the

[1] Package is a tool supplied with AFS that *pulls* in files and directories to a machine based on a configuration file. It detects new versions and installs them as needed.

problem. The YP master function has been moved there a year ago when the real YP master machine had a CPU failure, and never got moved back. Oops.

Since this was not the first time something like this happened, and that the request from the NOC was quite reasonable, we decided it was time to come up with a solution. For the past six years, the RCS[2] userids at RPI have been managed with a locally developed package called Simon [2, 3] which is built on top of an Oracle relational database. In addition to managing userids, this system also managed other aspects of our site such as the printing configuration file [4], the host table and DNS files, and even such mundane things as our telephone directory [6].

Given our past success with using Simon (and Oracle) to solve any problem, the direction for this solution became pretty clear. While the request from the NOC was for a ''server list,'' given the nature of the data, a hypertext document would be better for general use as you navigate from service to server instance to client and back. Between the telephone directory project, and other projects that required the generation of HTML pages from the database [5], we had the underlying technology to approach the problem.

MyState

Like many other sites, we had the desire to be able to track system configuration in a central location. To this end, we wrote a *Self_Exam* script that maintains a file called `/etc/MyState` which holds a number of records with information on that particular system. This file would then be moved back into our central file system as part of the standard *Post_Package*[3] run and placed in a `ShippedBackFiles/MyState` directory, using the hostname as the filename.

[2]RCS, the Rensselaer Computing System, a collection of 700 workstations and Unix timesharing machines available to all students, faculty and staff.

[3]After a successful package run, a post package script was run that would restart servers, and clean up installation details that can not be handled by simply copying in a file.

Each record in the file consisted of three fields, the item name, the item value, and the method used to obtain the value. This was important as different OS versions would report the same item in different ways or in different units. The data from a typical `/etc/MyState` file is shown in Table 1.

These `MyState` files were useful just as stand alone files. Since we had them all in the central filesystem, all stored in the same directory, you could make quick searches for things by using simple UNIX tools such as `grep *`, or just simply `cat` out the file for the host of interest. Unfortunately, that was about all you could do with it. Other information such as the location of the machine, the owner, maintenance status, and so forth was not readily available. In addition, we often were interested in grouping machines by different attributes. For example, OS upgrades often required that we identify all machines of a certain type, with less than a certain amount of disk or ram. In addition, there was no facility for tracking changes in configuration, or even knowing when a particular file was obtained (knowing when the configuration of a machine changed is handy when you have memory thieves operating at your site!).

In addition to the `MyState` files, there are often other interesting machine configuration available on some platforms. For instance, under AIX, you can issue an `lscfg` command and get lots of useful information about the hardware configuration of a machine. We often find ourselves juggling machine configurations, moving interface cards, disks and memory between machines as our needs changes. While the `MyState` file could tell you how much RAM a machine had, it would not tell you if you had one 32MB SIMM or eight 4MB SIMMs installed in a machine (and how many available RAM slots were left). Rather than visiting every machine when we

Name	Value	Method
hostname	simonsrv.sss.rpi.edu	hostname
hostid	0x8071640c	hostid
ipaddress	128.113.100.12	grepped from /etc/hosts
gateway	128.113.100.244	netstat -r
ypmaster	asher.its.rpi.edu	ypwhich
memory	128M	lscfg
macaddress	08.00.5a.cd.5d.42	netstat
arch	power	assigned to all rs6000s
model	rs6000 250	uname -a
os	AIX 4.1	uname -a
swap	256M	lsps -a
disk	5596M	lspv

Table 1: Sample `/etc/MyState` file contents

wanted to ask these questions, or expanding or duplicating the existing MyState files, we decided to dump everything into the database.

To this end we defined a new table, System_State, in the Simon database (see Table 2). We make an entry in the table for each line in each MyState file. When we first load a MyState file, we record the System_Id [4] of the host, the means used to collect the data (in this case, MyState), the source of the data (in this case, the MyState file), and for each line, the name of the item, the current value of the item, and the third field. We also record the current time and date in the Date_Obtained field.

When we run the load_system_state program on a machine, it selects all the existing records for that particular machine and means, and sorts them by Item_Name. It then reads and sorts the data from the MyState file. Once both of these lists are in place, it compares them one by one. If there is a new entry, it is inserted as described above. If an entry is missing, the record is marked as deleted by setting the Date_Obsolete field to the current date. if the Item_Names match, the Item_Value and Item_Extra are compared. If they are still the same, the Date_Verified field is updated. If they are different, the old record is deleted and a new record is inserted with the updated information.

In order to handle data sources besides /etc/mystate, the load_system_state program has a list of items to check. Each item has either a filename to read or a pathname to execute, a flag indicating file or program, a standard value for the Means, some parsing information[5] to assist in decoding the information and what operating system the entry can handle. The OS information lets us collect information using vendor specific commands, such as lscfg under AIX.

All of this information is available to other database applications. For instance, we will very likely use the "OS" information for a machine to sort the application usage via license server logs [7]. This will be useful in deciding how to handle version differences of applications across the different platforms. Of perhaps more general use to our staff, we also generate web pages, one for each system, that include administrative information (owner, sys admin, contract status, etc), as well as the information for that system stored in the System_State table. We also generate index pages for each of the common items and common values. When generating these pages, the extraction program attempts to identify common attributes (such as memory or disk size) and generate the page. Unique

[4]The System_Id is an internal database key that corresponds to one particular machine. One advantage of this, is that the entries (except for hostname!) are not impacted by a host name or domain change, and we can trace the history of a physical machine.

[5]Currently we support fixed columns for the fields or character delimited fields.

Name	Type	Size	Description
System_Id	Num	22	The Simon.Systems.System_Id of the machine in question.
Means	Char	12	The facility used to obtain this information. This might be /etc/My_State, or possible platform specific configuration commands such as lscfg. All hosts should have at least MyState info recorded.
Item_Name	Char	32	The name of the particular item we are recording. For MyState files, this would be the first column. This essentially addresses the question "what"
Item_Value	Char	256	The value for this particular item. For MyState, this is the second column. This tells us "how much."
Item_Extra	Char	256	An additional field to handle extra info for this record. For the MyState file, this is the means to obtain this particular bit of information. This can be important when attempting to compare the same "what," since different units or methodologies may have been employed to get the result. The specific definition of this field depends on the means.
Date_Obtained	Date	7	The date when we first encountered a record of this type. For CHANGES to a value, this value will be carried forward.
Date_Verified	Date	7	The last time we checked this information against the live system.
Date_Obsolete	Date	7	This is when this record is obsolete. This can be because the value changed, or the Item_Name no longer exists.

Table 2: System_State Oracle Table Definition

information such as a hostname or ip address do not get index pages.

Services and Servers

The key to the entire project is identifying all of the services we provide. These services range from our AFS cell and DNS, to license serving for applications, to information services such as email and usenet. These services are generally provided by one or more server machines. It is important to maintain the distinction between the "service" we are describing, and the "servers" (machines) that we are actually using to provide the service.

Each service has a number of attributes. One of the important attributes of each service is its priority. The priority will help the operations staff determine the order to try to solve problems when faced with more than one, and even if a late night page or phone call to the systems administration staff is appropriate. For example, a failure of the domain name service will result in problems all over campus, affecting hundreds, or even thousands of users, as compared to a failure of the AutoLev[6] license server which will only inconvenience a few people. Thus, we will give a

[6] I am not quite sure what AutoLev is, but according to the license server logs, only a few people use it.

Service Priorities	
Critical	Provides an essential service to the campus community. Many people are impacted by a failure. An example would be the Domain Name service.
Business High	Likely to be an administrative service, these are needed during the business day.
Academic High	These are things used in the curriculum, often in the classroom. When classes are in session, these have to be operational.
Moderate	A failure here will inconvenience a number of people, but does not warrant heroic efforts at recovery that the critical or high might require.
Low	Services that are nice, but not needed for the overall mission. These may be very lightly used applications, or more recreational type applications.
Experimental	Generally not in the repair queue at all. The person deploying the service may be interested in problem reports, but no one else.

Table 3: Service priorities.

Name	Type	Size	Description
Service_Id	Num	22	A Simon.Peoplecount.Nextval value that is used as a the primary key for the service management system. Many other tables will reference this column.
Service_Name	Char	64	The actual name of the service, such as "Domain Name Server" or "Xess License Server".
Service_Type	Char	32	An optional service type to be used in ordering reports and collecting similar function together.
Priority	Char	32	A rough indication of the priority of this service in general.
Clerk	Num	22	The Simon.People.Id of the person to create or last change this record.
Effective_Date	Date	7	The date when this record was created or last changed.
When_Inserted	Num	22	The Simon.Transcount.Nextval of when this record was inserted into the database. This makes the Effective_Date somewhat redundant, but that format is easier to display.
When_Updated	Num	22	The Simon.Transcount.Nextval of when this record was last changed. When_Marked_For_DeleteNum22T{ The Simon.Transcount.Nextval of when this record is considered obsolete. It may have been replaced, or simply considered to be deleted.
Comments	Char	2000	A place to hold a short description of this service. This information is included in some reports and on web pages and the like.

Table 4: `Service_List` Oracle Table Definition

higher priority to restoring the DNS service rather than restoring the AutoLev service. Our current list of priorities[7] is listed in Table 3. These priorities are likely to be revised once we have worked with them for a while. The priority classifications could be of interest to our users, but could potentially be a political nightmare.

Another attribute of a service that we find useful is a service type. These are just general categories like "File Server" which would include our AFS file servers, Novell file servers, NFS exports, and so on. Other categories include "Name Servers," "License Servers," "Unix Commands." These types are used for grouping sets of services together in some of the index web pages, or when searching for a specific service. This information is stored in the `Service_List` table (see Table 4).

All of the services need to be defined manually, so we needed some sort of program that our system administration staff could use. Since all of this work is being done in Oracle, we were able to use SQL*FORMS[8] to come up with a `Service List` form (see Figure 1) to enter and update information on services. The form allows our staff to define new services, as well as search for and edit existing services. When entering a new service, once the service name has been entered, the priority and service type values are selected from lists of values. This ensures some consistency and simplifies grouping for output. There is also a space available for comments on the service; although the box on the form looks small, we currently allow comments up to 2000 characters long. An editor for this field is just a keystroke away. The form automatically takes care of recording who made the latest change and when they did this.

There are many other things we want to associate with services. Unlike that attributes listed above, which occur only once per service, the others may have more than one entry per service. To help with maintaining these, the form has four buttons along the bottom that will pop up a sub window to allow you to edit the methods, documents, contact information and dependencies on other services. The data in these sub windows are linked to the current record being displayed in the `Service_List` window. This makes it simple to step through services and see the related information. The fifth button will bring up the `Servers` form, but that is a stand alone form that does not track the current record in `Service_List`. Methods and Servers will be described in later sections.

[7]Since we are college, the overall mission is to educate students and do research. With the possible exception of our phone switch, non of our operation deals with life-safety issues.

[8]SQL*FORMS is part of the Developer/2000 package, an Oracle product that allows you develop GUI programs quickly and easily. Our current release runs on both X and Wintel platforms and the next release will generate JAVA for web applications.

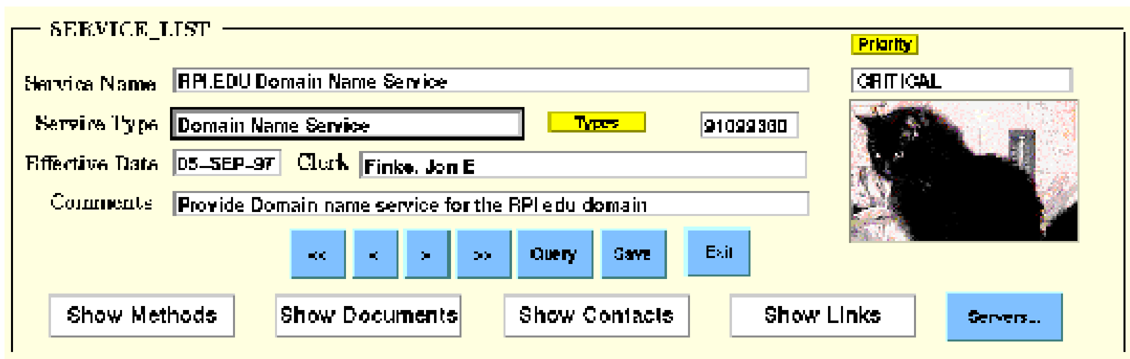


Figure 1: Service List Screen

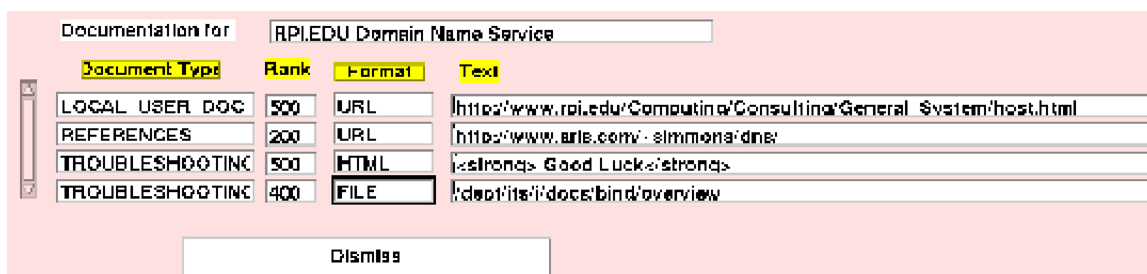


Figure 2: Service document entry/edit screen.

Service Documents

When you press the `Show Documents` button the `Service_List` form, a document entry/edit window pops up (see Figure 2). Unlike the master form which only displays a single record at a time, this form displays up to four records at a time. If there are more, you can use the scroll bars to move through them. Each document has four attributes of interest, the document type, a rank, a format and the actual text or reference.

As with service types, document types (see Table 5) are generally used for sorting and indexing on the output pages. However, it would be trivial to generate lists of services that are missing specific types of documents, such as "Troubleshooting." From a management perspective, this could go a long way to ensuring that we have at least some troubleshooting information for each of the services we offer. If we generalize our definition of "service," we can also use this to provide a general framework to hold much of our system documentation.

The next document attribute is the `Rank` field. This is a number between 1 and 999 that is used to order documents when more of one type is available for a specific service. Along with rank, we also include a document format (see Table 6). Since our primary output format for the service information is HTML, we use the document format to determine how to provide hot links when possible. This allows us to link in many of our existing documents, most of which are plain text files ("FILE"), and a few already on the web("URL").

The final field in the document screen, is the `Text` field where the actual reference or even the document text can be stored. As with the `Comments` field from the master screen, the text can be up to 2000 character in length, allowing for short in-line documents or very long URLs.

Service Contacts

One document that is usually out of date, is our list of contacts for each of our applications and services. Originally, this was a flat file that would get updated once or twice a year. An additional limit was that it only listed technical contacts for the application. In our current operation, primary support is often provided by one our two staff members, with other people providing backup support. In addition, the were often one or more folks from other departments who might provide other kinds of support. Another problem with this list, was that names were not always entered the same way, so even if you used `grep` to find your assigned areas, typos, misspellings and nicknames

Document Types	
Installation	Information on how to install the given service; generally from the vendor.
Local_Install	Local notes on installing the service here.
Local_User_Doc	Locally produced documentation that might be of interest to a user of this service.
Operational	Document for use by operations or systems administration staff; standard procedures
Overview	A document that briefly describes WHAT this service is.
References	Pointers to other relating information that may be of interest.
Troubleshooting	Information on correcting problems with this service.
Vendor_User_Doc	Documentation supplied by the vendor that might be of interest to users of the service.

Table 5: Document types.

Document Formats	
Dvi File	Document in DVI (TeX) format. (File Reference)
External	Contains a general reference to a book or other offline document.
File	Contains a pathname that points to the file, presumably in AFS space. This is used when due to location or file permissions, the doc in question is not reachable via a web server/browser.
Html	Contains HTML formatted text.
PostScript File	Document in Postscript format (File reference)
Text	Contains plain text, no HTML or other formatting information.
Url	Contains a URL to the actual information. Should be extracted as a hot link where appropriate.

Table 6: Document formats

could result in your missing some of your assignments.

To automate this, we added another database table, Service_Contact_List, which is used to record Service_Id, Service_Type, Person_Id triples. This is accessed by pressing the Show Contacts button on the bottom of the master form. The Service_Id field provides the linkage to the specific service. We have defined a few acceptable contact types (see Table 7) and the Person_Id links to specific person. Since we have the campus phone directory in the same database, when we map back to a person's name, we can also include their current phone number and email address. This also makes it trivial to generate a list of services assigned to each person. This can be especially helpful when someone leaves, we quickly can determine what areas need support.

Service Dependencies

We want to record the interdependence between services; for example, our PC printing service depends on the proper functioning of our Kerberos authentication service. If both are down, you have to first bring the authentication service. Interest in establishing the service dependency chart was greatly increased when we had a UPS failure and all of our servers were turned off at the same time (the first time this had happened with this configuration.), and there was concern that we might not be able to bring the whole set of systems back up.[9] Like the contact information, the service dependency window can be accessed by pressing the Show Links button on the bottom of the master form. Again like the contact info, the dependency information is stored in Service_Id, Dependency_Type, Service_Id triple. The second Service_Id is in fact the service id of the service that the current one depends on. One interesting offshoot of this, is that you can now have dependency chains! For example, in order for Pro/ENGINEER to run, it needs to contact the Pro/E license server. However, the Pro/E license server required that the Domain Name service be running. As a result, Pro/ENGINEER indirectly needs the DNS in order to start up. Fortunately, this dependency chain is easy to detect, so when the list of required services for Pro/ENGINEER is generated, not only are the direct dependencies (Pro/E License Server) listed, but the second order dependencies (DNS), and even higher are included automatically.

There is more than one kind of dependency though. In the previous example, Pro/ENGINEER needed the license server to start running, but once it is running, it no longer needs it. Some services, such as our SAMBA[10] service requires that the AFS service be up and running all the time. At the other end of the spectrum, we have services like our Domain Name service that require AFS if you want to make a configuration change (as the tools are stored in AFS), but normal operation and even startup are independent of AFS. To help keep track of this, we have defined some dependency types as seen in Table 8.

[9]This might seem to be rather farfetched, but we had been faced with this exact problem a number of years ago, and had to do some quick re-cabling in order to get our file servers back up after a power failure.

[10]Samba is a software package that allows wintel users to access their Unix files.

Service Contact Types	
Admin-Support	Person to contact for administrative change to service, in the case of the DNS, this would be hostmaster.
Main-User	Person(s) who have a special interest in the service and may need to be notified in case of changes or long term problems.
Primary-Support	Primary person(s) to contact in case of an outage.
Secondary-Support	Backup to the primary person(s) to contact in case of an outage.

Table 7: Contact types.

Service Dependency Types	
Operational	The service is needed for regular operation.
Periodic	The service is needed from time to time, but things can run for a while before a failure will occur.
Startup	The service is needed at startup, but not once the server is up and running.
Administration	The service is needed to make administrative or configuration changes.

Table 8: Service dependency types

A Method to this Madness

One of the objectives of this project was to automate the discovery of servers providing any given services and ideally have ways of testing that the servers are configured to provide the service and that they are actually providing that service. There are many ways to discover servers that are, or should be, providing a service. Depending on the service in question, you may be able to query some master authority such as the Internic to find your name services, or read a license file stored in the campus wide file system to find license servers. If you are on a server machine, you can look for particular configuration files, log files, or even running processes. Other potential servers can be detected by asking the clients of the service. A quick look in the `/etc/resolv.conf` should give you a list of machines that SHOULD be running name servers, or a call to `ypwhich` should tell you who should be running `nisbind`.

In addition to detecting possible servers, given a list of servers, it would be nice to be able to test those servers to see if they are in fact operational or at least configured correctly. These test methods may be very similar to the detection methods.

Attempting to write a program that can verify the configuration of all servers at a site could be an immense task. But many large tasks can be handled if you can break them up into a bunch of smaller tasks. Given that we have identified each individual service, and identified a number of different ways to try to detect or check a service, we can finally get around to writing our actual methods. Each method can a short program, a perl script, a SQL script or any other handy technique that will perform one particular test or generate a list of potential servers. These should be easy to write, and in many cases will incorporate tests developed for other systems such as *pong* [10].

Running Methods

When the time comes to run a method, you need to consider the runtime environment for the method. When you define a method (using the `Show Methods` button in the master form, you specify both the UNIX userid and the UNIX group that should be set when it runs. This helps avoid running things as root that don't need to be root. We also make a couple of other assumptions about the run time environment, at least for the global cases; that it is running on the Simon Database machine which allow access to the database without ID/Passwords pairs and that adequate AFS and DCE credentials exist to access files of interest in the AFS and DFS cells. In addition to the runtime environment, we also need to know how to make the method report back to system. Rather than requiring that each method invoke some program to report results, we instead defined several method result types (see Table 9). Depending on the method, it can simply set the return code before exiting, or output a list of values.

Global Detect Methods

A global detect method can detect a potential server from just about anywhere. In practice, this really means that it can look in the campus file system (and has appropriate file access to do so), or can query the Simon Database to extract configuration information, or query some master server.

An example of detecting a server via the file system, would be the Matlab license server. When a Matlab client starts up, it consults a license file to find the names of the license servers. In order to extract this list, just issue the command in Listing 1. This returns a list of all the license servers that Matlab clients will try to use. In this case, we didn't have to write any programs at all, just use existing system commands.

We can also write methods that contact other information servers for the list of servers. These can be trivial, such as the first of the following examples which determines our YP servers, or the second example which is slightly more complicated which we use to find our current web servers.

```
ypcat ypservers
nslookup www.rpi.edu | \
     grep 'Addresses:' | \
     cut -d: -f2
```

The second example could in fact be more complex.[11] Ideally, we would poll each of our name servers (assuming we can figure out where they are!) and ask each of them which machines they think provide our web service. But generally, all of our name servers are pretty good about giving out the same information.

All three of the above example return a list of servers providing a given service. The first two give one server per line, and the last gives a comma

[11]This example actually has at least one fatal flaw – if there is only one address, the column is labeled "Address:," but the address of the server is ALSO printed with an "Address:" label.

Method Result Types	
Client_List	This method returns a list of server/client pairs, either hostnames or IP addresses.
Exit 0 Ok	This method reports success by exiting with a return code of 0.
Server_List	This method returns a list of server names and/or IP addresses.

Table 9: Method result types.

delimited list of servers. The `find_server_state` program can handle both types of lists.

Server Config Detection Methods

Another way to check for a particular service on a machine is to the system configuration files on that machine. For example, running a PH server[12] would require the line "%define PhServer" in the `/etc/package.config` file. The next package run would install all the parts for that server. Alternately, we could attempt to install one manually by making an entry in `/etc/inetd.conf` for the server and a cron entry to regenerate the local PH database. In any event, seeing any of those three would indicate that at least an attempt was made it install a PH server. To detect that, we could execute the commands in Listing 2. If any one of those three indicators is found, we flag the host as a potential PH server (note: the three grep commands should all be one line, joined by a logical "or").

Server Activity Detection Methods

A different approach to finding a service on a machine, is to look for indications that the server process is running. This could be as direct as checking `ps` for a process of the appropriate name, or by looking for traces left behind by a server such as log files, or messages written to system log files such as `/var/adm/messages`. You would want to be careful in checking log files that you are looking at recent log files. A quick check for an Oracle server using the `ps` command could be something like:

```
% ps aux | grep -v grep \
      | grep -q ora_smon
```

[12]"ph" is phone directory server originally developed at University of Illinois at Urbana

Client Server Detection Methods

Perhaps one of the best ways to discover what servers your machines are expecting to find, is to ask the individual machines; after all, they are trying to use the services. These checks need to be done on the client machine, either directly by accessing the client file system (reading `/etc/resolv.conf`) or running daemons (`ypwhich`) or indirectly by reading saved data from `MyState` or things like that. In this case, it is important to also record which client detected a given server, as the error may be with the client. Two approaches to finding yp servers, the first runs on a client and just returns the server name, and the second runs globally and returns client/server pairs. One danger in reading saved data, is the data may be out of date. See Listing 3.

Global Server Testing

Instead of detecting servers, we can also use methods to test if a server is currently providing a service. In general, to test a service from a central point, you need to contact it, or "ping it." There has been a lot of work in the area at other sites, and we hope to start taking advantage of that soon. At this point, we have not done a lot with testing services.

Server Config Verification

Another useful check of a service, is to see if it configured correctly on the actual server. This is similar to the configuration detection methods, except you want to verify that ALL parts are in place, and not just some of them. However, in our current installation, most services are installed with package, which attempts to install all the required files and configuration changes. Any failures in this process are reported and generally corrected pretty quickly. In general, we find that services are installed and operating, or not installed at all; configuration errors are rare.

```
awk '/SERVER/ {print }'
/campus/mathworks/matlab/5.0/distrib/etc/license.dat
```

Listing 1: Extracting license list.

```
grep -q '%define PhServer' /etc/package.config || \
grep -q 'csnet-ns' /etc/inetd.conf || \
grep -q CreatePHDir /var/spool/cron/crontabs/root
```

Listing 2: Checking installation attempts

```
ypwhich
cd /dept/its/config/admin/etc/ShippedBackFiles/etc/MyState ;
grep ypmaster *.rpi.edu | cut -d: -f1,3
```

Listing 3: Finding servers.

Local Server Activity Verification

Testing services by checking for running processes is also possible, although is often a crude check in cases where a server is hung up, but still appearing in `ps`. Most of our services fall into two categories, "reliable," where they run forever, or "flaky" where we have generally put something in place to frequently check them and restart them if needed. In these cases, automatic restoration of the service seems more important than reporting the failure.

Servers

Once a service has been defined, we can assign one or more servers to that service. This relationship can be established manually by a staff member, or can be detected automatically using one or more of the methods defined in the previous section. When a server/service relationship is set up, it is assigned a status value. For entries made by staff members, this could be something like "PRODUCTION," "TEST" or "OBSOLETE." For entries discovered by the system, it would get the value "UNVERIFIED." This is actually considered an error condition in most cases, and must be changed by a staff member. Until then, it will be included in a daily error report sent to the admin staff. The objective here is to ensure that some human has considered WHY we have a new server, to help avoid dependencies on servers we do not know about.

Since any given service may be provided by more than one server, we need to be able to store multiple server records for each service. To this end, we have defined the table `Server_List`. This table records the relationship between a service and a particular server machine. This includes the status as described above, the date when the status was changed, and who changed it, along with their comments. It also records an optional review date with comments on that. It also will record automatic detection information from the methods described above. For each method (global, server and client detect), it records the date first detected, the date cleared (if the last check failed), and the date the last check was done. In addition, for client detection, the `System_Id` of the client is also recorded. Our original intention was to have all the user interface for this

SERVER_LIST

Service Name: YP Server for RCS
Status: UNVERIFIED <-- Warning!
Server Name: asher.its.rpi.edu
Verified by:
Verify Date
Verify Clea
Review Date
Review Comments

Global Detection
Current Status: DETECTED
Last Checked: 07-SEP-97
Date Set: 07-SEP-97
Date Cleard

Server Detection
Current Status
Last Checked
Date Set
Date Cleard

Client Detection
Current Status
Last Checked
Date Set
Date Cleard

Include Clients
Exclude Clients

<< < > >> Query Save Exit

Figure 3: Server Entry/Edit Screen

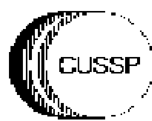
project be using a web server, but some problems with getting that installed in a timely manner, resulted in us developing a SQL*FORM application, `Server_List`, to allow our staff to update the status values for each server record (see Figure 3). The upper half of the form has fields that can be updated by staff members, and the lower half has the three sets of detection results. One of the side effects of the client detect process, is that we get a record for each client of a particular server/service pair. Rather than display all of these records on this form, there is a switch at the bottom of the form that allows the user to include or exclude the client detect records. However, since client detection information may be important, when we record client detection information (date set, etc), we also record the status and date checked and date set in the "master" record (the record that has the global and server detect information, as well as the manual information. We don't expect our staff to go in and validate each client record individually. In order to help with the validation process (where a staff member changes the status from "Unverified" to something else), we bring up a warning message and arrow, alerting the user to the questionable status.

Weaving Everything Together

With the information described above, we are able to generate the server lists requested by the NOC staff. Not only the list of machines providing each service (which is handy when the call comes in that says the XXX service is broken), we can easily generate the list of services provided by any given machine (which is handy when we learn that a particular machine has gone down). These lists are generated automatically and sent to the NOC as needed. Although hardcopy lists may seem old fashioned, they are nice to have when you are reading them by flashlight, attempting to determine what order to power up things once the UPS is repaired.

Server (System) Pages

We wanted more than that though. By generating HTML pages, we are able to include a lot more information, not only for services, but for the servers themselves. We are already collecting all of the `/etc/MyState` information for all machines, so this provides the basis for servers pages, in fact, all we need to do is add the services information as part of the MyState page generation and we have a pretty detailed picture of any given machine. Figure 4 is partial view of one of the server (actually, system) pages. This includes some system information (machine type, location, serial number) drawn from the Hostmaster database, the server information, listing two services that this machine provides, and continues on to DNS information, and the MyState information, allowing our system administrators to get a pretty complete



netserv1.its.rpi.edu

Comments on the format and layout of these pages, as well as ideas for improvements, should be directed to finkej@rpi.edu Page generated at:Sun Sep 7 16:02:50 1997

System Info

- Location: VCC Machine Room
- Hardware: Sun SPARC4
- SN: 605F00E7

Server Info

Priority: CRITICAL

- RPI.EDU Domain Name Service Status: PRODUCTION

Priority: HIGH

- Print Server Status: PRODUCTION

DNS info for netserv1.its.rpi.edu

IP Address	128.113.1.5

mystate

Figure 4: Server (System) HTML page for Netserv1

picture of the machine. In order to aid navigation, we include a lot of links. We have both generic links back to the appropriate index pages (if you select "Critical," you will go to the index page of critical priority services.) and specific links such as "RPI.EDU Domain Name Server," which takes you to that page (see Figure 5).

Service Pages

This page contains a lot of information about this service. As you can see in Figure 5, it starts with some general information about the service, a short description, followed by the priority, service type and contact information. The contact information automatically includes the person's phone number and email address if available. The email address is even set up as a `mailto:` URL. The next section lists which machines are providing that service, along with the status, and who verified that status and when. Clicking on the machine name will bring you to the page for that machine (like Figure 4). The next section lists the services that THIS service requires, and what type of dependency, along with any comments on that dependency. Following that, we include whatever documentation we have available for that service, ordered by type; again, we have provided hypertext links where possible. Finally, we have a list of services that rely on this service. As with the server(system) pages, we try to provide all the relevant information about a service in one neat package, and provide quick navigation to the related topics.

Other Pages

We also generate a number of other types of pages. We can generate document pages, sorted both by document type (troubleshooting, operation, etc),

General Information

Provide Domain name service for the RPI edu domain

- Priority: CRITICAL.
- Classification: Domain Name Service
- Primary Contacts: David K Hudson hudsod@rpi.edu 8836
- Secondary Contacts: Jon Finke jfinke@rpi.edu 8185

Servers providing this service

netserv1.its.rpi.edu
- Current Status is: PRODUCTION
- Verified by Jon Finke jfinke@rpi.edu 8185 on 07-SEP-97

netserv2.its.rpi.edu, vccprntserv.its.rpi.edu
- Current Status is: PRODUCTION
- Verified by Jon Finke jfinke@rpi.edu 8185 on 07-SEP-97

Services required by this service.

- Direct dependence servers.
  Simon – ADMINISTER
  Hostmaster Data is stored in Simon.

Documentation

REFERENCES
http://www.aris.com/~simmons/dns/
TROUBLESHOOTING
file:/dept/its/i/docs/bind/overview
LOCAL_USER_DOC
http://www.rpi.edu/Computing/Consulting/General_System/host.html

Services that rely on this service.

- Direct dependence on this service.
  Sun Compiler License Server – OPERATIONAL
  Needs to look up host names
  Simon SqlServer – OPERATIONAL

Figure 5: Service HTML page for RPI.EDU Domain Name Service

and by service type (Unix Command, License Server, etc). We also generate Contact pages, so you can easily check what services are assigned to a given individual. Using the relational database to store the information gives us many options in organizing and displaying the data; hopefully we can be all things to all people.

Future Directions

Our immediate challenge is to populate the database with the rest of our services. I expect we will expand the definition of ‘‘service’’ to include all of the applications we support, if for no other reason, than to simplifying tracking the contact information and documentation for each application.

We will also be writing programs and SQL scripts to provide cross checking for errors and inconsistencies. These will range from missing information such as contact person and troubleshooting documentation, to differences found by different types of detection. If a server is detected with one method and not with another, there is a sure sign that something needs to be investigated.

Once we get the latest version of the Oracle Web server installed, I expect to make the forms available as web pages. We will also be investigating generating at least some of the pages in real time.

We are also considering dumping the entire service/server web tree to a CD which would be available for use on a stand alone machine (handy for when you have major problems), or even for the on call person to take home to run on their home machine. While this has the drawback of the CDs getting out of date, they still may be handy as a reference, especially when network access is slow or missing altogether.

References and Availability

All source code for the Simon system is available for anonymous FTP. See ftp://ftp.rpi.edu/pub/its-release/simon/README.simon for details. In addition, all of the Oracle table definitions are available at http://www.rpi.edu/campus/rpi/simon/misc/SIM2/SIMON-index.html .

Given that much of the output of this system is web based, it would be very nice to be able to make all the pages generally available. If nothing else, it would make it a lot easier for our own staff to access it. However, before we can release the pages, we need to assure ourselves that releasing it will not result in the unplanned release of confidential information or reducing site security. If there is sufficient demand, I can release a subset of the pages for demonstration purposes with sensitive material removed.

Author Information

Jon Finke graduated from Rensselaer in 1983, where he had provided microcomputer support and communications programming, with a BS-ECSE. He continued as a full time staff member in the computer center. From PC communications, he moved into mainframe communications and networking, and then on to Unix support, including a stint in the Nysernet Network Information Center. A charter member of the Workstation Support Group he took over printing development and support and later inherited the Simon project, which has been his primary focus for the past six years. He is currently a Senior Systems Programmer in the Server Support Services department at Rensselaer, where he continues integrating Simon with the rest of the Institute information systems, and also deals with information security concerns.

Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him electronically at finkej@rpi.edu. Find out more via http://www.rpi.edu/~finkej.

Bibliography

[1] Paul Anderson, ‘‘Towards a high-level machine configuration system,’’ *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pages 19-23, USENIX, San Diego, CA. September, 1994.

[2] Jon Finke, ‘‘Automated userid management,’’ *Proceedings of Community Workshop ’92*, Papers 3-5, Rensselear Polytechnic Institute, Troy, NY, June, 1992.

[3] Jon Finke, ‘‘Relational database + automated sysadmin = simon,’ Invited Talk for SUG-East 93, Sun Users Group, Boston, MA, July, 1993.

[4] Jon Finke, ‘‘Automating printing configuration,’’ *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pp 175-184. USENIX, San Diego, CA, September, 1994.

[5] Jon Finke, ‘‘Sql_2_html: Automatic generation of html database schemas,’’ *Ninth Systems Administration Conference (LISA ’95)*, pp 133-138. USENIX, Monterey, CA, September, 1995.

[6] Jon Finke, ‘‘Institute white pages as a system administration problem,’’ *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pp 233-240.USENIX, Chicago, IL, October, 1996.

[7] Jon Finke, ‘‘Monitoring application use with license server logs’’ *The Eleventh Systems Administration Conference (LISA 97) Proceedings*, USENIX, San Diego, CA, October, 1997.

[8] Stephen E. Hansen and E. Todd Atkins, ‘‘Automated system monitoring and notification with swatch,’’ *USENIX Systems Administration (LISA VII) Conference Proceedings*, pp 145-156., USENIX, Monterey, CA, November, 1993.

[9] Dr. Magnus Harlander, ‘‘Central system administration in a heterogeneous Unix environment: Genuadmin,’’ *USENIX Systems Administration*

(LISA VIII) Conference Proceedings, pp 1-8, USENIX, San Diego, CA, September, 1994.

[10] Helen E. Harrison, Mike C. Mitchell, and Michael E Shaddock, "Pong: A flexible network services monitoring system," *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pp 167-171, USENIX, San Diego, CA, September, 1994.

[11] Todd Miller, Christopher Stirlen, and Evi Nemeth, "satool – a system administrator's cockpit, an implementation," *USENIX Systems Administration (LISA VII) Conference Proceedings*, pp 119-130, USENIX, Monterey, CA, November, 1993.

[12] John P. Rouillard and Richard B Martin, "Config: A mechanism for installing and tracking system configurations," *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pp 9-18, USENIX, San Diego, CA, September, 1994.

[13] J Schonwalder and H Langendorfer, "How to keep track of your network configuration," *USENIX Systems Administration (LISA VII) Conference Proceedings*, pp 189-193, USENIX, Monterey, CA, November, 1993.

[14] Rex Walters, "Tracking hardware configurations in a heterogeneous network with syslogd," *Ninth Systems Administration Conference (LISA '95*, pp 241-246, USENIX, Monterey, CA, September, 1995.

[15] David C. M. Wood, Sean S. Coleman, and Michael F. Schwartz, "Fremont: A system for discovering network characteristics and problems," *USENIX Technical Conference Proceedings*, pp 335-347, USENIX, San Diego, CA, January, 1993.