Pelendur: Steward of the Sysadmin

Matt Curtin – Interhack Corporation
Sandy Farrar & Tami King – The Ohio State University

ABSTRACT

Here we describe Pelendur, a system for the management of common system operation tasks. Specifically, Pelendur focuses on the management of user accounts and related information (such as groups) across platforms and even for particular software packages (like databases) that require user authentication. Pelendur has reduced the massive process of deleting expired accounts and creating new accounts between terms from a week-long operation by several part-time operators (with subsequent cleanup by a collaboration of instructors and staff) into a completely automated process that requires less than 15 minutes of staff work and completely eliminates the need for instructor intervention.

Introduction

In 1998, our department was in the midst of a massive migration of our computing facilities wherein we moved from an architecture of many HP-UX clusters to an architecture using Solaris-based function-specific servers with thin clients in offices and labs. Some of the software used in the old environment needed to be replaced [3]. Because of severe limitations in the functionality and correctness of the largely ad-hoc scripts run by operators for the creation of accounts, it was determined that an account management system capable of managing our evolving multi-platform environment was needed.

Yet Another Account Management System?

The idea of implementing a software system for the management of user accounts is not new; past years' LISA conferences have seen many such systems. Even if we briefly ignore the issue of availability, some systems described were unsuitable because of extreme differences in the way that accounts are created and managed [1, 5] and incompatible means of handling account data [7, 11, 9, 10, 4, 8].

In the end, the most compelling reason for us to build our own software was that a grander vision existed: a single data repository for our department, which would include such things as data needed for user accounts, course-specific computing requirements, and access to various limited-access department resources. We could not find any available account management system that would work easily with any sort of database that we would construct.

The Academic Environment

Account management, though a fairly straightforward task, is quite demanding in an academic environment. Each term, we receive course rosters from the university registrar. Although students who major in computer and information science have "permanent" accounts (those that will remain until after they graduate or change majors), we have thousands of other accounts that are created specifically for the duration of the term. We have one week between most terms, which means that during this time, we need to delete potentially more than 2,500 accounts and then create another 3,000 accounts for the next term.

In this paper, and in our environment, we use a term that will be important to understand: "section". This is a specific "class" that meets together. This term is introduced because many sections of a given class can be scheduled for the same term and we need to know the most granular level of grouping available from the registrar.

System Requirements

Specific requirements for this system were identified. In its full design, our scope is actually much more broad than the rather specific task of account management. The reason for this is that parts of account management (such as determining how much quota to associate with an account) are dependent on other criteria like the requirements for courses that the user of that account is scheduled to take. For example, a course that deals with particularly large data sets might have a requirement for more than the default amount of disk quota.

Initial requirements focused on the actual management of user accounts and dependencies.

Account Management: This is the management of individual account profiles. Adding, editing, expiring, and purging them.

Course Management Courses have particular requirements (such as the need for one platform or another and the use of a software system like Sybase) that need to be configured and managed. Some of these configuration options are simple matters of preference. Others are matters of policy, which the course coordinator does not have the authority to change. (Though we refer specifically to courses, there is nothing that prevents these entities to be managed from being project groups, departments, or any other sort of group that might have particular requirements in other environments.)

Resource Management: Anything that exists in our environment (such as ''Unix machines'', ''NT machines'', ''Sybase database'', ''disk quota'', ''print quota'', and ''color printers'') might also need to be managed. As these are neither accounts nor courses, but share basic properties, we classify all of these as ''resources''. The basic point here is that users who are administratively responsible for these resources manage them through Pelendur instead of having us perform all of the management tasks for them.

Additional requirements were slated for future development. Some of these features are now implemented and others are still on our ''to-do'' list.

Unix groups: As we use groups to manage sets of users on the Unix systems, Pelendur should be aware of groups and know how to use them. (This interface is now partially implemented; new groups are created, but old groups are not garbage collected.)

Mailing lists: Some of our course instructors and students prefer to use mailing lists to stay in touch. We presently use both faculty-maintained aliases files and Majordomo [2], but are now investigating the possibility of replacing both of these mechanisms with Mailman [12].

Course directories: Because some courses have group projects or software that is specific to the course, we need to be able to associate directories with a given course. Handling of filesystem permissions should enforce the policy for read and write access established by the instructor.

Multidomain management: Currently, everything that is a part of the instructional environment is considered ''the system''. Accounts that belong to individual research labs are not managed by Pelendur but it could be convenient for us to have that option available. Should we do this in the future, it would be nice to have the option of having a single Pelendur installation be able to manage multiple ''systems'', rather than having to make a new installation of the software for each domain that needs to be managed. This feature would also be useful to include access to limited-access machines, such as those that are set aside for long-term computationally-intensive jobs.

Electronic lock systems: A relatively new addition to our department is an electronic lock system, whereby ID cards are used for access control instead of physical keys. Presently, this is managed by a standalone DOS-based system.

Course newsgroups: Most courses in our environment is assigned a newsgroup on our local news server. This is the default means of providing an ''out of class'' communication channel. Presently, we just create new groups as new courses are added and cancel the messages in the groups at the end of each term by hand. Though this is not a big time-sink, we would like for these processes to be automated.

Course-specific environment: Some courses have particular environmental needs, such as a course-specific $PATH setting, for example. Pelendur can provide this.

''Any computing resource'': As we continue to move forward, other resources are identified and incorporated into the system's functionality. At a very high level, the goal of the system is to manage the systems' configurations so that system administrators can do other things that computers can't do very well, like planning.

Design and Implementation

Pelendur is a large system, made up of several programs. We'll first describe the philosophies that influenced the system's design and then consider the programs and major library modules that these programs use.

Data-Driven

The entire system sits atop a Sybase relational database. Rather than creating code that would depend upon very specific data, working with entities that make sense for our environment, we opted to put as much of the system in data as possible and to make those data be as generic and flexible as possible. Thus, rather than dealing with courses, sections, and instructors, other users of Pelendur will be able to work with teams, departments, and project coordinators. Each deployment of Pelendur will define its own terms and the relationships among them that make sense.

We believe it important to emphasize that this makes the integrity of the database especially critical. In such a highly dynamic system, we're not dealing with simple cases of the Wrong Thing failing to achieve the desired result. Data that have been compromised by a moderately clever attacker can be used to attack the system itself, creating accounts for attackers, granting them privileges to the entire system, and possibly even running commands with superuser access.

The schema is represented in Figure 1. Here we describe each of these tables in some detail.

People contains information about a person. When a person is added to the table they are assigned a database identifier and a user login. Those two values will be used to tie a person to their resources.

Account is a table that contains information about individual accounts.

Classification contains information on thingies in the system. A thing can be anything with the exception of a person or an individual account. Most classifications define a resource or a membership group. There are also templates and defaults in Classification that are used to create resources and membership groups or define certain values in the system (e.g., what the current quarter is). Each classification is assigned a unique number (SID) when it is added to the database.

MembershipIn contains the memberships for the membership groups from Classification. It maps a user login to a SID and is used to determine what resources that login should have.

UserRights defines owners of classifications and grants access to users for a classification. This allows a user to manage resources for their membership groups.

ResourceUsedBy defines what resources a classifications has. It also indicates if the resource can be edited by the owner or proxy of the classification.

ResourceGroup links classifications together.

Flexible

We made an effort to avoid ‘‘hardwiring’’ anything in the system. This was largely accomplished by taking a very dynamic view of the data. That is, instead of having an ‘‘account’’ with a ‘‘disk quota’’ field that would be assigned a value based on the state of the system at the time of the account’s creation, all information about an account must go through a process of resolution, where its dependencies are determined and the values for each of the account’s properties are resolved at run-time. This view was taken for everything in the system, not just accounts.

Something else that we incorporated in order to allow maximal flexibility is a system of property inheritance. Although dealing with a relational database system at the core, we were able to provide the ability to inherit properties from a parent by specifying a relationship between various records in the database by defining a ‘‘parent ID’’ (Called ‘‘PID’’) as a means of determining which ‘‘SID’’ is one step closer to the root than the current.

This gives us the ability to specify object-oriented ‘‘is-a’’ relationships between records in the database. Thus, the ‘‘tree’’ of elements to be resolved can be of an arbitrary depth, allowing each site (and each particular type of resource being managed) to have an appropriate number of levels to support the sort of abstraction desired, without forcing some high level of overhead on those who do not need to deal with such abstractions.

A good example of how we use the ability to inherit properties is in the case of a series of courses that a student will take in sequence. There will be some Classification table entry that will identify the series. Each course in the series will have its own Classification entry whose PID identifies the Classification entry of the series as the parent. Each section in a course will have its own Classification entry whose PID identifies the Classification entry of the parent course. Thus, configuration changes are made at the appropriate level: those that affect all sections in the

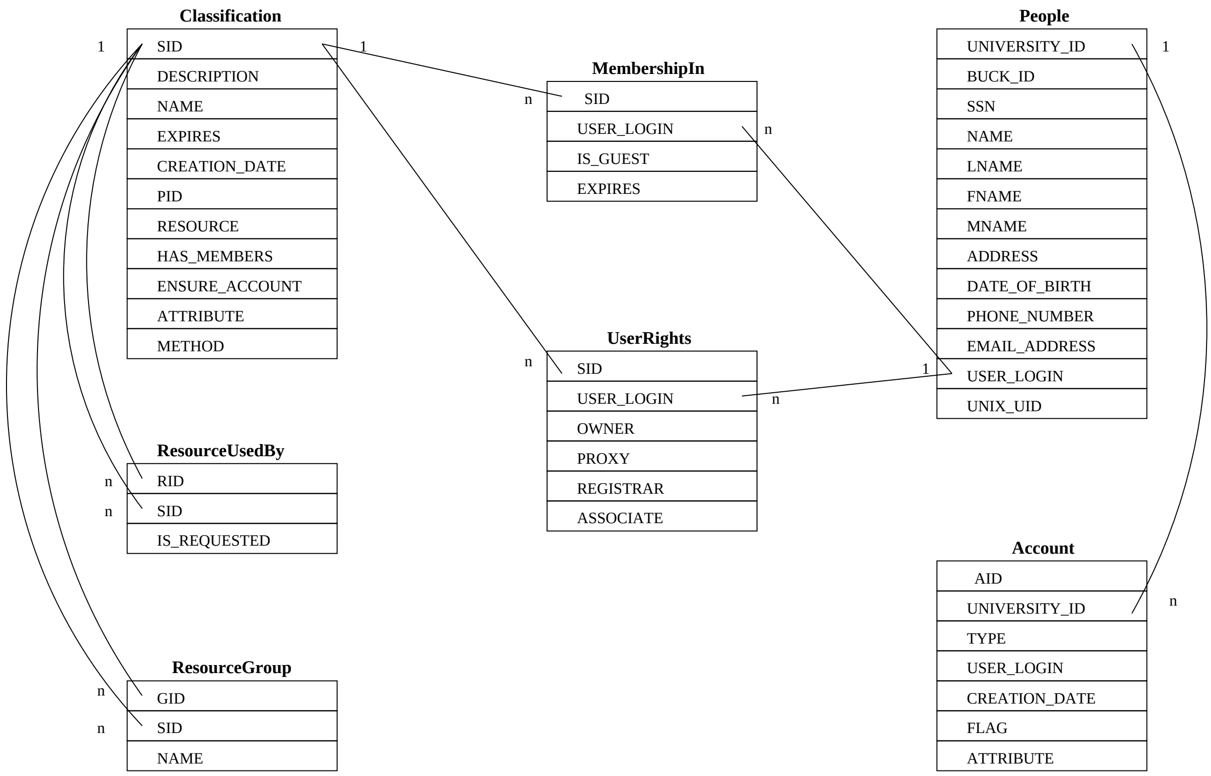

Figure 1: Schema of Pelendur’s Database

series will be made in the Classification entry for the series, those that affect all sections in a specific course in the series will have those changes made in the course's Classification entry, and those that are specific to a particular section will be made in that section's Classification entry. When we need to determine how much quota, for example, an account has, we'll determine which sections in which the account has membership. Those will be resolved by walking up the tree until we get to the root object (as determined by having a PID of 0), populating the fields in memory with the values in the database and returning. By the time the initial Classification entry returns, we will have queried each level in the tree, populating the object with the levels specified at the highest level first, and overriding those with whatever (if anything) was specified in the lower levels.

Additionally, where values are numeric, they need not be absolute; signed numeric values indicate relative values. Consequently, a property of a Classification might be to increase disk quota by 30MB, instead of specifying some absolute value.

Modular

Rather than requiring code changes in many different places in order to create new interfaces in the system, we designed the system to be made up of a basic core which includes the database and the resource resolution logic. The rest of the system interfaces to that core. The core understands when an account needs to be created, or what the properties of an account at any given time should be, but it knows nothing about creating accounts. Instead, it can tell what the account's properties are and what system(s) need to reflect those properties. If the account needs access to both Unix and Sybase, the core will pass the appropriate information to the modules for Unix and Sybase through a standardized interface.

As described in section on implementation language, this design makes it possible for even the database itself to be replaced with another database that has the same schema.

Implementation Language

Perl was chosen as the implementation language. Its freely available DBI (database interface) and DBD (database driver) package for talking to a wide variety of databases makes it an excellent option for building atop a database:

- Very little investment is made in an interface to any particular database (since we code to DBI), thus allowing us (theoretically) to use any database for which a DBD module exists. In practice, we did use one Sybase-specific feature, which we can and will in the future stop using.
- We can spend our time focusing on the problem at hand, instead of how to write to the database.
- Being free, the price is right.
- Since source code is available, if there is a problem with it for which a fix is not available, we can fix the bug ourselves.

Because parts of this system will need to run with superuser privileges, we're concerned about safety of our code. Perl has some excellent safety features: namely, the ability to identify "tainted" data and support for arrays and buffers that grow dynamically.

Finally, Perl is available on a huge number of platforms. As long as we keep portability in mind, Perl will provide us all of the language support necessary to allow our code to run unmodified on essentially any system we could use in the forseeable future.

Programs

rosterload is the program that loads rosters into the database. In our environment, rosters are course rosters that identify which students and instructors should be associated with a given section. For the most part, rosterload just calls **Roster::rosterload**, which does all of the work. Our plans are for rosterload and Roster.pm to be modified to be able to load roster information directly out of the Data Warehouse or from email.

makeaccounts is the program responsible for creating things, that is, resources and accounts. First it will initialize the resource methods, then it creates any resources that don't already exist. Then it creates any accounts that don't exist. We have a cron job run makeaccounts runs twice per day.

expireaccounts is our garbage collector; it's responsible for removing things from the system. Expired resources are removed from both the database and from the systems that were used to support the account. After this has been accomplished, it will remove entries in the "MembershipIn" table that have expired and then any user resources that are specified for the user in the database. cron runs expireaccounts once per day.

notifier sends account removal notifications. All accounts scheduled to be removed will receive a seven day notification. Account that are 'persistent' will also receive a 30 and 14 day notification of expiration.[1] Note that this includes not only accounts for operating systems, but this also includes "accounts" in software systems like Sybase. As Pelendur continues to help us blur the distinctions among different systems that comprise the "computing environment", we'll move away from providing notification that specific systems' accounts will be unavailable and provide notification only on the user's "meta-account" in the department. The Sybase

[1]The amounts of time on the notification are configurable parameters.

account will be created as soon as the student shows up on the roster for a class that has this resource. The account will always[2] match the Unix login name and will have the typical default password.

crconfig is the course configuration interface for instructors. This provides a convenient means for them to manipulate the database in a controlled manner, allowing them to change only that which is under their administrative authority without creating artificial restrictions that require staff to perform their administrative tasks for them. Instructors and their "proxies" (those whom they designate) to add and to remove resources from a particular section and a course default. Project and course directories are also configured through this interface.

cradmin is the account management administration tool. Essentially, this is the interface that is used to manipulate the state of the database on anything that is in the database. Where crconfig deals with abstractions and has a "course-specific" view of the world, cradmin allows manipulation of non-course-related resources. The user interface itself still works with many of these abstractions, but it is in this program where we can make changes to the user interface to allow administration of new resources.

Modules

Here we describe the major modules of the Pelendur system. Many of these modules are shared by various standalone programs in the system.

IICFDB.pm is the module contains the generic routines for interacting with the account management database. There is typically a routine for each table for searching, adding, and removing. They follow the naming convention get_<table-name>, add_<table-name>, remove_<table-name>. Table names are converted to lowercase and underscores are used where there would be white space ("MembershipIn" becomes membership_in]). For some tables an update function exists also. The get_ routines are all polymorphic and will do different searches depending on what data is passed to the routine. This module also contains the routines to walk the database recursively (resolve_pids) in order to resolve all dependencies and provide an up-to-the-second view of an account's properties, as determined by what the database knows about the account.

IICFLog.pm is our logging mechanism. Presently, this basically accepts messages and puts them in the "right place", but the intention is that it will be a general-purpose log gatherer for all applications in our environment.

IICFLogin.pm contains the routines that deals with logins in our environment. It contains the routines that generates new usernames and default passwords.

NT.pm contains the routines to do all things on the NT systems. It is currently not implemented; an older standalone account creation and deletion system was developed locally for NT accounts. Instead of implementing this part of the system initially, we opted to focus on the Unix, Sybase, and core database portions of the system, building an interface between Pelendur and the old NT account management scripts. This decision has turned out to work relatively well for us, and has saved us from what could have become a large amount of redundant work as Microsoft seems to think that making major interface changes from version to version of its operating systems is an appropriate thing to do. Thus, we can avoid writing most of NT.pm until the NT based systems are on a newer version of the operating system than the current programs support.

PQuota.pm interfaces with the printing system's quota handling. We use LPRng [6] for our printing system throughout the department.

ResourceMethods.pm is a module that contains all of the methods for the resources in the account management system. Each resource has its method defined for it in the METHOD field in its "Classification" entry. When called, this method will 'do the right thing' for the resource. For example, a resource like "mailing list" might have a method that specifies a program to be run in order to create or to delete the mailing list.

Roster.pm contains the routines for processing the roster.

Sybase.pm contains all of the routines for dealing with Sybase resources.

Unix.pm contains all of the routines for Unix resources. It adds and removes accounts in the password file, manipulates the group file, and generates the Quotas file for disk quotas.

mkcisdir.pm is used to create and to remove directories on the Unix systems. It handles several types of directories: users' home directories, group project directories, and the directories used by Submit, our program for electronic laboratory submissions. Where possible, this module will run as the owner of the directory instead of root.

The Effect of Pelendur

Our environment has benefited tremendously from Pelendur in many ways. What used to be a

[2] "Always" is a pretty strong word. There are a few exceptions, as the result of ancient accounts that predate Pelendur that still contain dashes (-) so they can't be used as logins to Sybase. If the username has a dash in it, the dash will be converted to an underscore (_) for the Sybase account. New accounts are always created with names that are portable across our systems so that these kinds of conversions will not be necessary.

painful experience for all is now essentially a non-event.

Labor

The total staff labor expenditure for processing accounts between terms is greatly reduced.

- Accounts to be removed are no longer processed manually. When an account has no more references (managed through the MembershipIn table), the account is garbage collected.
- Instructors would manually add and remove students from their sections using hardcopy provided by the university registrar. We now get these data from the registrar directly and automatically add and remove students.

Error Rate

Historically, this has been a problem. The old software had to run by someone who knew its idiosyncrasies and limitations. Mistakes were frequent and could easily require several hours to fix. Errors are now much less frequent, because we get data directly from the registrar and do not require any manual intervention before account configuration. Because the entire state of the system is driven by the database, errors can now be fixed by making appropriate changes in the database and waiting for Pelendur to propagate them.

Since managing our systems with Pelendur, we have been able to identify accounts that have expired long ago but were never removed, to identify problems that arise because of an account being misclassified, and generally to free ourselves of the kinds of concerns that come about when the administrators need to manage things manually.

Latency

In section "Programs", we identified which parts of the system run on a regular basis in our environment. These are configurable to a site's requirements. In our environment specifically, this means that changes made take no more than one day to take effect. This is a huge difference from the days, weeks, or even more that it took under the old system.

Future Work

Quite a lot can still be done with Pelendur. Specifically, we need to increase the number of systems against which we can interface, including native NT account management, more intelligent

Conclusions

Account management can be greatly simplified by taking a more abstract view and thinking of system access as a property that results from the state of the account. Pelendur has proven to be a highly effective means of managing a very large number of highly variable accounts.

Availability

Although the system has been designed and implemented in a way that emphasizes flexibility and freedom from very a very site-specific view of the world, it will still take quite a bit of effort for another site to bring Pelendur into production. We're currently working on finishing the functionality and hope that we will be able to revisit some of the areas of the system that work for us but would make it difficult or impossible for other sites to use the system as-is. This work is geared toward making a general release of the system. We have no idea when this could possibly take place.

Author Information

Matt Curtin is founder of Interhack Corporation, which helps developers and system managers build and run systems they can trust. He is also a Lecturer at The Ohio State University's Department of Computer and Information Science. His current interests include Lisp programming, secure systems development, and Internet privacy. Reach him electronically at cmcurtin@cis.ohio-state.edu .

Tami King is a senior software specialist for the Computer and Information Science Department at The Ohio State University. Since graduating from Utah State University with a B.S. in computer science, she has worked as programmer, UNIX systems administrator, and database administrator. She also managed a group of UNIX systems administrators before returning to programming, what she enjoys most. Her free time is taken up by her husband, her new baby boy, and pedaling out of sync. Reach her electronically at tami@cis.ohio-state.edu .

References

[1] Arnold, Bob, "Accountworks: Users create accounts on SQL, notes, NT, and UNIX," *Twelfth Systems Administration Conference (LISA '98)*, page 49, USENIX, Boston, Massachusetts, December 6-11 1998.

[2] Chapman, D. Brent, "Majordomo: How Imanage 17 mailing lists without answering "-request" mail," *Systems Administration (LISA VI) Conference*, pages 135-143, Long Beach, CA, USENIX, October 19-23 1992.

[3] Curtin, Matt, *Creating an Environment for Reusable Software Research: A Case Study in Reusability,* Technical Report OSU-CISRC-8/99-TR21, The Ohio State University, Department of Computer and Information Science, August 1999.

[4] Geer, Daniel E. Jr., "Service Management at Project Athena," *Large Installation Systems Administration Workshop Proceedings*, page 71, Monterey, CA, USENIX, November 17-18 1988.

[5] Harris, J. Archer and Gregory Gingerich, "The Design and Implementation of a Network Account Management System," *10th Systems Administration Conference (LISA'96)*, pages 33-41, Chicago, IL, USENIX, September 29-October 4 1996.

[6] Powell, Patrick and Justin Mason, "Lprng – An Enhanced Printer Spooler System" *Ninth Systems Administration Conference (LISA '95)*, pages 13-24, Monterey, CA, USENIX, September 17-22 1995.

[7] Riddle, Paul, Paul Danckaert, and Matt Metaferia, "AGUS: An Automatic Multi-platform Account Generation System," In *Ninth Systems Administration Conference (LISA '95)*, pages 171-180, Monterey, CA, USENIX, September 17-22 1995.

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

[9] Spencer, Henry, "Shuse: Multi-host Account Administration," *10th Systems Administration Conference (LISA'96)*, pages 25-32, Chicago, IL, USENIX, September 29-October 4 1996.

[10] Spencer, Spencer, "Shuse at two: Multi-host Account Administration," *Eleventh Systems Administration Conference (LISA '97)*, page 65, San Diego, USENIX, California, October 26-31 1997.

[11] Tomas, Gregory S., James O. Schroeder, Merrilee E. Orcutt, Desiree C. Johnson, Jeffrey T. Simmelink, and John P. Moore, "UNIXhost Administration in a Heterogeneous Distributed Computing Environment," *10th Systems Administration Conference (LISA'96)*, pages 43-50, Chicago, IL, USENIX, September 29-October 4 1996.

[12] John Viega, Barry Warsaw, and Ken Manheimer, "Mailman: The GNUmailing List Manager," *Twelfth Systems Administration Conference (LISA '98)*, page 309, Boston, Massachusetts, USENIX, December 6-11 1998.