An Improved Approach for Generating Configuration Files from a Database

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

Much of our site configuration information is stored in a relational database, which means we need to extract this information in the appropriate format for servers and daemons. In the past we have done this with lots of little custom C programs and scripts. We have recently changed to a new approach of generating the files within the database itself using PL/SQL packages, and then using a generic file extraction program to handle the details of putting ASCII characters into Unix (or other) file systems. This has allowed us to reduce development time of programs to generate new file types, and greatly simplified supporting multiple platforms.

Introduction

At Rensselaer, we maintain a lot of our system and site configuration information in a database, using a package we developed call Simon [5, 7]. We are not alone in these efforts. I have talked with system administrators at many other sites who are working on similar projects, including the University of Alberta, Simon Frasier University, SUNY Albany, and the University of Connecticut. One of the early projects of this type was at MIT's project Athena and their Moira [15] package. This is not limited to educational sites, I have also spoken with folks at Cisco Systems and Collective Technologies about similar projects. A quick glance at the last few LISA proceedings show a number of similar projects including Accountworks [3], NFS Configuration Management [4], Unix Host Administration [16], Aurora [12], Exu [14] (Ok, they talked about doing it) and others.

In practice, a number of configuration files need to be extracted from the database. These files range from host specific files such as /etc/printcap [8] to platform specific files like /etc/group to site wide configuration files like the resource record files for the domain name system [6], to HTML and LDIF files for the University telephone directory [10]. Traditionally, each file is generated by a file specific C program, or in some cases, a SQL*PLUS script. Depending on the specific requirements of the file, the program might deal with version control (generate only when information changed), trigger post processing, and have to run under different operating systems and environments.

The general pattern when a new file type was needed was to grab one of these existing programs, change part of it and recompile it (possibly on several different versions of Unix). As the number of different versions of unix grew, and we started working with non unix platforms, maintaining all the different versions of these programs and getting them distributed to the correct machines got to be more and more of a hassle. In addition, these programs varied widely. They used different methods of connecting to the database, supported different concepts of version control, or had additional control options, so you would at times find yourself having built on the wrong ''base,'' and have to do even more work.

As our systems became more distributed and specialized, maintaining and developing these custom programs became more challenging, as some of the target systems did not share file systems with the development systems, tools were not available, etc. As a result, this made what should have been trivial changes in a file format (like adding a newline between records!) into an hour or more of work. For example, when we wanted to build DHCP configuration files, we wanted to build them on our DHCP servers. Unlike our existing DNS servers (Solaris) which mounted our AFS file system, the DHCP servers (OpenBSD) do not even have an AFS client available. Our existing practice of storing the file generation programs or even the configuration files in AFS space was not going to work.

Building Files in the Database

Our new approach to generating files is to generate the file in the database itself then simply write a short program to dump the ''file'' out of the database and into the file system of the target machine. This allows all of the file generation code to be stored and maintained in the central database, using a consistent set of tools. In addition, the program to extract the file can be generic, and once built for a particular operating environment, could be used for any of the files we might generate for that system.

Our solution to this (see Figure 1), consists of three layers. The first layer is a generic program (Generate_File) that runs on the target system that can connect to the database and write a file on the target system. This calls on the second layer: a PL/SQL package

[13][1] File_Gen which runs on the database server. This module provides the sole interface to the database for the file generation program, and handles all of the access control for each of the desired files we might generate. This in turn calls the bottom layer for the file specific packages, such as Gen_RR_File, Gen_DHCP_Config and Gen_Ldif that handle the details of extracting the data and formatting it for a given type of file.

This has proven to be a very powerful technique, allowing us to keep the data extraction and formatting of the file in the database, while doing the actual file generation and version control management on our target systems. We can also use the Generate_File program for any number of different files, which means we only need to compile it once for any given platform. Adding a new file type involves writing a PL/SQL routine to format the data and registering it with the file generation package.

General Operation

To generate a file, the Generate_File program is executed with the file target as a parameter. The program connects to the database and calls the File_Gen.Get_Attr[2] stored procedure, passing it the file target. File_Gen.Get_Attr does some access checking, possibly calls a target specific procedure, and returns a file name, and optionally, some version information. The Generate_File program then opens the file and calls File_Gen.Get_Data routine, which calls a target specific package, and passes the result back up and the line is written to the file until a Null is returned. The file is then closed, and moved into place.

For many of the files we generate, (/etc/printcap, /etc/passwd, /etc/group, etc.), we just want a single file produced. In other cases, we want to generate more than one file at a time. For example, when we are generating the resource record files for Bind, we want to regenerate all files that have changed. A single invocation of the program may create a lot of files. To handle multiple files, the Generate_File program then calls File_Gen.Get_Attr to get a new file name and the cycle is repeated until File_Gen.Get_Attr indicates that there are no more files.

Generate_File

In our current implementation, the host program is written in C, using the Oracle C pre compiler (PRO*C). However, we are not limited to using this interface. We are planning on writing one in JAVA, and one could be written in PERL or any other language that can access the database. What is more, there is no problem in mixing and matching, all interfaces can generate all file types.

In our implementation, the program does a few more housekeeping details for us. When we open a file for writing, we actually open the file *filename*.new. Everything is written there, and after it successfully closes the file, we move *filename* to *filename*.old and then move *filename*.new to *filename*. This gives us a little bit of protection from full file systems and network interruptions. It is possible for other applications to have the target file open. The script calling Generate_File program needs to be written to take the appropriate action such as signaling the other process.

The program has no special privileges. It is up to the caller to ensure that it is run in the proper directory with the proper access to the file system. In appendix 1, we have the script[3] used to generate our telephone directory web pages. It is run as root via cron once a day. After setting some variables the script attempts to authenticate to AFS. The hosts.klog is a script we use to authenticate root processes on a system to AFS. Next we invoke a simple "locking" script[4] to ensure that we are the only copy running, cd to the target directory and generate the files. The program is invoked with the "-nvr" (short for Needs Vos

[1]PL/SQL Packages are a collection of functions, procedures, cursors and variables. These can be both public and private. Access to the public procedures can be granted to oracle users or roles. Once accessed in a session, a package maintains state between calls.

[2]Procedures in packages are referenced as *package name.object name*. This is the Get_Attr procedure in the File_Gen package.

[3]The /usr/vice/etc/pagsh is an AFSism, used to help control AFS authentication.

[4]It is crude and depends on a file existing or not. But it has not failed us in six years of operation.

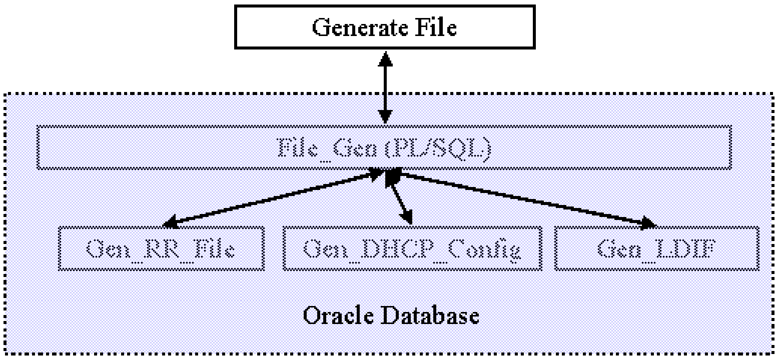

Figure 1: Service model.

Release) option. If we regenerate a file, we write that file name to the file specified by -nvr. After we are done with all the file generation, we check for the file specified by "-nvr" and if it exists, request a vos release.[5] Instead of a vos release, other post processing could be trigger such as a "make yp," or whatever is needed.

Our version of the Generate_File program can connect directly to the database (if it is on the database machine), or can connect via SQL*NET to a remote database. In this case it can read an Oracle id and password from the command line or from a file. This works well in our environment.

File Version Control

One of the bits of information the program gets from the File_Gen.Get_Attr call is file version information. For example, we have around 170 different sub domains at our site. When we regenerate the resource record files, we don't want to build the entire set each time, but rather, we want to just update the files that have changes.

```
;
; RR file for rpi.edu
;
; Simon Database Version:26473884
;        Generation Date: 23-MAY-2000 11:07
...
```

Figure 2: Version Number in file.

To this end, we maintain a version number for each file. In many cases, this is written in the first 10 lines of the file with a particular flag string (see Figure 2) where we scan for the string "Simon Database Version" in the header comments of the RR file. For files that do not support comment lines, we write a parallel file with the string .vers. prepended to the file name; ie the version file for a file called passwd would be .vers.passwd.

[5]The "vos release" command is an AFSism that is used to reclone read only, replicated disk volumes.

The File_Gen.Get_Attr routine returns, along with the file name to use, an optional version number for that file and an optional flag string. If the flag string is present, it will scan for it in the target file. If there is no flag string, then it will look for the parallel version file. The program then compares the database version number with the file version, and if they don't match, the file is regenerated. The Generate_File program has a -force option to make it ignore the version numbers and generate all files.

Post Processing Control

With the version control enabled, it is possible to run the program for a given target and not write any files. In this case, no post processing would be needed. To assist in this, another option lets you specify a flag file. If a file is written, its name will be written to the flag file. Post processing code can check to see if the flag file exists and take appropriate action. We use this to automatically replicate files in AFS after generation. This is shown in appendix 1. Another example might be generating the /etc/inetd.conf file. If you updated the file, you would want to restart inetd. If there was no change to the file, there would be no need to signal the inetd process. This feature was used in some of our older programs and proved useful so we kept it as we moved to this new approach.

File_Gen Package

The core of this entire project is the File_Gen package, which acts as the gatekeeper. It provides access control and handling many of the interface details to the actual file generation procedures.

Get_Attr

When the Get_Attr procedure is called, you supply the name of the desired target (and an optional parameter; see Figure 3). It first checks to see if the current Oracle user is allowed to generate that file (access the data). If they are, it returns a file name, some control flags and some version control information. If there is some problem (unknown target, no

Name	Type		Description
Target	Varchar2	In	Name of file set to be generated.
ErrMsg	Varchar2	Out	If not null, display the message and terminate.
Filename	Varchar2	Out	Name of the file to be generated. When null, there are no more files to be generated.
DBMSOut	Varchar2	Out	If "Y", then DBMS_Output routines may be used.
Direction	Varchar2	Out	Direction of transfer, "GET" a file from the database, or "PUT" a file into the database.
Version_Number	Number	Out	The current version number of the file to be generated. If null, don't check version before generating file.
Version_Str	Varchar2	Out	If not null, scan the start of the file for this string and the version number.
Par3	Varchar2	In	A parameter passed from the command line of the caller to the file generation package. Usage defined by the individual package.

Figure 3: Get_Attr parameters.

access), the error message is returned instead. Get_Attr is the key routine which drives the entire generation process. Once a file is generated (or skipped due to version numbers matching), this routine should be called again for the next file.

In our initial implementation of the File_Gen package, we used a case statement[6] This required changing and recompiling the package every time a new file target was added. This also made File_Gen dependent on all of the file-specific packages. Fortunately, Oracle provides a dynamic SQL package called DBMS_SQL [1] that allows a PL/SQL procedure to parse and execute an arbitrary PL/SQL routine. With these details worked out, adding a new file target was reduced to making an entry in the Generate_File_Types table (and writing the file-specific package of course); see Figure 4.

While the general file generation model has the File_Gen.Get_Attr called until there are no more files to be generated, many of our file targets only produce a single or fixed set of files. Having to write a Get_Attr_Rtn that keeps track of the current state (StartFile or EndFile), returns the file name, the version string, and so on was a bit of a hassle. So, if the Generate_File_Types.Get_Attr_Rtn is null, the File_Gen. Get_Attr will "fake it," using the values supplied in the table. This makes it very easy to add simple file set targets. By using different sequence numbers, multiple files can be generated without having to write a file specific Get_Attr_Rtn. We use this to generate both our people and department LDIF files for our LDAP server.

[6]Actually, PL/SQL does not have a case statement, so it was a large if/elsif/elsif statement.

Set_Program_Version

Set_Program_Version is used to make host information (current user, hostname, program version) available to the generation routines. We frequently include version information and other diagnostic info in the header of a file (see Figure 5). This is called at the start of the run, before any calls to Get_Attr or other routines.

Get_Data

Once a file is opened, the Get_Data routine is called and the result is written to the output file. If the DBMS_OUTPUT flag was set by Get_Attr, then the Dbms_Output buffers should also be written to the output file. This cycle repeats until Get_Data returns null. DBMS_Output [2] is a package supplied by Oracle that will buffer text provided in calls to DBMS_Output.Put_Line until the application retrieves the text via calls to DBMS_Output.Get_Line.

Put_Data

When we are "PUT"ting a file into the database, we call this routine for each line of the file. When we

Name	Type	Size	Description
Target	Varchar2	32	The name of the target file set.
Seq_Number	Number		Used to order files within a file set.
RoleName	Varchar2	255	Oracle role required to process this file.
Get_Attr_Rtn	Varchar2	65	Name of the file set specific get attribute routine.
Get_Data_Rtn	Varchar2	65	Name of the stored procedure to be called to return a line of the file.
Put_Data_Rtn	Varchar2	65	Name of the stored procedure to be called to put a line of the file into the database.
End_Data_Rtn	Varchar2	65	Name of the stored procedure to be called when all lines of the file have been read.
Def_Filename	Varchar2	65	Default file name to be used if Get_Attr_Rtn is null.
DBMSOUT	Varchar2	1	When "Y", indicates that this file target may generate output via the DBMS_OUTPUT package.
Direction	Varchar2	4	Default direction to be used if Get_Attr_Rtn is null.
File_Version_Str	Varchar2	64	File version string to be used if Get_Attr_Rtn is null.
File_Version_Number	Varchar2	32	File version number to be used if Get_Attr_Rtn is null.
Package_Version	Varchar2	128	A version number (if any) for the package being used to generate the file. We manage our stored procedures with RCS and use the "Id" value here.
Package_Header	Varchar2	255	A file reference to the source file used to create this package. We use the "$Header$" value here.
Create_Date	Date		The date when this file target was first made available.
Update_Date	Date		The date when this file target was most recently changed.
Comments	Varchar2	255	A short description of this file target.

Figure 4: Generate_File_Types table.

have reached the end of the file, we call the End_Data routine which signals the database to do any post processing required on the data that was just loaded. Both Put_Data and End_Data have an ErrMsg parameter. If this is not null, the message should be displayed and the file load terminated. This allows the underlying file load packages to signal fatal exceptions.

Generate_File_Internal

There is a second PL/SQL package called Generate_File_Internal. From a strictly programming standpoint, this should be part of the Generate_File package. However, the Generate_File package is permitted to the public, thus all public procedures are available for anyone to use. Since the Generate_File_Internal defines some procedures to define and change file targets, it would not be good to let just anyone access these. Rather than put specific access control inside these routines, we kept them here and use Oracle access control to limit who can define new targets.

Add_Target_Complex

This entry point is used to add a complex file target, that is, one that provides a Get_Attr_Rtn to supply the file name, version info, etc. By using a procedure to add entries to the Generate_File_Types table, we are able to do a bit of sanity checking and maintain the create_date field. If there is an existing target with the same sequence number, it is replaced with this one. You will note that the ordering of the parameters (see Figure 6 for a list) is different from the earlier procedures and table definitions. Some of the parameters are optional, so these are left to the end of the parameter list.

Add_Target_Simple

This routine is used to define simple targets. Ironically, this particular call is more complex than the previous call. Of course, since we are off-loading more of the work on the Generate_File package, the trade-off is ok. Like the previous routine, some of the parameters (see Figure 7) have been moved around a bit, and some of the later ones are optional.

There is also a Add_Target_Put which is similar to the Add_Target_Simple, except it defines a simple ‘‘put’’ entry rather than a simple ‘‘get’’ entry.

Target=LIST

In order to test these routines, as well as add some handy tools, this package also defines a ‘‘LIST’’ target that lists all the names and descriptions of all define targets. This can be handy when you need to generate a file and can’t remember the target name. (This was a very annoying lack in the previous

```
# Simon Database Version:26512881
#        Generation Date: 30-MAY-2000 18:12
#    Generation Hostname: vcmr-42.server.rpi.edu
#           Program Name:../Generate_File
#              User Name: finkej
#          Generate_File: $Id: Generate_File.pc,v 1.3
#                         2000/05/18 23:34:20 finkej Exp finkej $
#    Generate_DHCP_Files: $Id: Create_Package_Generate_DHCP_Files.sql,v
#                              1.3 2000/05/11 23:10:17 finkej$
#         Host_Dns_Maint: $Id: Create_Package_Host_DNS_Maint.sql,v
#                              1.2 2000/05/11 22:50:17 finkej Exp finkej $
#             Host_Maint: $Id: Create_Package_Host_Maint.sql,v
#                              1.2 2000/04/28 20:07:00 finkej Exp finkej $
```

Figure 5: Version information.

Name	Type	Description
Target	Varchar2	The target file set name.
RoleName	Varchar2	The role required. Note, this is checked only for the first entry for a given target.
Pack_Version	Varchar2	The version number of the package (“Id”).
Pack_Path	Varchar2	The path name of the source file of the package.
Comments	Varchar2	Comments on the package.
Get_Attr_Rtn	Varchar2	The routine name. Since this is a complex target, this must be provided.
Get_Data_Rtn	Varchar2	The get data routine name. Technically, this could be an optional parameter, but so far all complex targets have one.
Seq_Number	Number	An optional sequence number. If not specified, it will default to 10.
Put_Data_Rtn	Varchar2	(Optional) A put data routine name.
End_Data_Rtn	Varchar2	(Optional) An end data routine name.

Figure 6: Add_Target_Complex parameters

version.) There is also a version of this that lists packages and the location of the source for each of them.

File Specific Packages

Once the upper two layers are installed, the rest of the work takes place at the file specific package level. Up to this point, we did not really know nor care what files we were generating or reading. At present, we use this system to generate resource record files for BIND, host files, DHCP configuration files, /etc/passwd, /etc/group, white pages directories in ph,ldif and CSV formats and some web pages. We also use it to load in accounting records from our backup system and directory information from a remote campus. This list will be growing over time.

A rather simple example of this, is the package to generate the /etc/passwd file. A source file to create this package is available in appendix 2. Our practice is to have a source file that we feed into SQL*PLUS, the Oracle SQL interface. To facilitate test versions, we use a "define name=" statement to set the package name. This is substituted in the appropriate places in the file when executed in SQL*PLUS. During development, we can change the name and not risk wiping out the production version.

The source file has three sections: defining the package specification; the package body and finally, registering the new target with the Generate_File routines.

The prompt statement prints the message. This proves helpful when there are errors, as you get a better idea which section had the failure. The first section, defining the package specification (Create or Replace Package &name) is pretty simple. It includes an RCS "$Header$" and the definition of the Get_Data routine. The parameters P1 and P2 are optional parameters that are passed in from the command line. Their usage is based on specific package. They might be used to enable debugging or in some way control details of the file generation.

The second section, defining the package body, (Create or Replace Package Body &name) has the actual PL/SQL code that is executed to produce the password file. The cursor sets up the query to extract the data. This happens on the first call to Get_Data when the cursor is opened. Each fetch statement brings the next row of data into the "R" variable. This record is automatically defined based on the columns in the select statement. The quoted strings in the select statement are column aliases. Finally, when the end of the data is reached, a null is returned rather than a PW file entry and the cursor is closed.

The third section is used to register this file type with the Generate_File routines. Since this is a simple case, we can use the Add_Target_Simple routine, define the target name, skip access checks, include the version info and some comments and set the output file name to be passwd_demo. Finally, we provide the name of the get data routine. We do not specify a sequence number, so this will take the default of 10, and replace any existing definition for etc-passwd_demo.

Conclusions and Futures

Execution Time

In some casual timing tests we found mixed results. For a simple extraction, where all data is in a single table, the custom program appears to be slightly faster than the PL/SQL approach. However, if the file is complex, requiring data from many tables, the PL/SQL approach is faster. This is in part because more of the work taking place on the database server, thus reducing the I/O between the server to the application.

There is also some overhead from using the Dynamic SQL codes rather than hard coding the switch statement. Some sample runs show about a 15% increase in run time by using the dynamic sql routines. However, given the reduction in development time, this seems worth the tradeoff. If this

Name	Type	Description
Target	Varchar2	The target file set name.
RoleName	Varchar2	The role required. Note, this is checked only for the first entry for a given target.
Pack_Version	Varchar2	The version number of the package ("Id").
Pack_Path	Varchar2	The path name of the source file of the package.
Comments	Varchar2	Comments on the package.
Filename	Varchar2	The name of the file we will generate.
Get_Data_Rtn	Varchar2	The get data routine name.
Seq_Number	Number	An optional sequence number. If not specified, it will default to 10.
DbmsOut	Varchar2	The value for the DBMS Output flag. Should be "Y" or "N".
File_Vstr	Varchar2	An optional version string to be used with version control.
File_Vnum	Varchar2	An optional version number.

Figure 7: Add_Target_Simple

difference is a problem, it would be possible to write a program to generate the source file.

Development and Deployment Time

The development time to add a new file to be generated to the system has been significantly reduced, in part because all of the changes are now isolated to the central database, rather than being distributed across many remote servers. Once the Generate_File program is installed on a machine, virtually no work needs to be done on that machine.

The PL/SQL language is well integrated in the Oracle environment, and many of my PL/SQL programs are noticeably shorter than the PRO*C programs they replaced. For example, the original PRO*C version of the program to generate the faculty/staff phone directory was 2639 lines of C/PRO*C code versus the 1588 lines of PL/SQL that replaced it. I feel that the PL/SQL was faster to write as well as being shorter.

Another factor that reduced development time, is that we often are doing a web component for the front end of these files. This results in a set of PL/SQL routines to access the tables, so half the work is done by the time we need to write a file generation routine.

Another advantage is that we are able to use display code used by the Oracle web server when we are generating static HTML pages. We are using this convergence to dynamically preview static web pages which reduces the development time with those as well.

Handy Tricks

To help validate post processing, when we generate our RR files for bind, we include an entry at the end that includes a "wc"[7] for the entire file in one of the fields. This allows post processing scripts to quickly determine if they are working with a complete passwd file, and not one that got truncated. Adding support for this to the system could be handy.

For our generated password files, we include a special entry at the end (show here folded):

```
TIMESTMP:##TIMESTMP:397:4000:
    13834 37930 1236688
    2000-09-20-13-01-07:/tmp:/bin/true
```

The GECOS field has a "wc" value, as well as the time and date of generation. This is handy when your password file distribution processes hit snags.

New Platforms

We are in the process of developing a JAVA version of this program to run on Linux systems. We don't have the Oracle development tools licensed for this platform, and our network servers are moving away from our centrally administrated AIX machines to this platform. This is actually being driven by our security team wanting to process the DHCP lease file[8].

[7]Unix wordcount; returns lines, words and bytes in a file.

[8]It is somewhat ironic that this paper started with describing how we generate DHCP configuration files and has come back around to the other side of that project.

New Directions

Since we already support "GET" and "PUT" operations, what else could we want? Well, rather than getting data from a file, how about getting data from a program. We currently have a custom C program that runs lscfg on a pipe, captures the output, digests it a bit and saves it into the database. We could replace it with a file set that does the same thing. What is more, if we want to add additional program runs, we can do that by changing the package stored in the central database and we no longer have to worry about getting new versions of the custom programs out to our clients. I expect that we will have our client machines running Generate_File with the file target of "Self Exam" periodically to bring information on the config back to our central management system[11]. One limitation of this is that it only works with text files. Trying to pick up binary files such as /var/adm/wtmp might still require custom programs such as the one we wrote to assist in doing demographic analysis of workstation use [9].

Along with running programs and capturing their output, we could run the other way and generate text and pass it to a program. This could be used as a quick and dirty interface to lpr or mail. One of the parameters available to the file generation packages is the platform (Hardware, OS) so it could presumably make reasonable assumptions as to what programs it can call, and where to find them. This support could also be used to initiate post processing, although that might better be handled in the calling script.

Internal Changes

I expect that we will merge the Generate_File_Internal package back in to the Generate_File package and build the access checking into the add target routines.

References and Availability

All source code for the Simon system is available on the web. See http://www.rpi.edu/campus/rpi/simon/README.simon for details. In addition, all of the Oracle table definitions as well as PL/SQL package source are available at http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.html .

While the specific file generation packages are Rensselaer specific, both the host programs and the File_Generate should be pretty generic and could be used at other sites with no modifications.

Acknowledgments

I would like to thank Alan Powell and Jackie Stampalia of Server Support Services, Rensselaer Polytechnic Institute for their willingness to read and comment on many drafts of this paper. Special thanks go out to Remy Evard of the Argonne National Laboratory who helped edit the final version of the paper.

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 nine 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. When not playing with computers, you can often find him building or renovating houses for Habitat for Humanity, as well as his own home. 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.

References

[1] Eric Armstrong, Steve Bobrowski, John Frazzini, Brian Linden, and Maria Pratt, *Oracle 7 Server Application Developer's Guide*, chapter 11, pages 1-22. Oracle Corporation, Dec 1992.

[2] Eric Armstrong, Steve Bobrowski, John Frazzini, Brian Linden, and Maria Pratt, *Oracle 7 Server Application Developer's Guide*, chapter 6, pages 23-28. Oracle Corporation, Dec 1992.

[3] Bob Arnold, "Accountworks: User Create Account on SQL, Notes, NT and Unix," *The Twelfth Systems Administration Conference (LISA 98) Proceedings*, pages 49-61, Sybase Inc, USENIX, December 1998, Boston, MA.

[4] Fabio Q. B. da Silva, Juliana Silva da Cunha, Danielle M. Franklin, Luciana S. Varejao, and Rosalie Belian, "An nfs configuration management system and its underlying object-oriented model," *The Twelfth Systems Administration Conference (LISA 98) Proceedings*, pages 121-130, Federal University of Pernambuco, USENIX, December 1998, Boston, MA.

[5] Jon Finke, "Automated userid management," *Proceedings of Community Workshop '92*, Troy, NY, June 1992, Paper 3-5.

[6] Jon Finke, "Simon system management: Hostmaster and beyond," *Proceedings of Community Workshop '92*, Troy, NY, June 1992, Paper 3-7.

[7] Jon Finke, "Relational Database + Automated Sysadmin = Simon," Invited Talk, July 1993, Sun Users Group – East Conference, Boston, MA.

[8] Jon Finke, "Automating printing configuration," *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pages 175-184, USENIX, September 1994, San Diego, CA.

[9] Jon Finke, "Monitoring Usage of Workstations With a Relational Database," In *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pages 149-158, USENIX, September 1994, San Diego, CA.

[10] Jon Finke, "Institute White Pages as a System Administration Problem," *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pages 233-240, USENIX, October 1996, Chicago, IL.

[11] Jon Finke, "Automation of site configuration management," *The Eleventh Systems Administration Conference (LISA 97) Proceedings*, USENIX, October 1997, San Diego, CA.

[12] Xev Gittler, W. Phillip Moore, and J. Rambhasker, "Morgan Stanley's Aurora System: Designing a Next Generation Global Production Unix Environment," *Ninth Systems Administration Conference (LISA '95)*, pages 47-58, Morgan Stanley, USENIX, September 1995, Monterey, CA.

[13] Tom Portfolio, *PL/SQL Release 8 User's Guide and Reference*, Oracle Corporation, December 1997, Part No. A58236-01.

[14] Karl Ramm and Michael Grubb, "Exu – a system for secure delegation of authority on an insecure network," *Ninth Systems Administration Conference (LISA) '95*, pages 89-93, Duke University, USENIX, September 1995, Monterey, CA.

[15] Mark A. Rosenstein, Daniel E. Geer, Jr., and Peter J. Levine,"The Athena Service Management System," *USENIX Conference Proceedings*, pages 203-211, MIT Project Athena, USENIX, Winter 1988.

[16] Gregory S. Thomas, James O. Schroeder, Merrilee E. Orcutt, Desiree C. Johnson, Jeffrey T. Simmelink, and John P. Moore, "Unix Host Administration in a Heterogeneous Distributed Computing Environment," *The Tenth Systems Administration Conference (LISA 96) Proceedings*, pages 43-50, Pacific Northwest National Laboratory, USENIX, October 1996, Chicago, IL.

Appendix 1: DirectoryGen.sh

```
#!/usr/vice/etc/pagsh
# Script to update Directory files from Simon
HTMLROOT=/afs/.rpi.edu/dept/acs/rpinfo/common/AutoGen
DirPgmRoot=/campus/rpi/simon/directory/2.0/@sys/bin
HTMLROOT=/afs/.rpi.edu/dept/acs/rpinfo/common/AutoGen
LOCKPGM=$ROOT/common/bin/wait_for_lockfile.sh
LOCKFIL=$ROOT/Dirgen_Sync_Lock
VOSRELEASE=$ROOT/common/bin/SQLVosRelease.sh
HNVR=$HTMLROOT/Needs_Vos_Release
# Define Oracle variables
LOGNAME=SimonXfr ; export LOGNAME
ORACLE_HOME=/opt/app/oracle/product/8.0.5 ; export ORACLE_HOME
ORACLE_SID=SIM3 ; export ORACLE_SID
PATH=$ORACLE_HOME/bin:$PATH ; export PATH
# Get a token
if [ -x /usr/local/etc/host.klog ] ; then
        /usr/local/etc/host.klog -t
else
        echo "Unable to get token"
        exit 1
fi
# Get a lock
if ( $LOCKPGM $LOCKFIL 10 ) ;
then
        echo "Unable to Lock $LOCKFIL"
        exit 1
fi
# Regenerate directory HTML files
cd $HTMLROOT
dirdeptgen=$DirPgmRoot/Generate_File
dirdeptpar="-target department_html -nvr $HNVR"
if [ -x $dirdeptgen ]; then
        su $LOGNAME "-c $dirdeptgen $dirdeptpar"
else
        echo "Unable to run $dirdeptgen"
        exit 1
fi
# See if we need a vos release
if [ -f $HNVR ];
then
        echo "==>[ignored: n]<== Vos releasing for"
        cat $HNVR
        if [ -x $VOSRELEASE ];
        then
                $VOSRELEASE simongen
                rm $HNVR
        else
                echo "Unable to locate sysctl, vos release aborted"
        fi
fi
```

Appendix 2: Generate_Passwd Package

```
define name=GENERATE_ETC_PASSWD_DEMO
prompt  Creating package &NAME
create or replace package &name as
-- $Header: /afs/.rpi.edu/campus/rpi/simon/prop/2.0/init_sql/RCS/
        Create_Package_Generate_Etc_Passwd_Demo.sql,v 1.1 2000/09/20
        01:11:41 finkej Exp finkej $
-- Generate a /etc/passwd file from the logins table
```

```
procedure get_data(result out varchar2, p1 in varchar2, p2 in varchar2);
end &NAME;
/
prompt  Creating package body &NAME
create or replace package body &name as
Cursor Get_Pw_Ent_Curs is
Select Username "UNAME", Unixuid "UID", nvl(Unixgid,4000) "GID",
       Public_Personal_Info "GECOS"
  from Logins
 where source like 'PRIMARY%'
   and when_marked_for_delete is null
 order by unixuid;
procedure get_data(result out varchar2, p1 in varchar2, p2 in varchar2)
is
        R        Get_pw_Ent_curs%Rowtype;        -- The data from Oracle
begin
        --
        -- First time through, we open the cursor
        if not Get_Pw_Ent_Curs%IsOpen
        then
                Open Get_Pw_Ent_Curs;
        end if;
        --
        -- Get the data into R, a record based on the query
        Fetch Get_Pw_Ent_Curs
         into R;
        if Get_Pw_Ent_Curs%NotFound
        then
                -- End of data, close the cursor and return a null
                Result := Null;
                Close Get_Pw_Ent_Curs;
        else
                -- Build the password file entry
                Result := R.Uname || ':*:' || to_char(R.Uid) || ':'
                       || R.Gid || ':' || R.Gecos || ':'
                       || '/home/' || ltrim(to_char(R.uid mod 100, '09')) || '/' || R.uname
                       || ':/bin/bash';
        end if;
end get_data;
end &name;
/
-- Call Add_Target_Simple to register this file target.
begin
Generate_File_Internal.Add_Target_Simple(
        'etcpasswd_demo',        -- Target Name
        null,                    -- Let ANYONE do this
        '$Id: Create_Package_Generate_Etc_Passwd_Demo.sql,v 1.1 2000/09/20
                                                  01:11:41 finkej Exp finkej $',
        '$Source: /afs/.rpi.edu/campus/rpi/simon/prop/2.0/init_sql/RCS/
                             Create_Package_Generate_Etc_Passwd_Demo.sql,v $',
        'Generate a /etc/passwd file as a demo for LISA',
        'passwd_demo',           -- Output File Name
        '&name..Get_Data');      -- Routine Name defined above
end;
```