DARRELL FUHRIMAN

identity management identified

Based in Portland, Oregon, Darrell has been managing UNIX systems for nearly thirteen years in a wide variety of environments. This fall he will be changing gears and pursuing a master's degree in geography at Penn State.

darrell@garnix.org

AS ANY IT WORKER WHO HAS SPENT time in the enterprise knows, the question of creating and managing accounts across a variety of systems is a thorny one.

Windows doesn't easily talk to UNIX, which doesn't easily talk to your ticketing system. None of them talks to your HR system, which has its own set of logins, plus everything wants to manage its own password and of course nothing stores passwords in the same format. And just how do we assign someone a phone extension?

When an employee gets hired, tickets have to be opened with each of the groups managing each system until finally, possibly days later, the new employee has all the access he or she needs. But then over the years, the employee changes roles and needs a different set of systems until finally, several years later, the employee leaves. Then someone has to figure out what systems the employee has access to (since the employee may not actually know anymore) and set about removing access, deallocating resources, forwarding mail to the manager, etc.

These are the kinds of problems that an Identity Management System is meant to solve.

What Is Identity Management?

In an enterprise IT infrastructure, we have a wide variety of systems, many of which need to know something about who's using it. What each application needs to know varies dramatically: An HR system may need detailed information on every employee; a Web server may simply need to know only basic authentication and authorization information; and the PBX may need to know who a person's assistant is, so the phone can be automatically forwarded if it's not answered after a certain number of rings.

Identity Management (IdM, if you're hip) in its simplest form is nothing more than the tracking of all attributes about an individual and the synchronization of those attributes among different data sources. Known as a *metadirectory*, this is the initial concept behind what we now call Identity Management.

This differs from the concept of account management, which is the process for allocating resources to a person (a login account, a home directory, etc.) and removing those resources when told to. Because there are typically numerous resources that need to be allocated to a person, it has traditionally been important to automate these processes as much as possible. That's the traditional view of account management, and one that has been much discussed at past system administration conferences.

In practice, account management often fails to take into account the question of when and which resources should be provisioned as a user's role changes and, more importantly, when they should be deprovisioned. In other words, we need to regularly update the resources allocated to a person through his or her entire tenure within the organization.

As system administrators, we have often automated the process of creation and deletion. I don't think I know a system administrator of significant experience who hasn't (re)implemented this multiple times; the process that triggers the creation and deletion is usually some action that is external to the network such as email, a phone call from HR, or the arrival of someone in their cube.

Rather than having a manual initiation, if we take our metadirectory, where we share data among different systems, and we add actions triggered by changes in these data, we would have a system that managed a user's resources and permissions with no manual intervention. We would have an Identity Management System.

You might say, "I can do that," or you might say, "That sounds hard." Either way, this is much harder than you think it is. At one regional conference I attended most of the stories were about how poor planning had led to repeated attempts to deploy the solution—one large company was about to attempt their fourth roll-out because of unrealistic timelines and a lack of understanding of the complexity of the problem.

So, How Do I Do It?

The modern enterprise typically has dozens of connected data sources (and accounts on each one), so there isn't going to be a single group that knows everything about all the systems. That means you need to get to know your other IT groups—you'll need them.

I worked on an IdM project at a university where our initial step was to do a survey to discover what accounts existed on which systems. We discovered seventeen different account types, including:

- Calendaring
- Ticketing system
- Student information database
- MySQL databases
- Temporary wireless access
- Online course management

Even after the discovery phase was officially over, we continued to find new account types up to a year afterward.

The next thing was to identify roles (Student, Staff, Student Employee, etc.) and associate them with the types of accounts they were allowed. This was especially daunting in a university, where roles were quite fluid. It was common for Students to become Faculty, for Faculty to become Staff, and Staff to become Students, often maintaining more than one role at the same time.

After identifying roles, you should build a matrix identifying which roles have what attributes. This will be invaluable as you progress in the construction of your IdM system.

The lesson here is that when implementing an IdM solution, the single most important thing you can do is to plan well. That doesn't necessarily mean you need to go crazy with the GANTT charts right away—what it does mean is that you have to really understand your data.

Planning a Deployment

We know we have a large number of account types and resources we want to manage, but the single biggest mistake you can make at this point is to bite off more than you can chew.

A successful implementation is a carefully *staged* implementation. By implementing your IdM in stages, you learn something new in each stage, and you can move from relatively simple implementations and workflows to more complicated ones.

For instance, as your first step you could implement a metadirectory that simply moves attributes from one data source to another, possibly modifying the data along the way.

This requires that you understand first the data in each source, then where that data needs to go, and finally who is allowed to change it. Reread that sentence—it's a key one.

Let's look at a fictitious corporate phonebook—a relatively simple application with two pieces of data, namely, a name and a phone number. Unfortunately, our phonebook has three methods of access—people can look it up inside Outlook, which looks in the Active Directory, or it can be accessed via a Web page, which pulls that information from a UNIX-based LDAP server, and of course people can use the directory functionality built into the PBX and phone.

So we have three data sources that ideally should have the same data. Obviously, the PBX knows best what someone's phone number is, but it doesn't really have any way of knowing the name assigned to it, unless you tell it. Similarly, we need to get the phone number back into the two phonebooks—probably through a manual process.

So, let's look at a table that reflects this:

Attribute	*Contributing Data Source*	*Export Mapping*
First Name	AD givenName, LDAP givenName	AD givenName, LDAP givenName, PBX FIRST_NAME
Last Name	AD sn, LDAP sn	AD sn, LDAP sn, PBX LAST_NAME
Full Name	AD givenName + sn, LDAP givenName + sn	AD cn, LDAP cn, LDAP gecos
Phone Number	PBX EXTENSION	AD telephoneNumber, LDAP telephoneNumber

In this table we have all[1] of the information we need to start implementing a metadirectory. We know that the PBX controls our extension and it needs to go onto both AD and LDAP. More importantly, we know that changes to that information in sources outside of the PBX are incorrect—and a good IdM system will correct those mistakes automatically.

1. This is a bit of a lie. For example, you still have to figure out how to deal with multi-valued attributes, and this simple table makes no mention of data transformation requirements—for instance, an LDAP date string probably can't be directly fed into an RDBMS DATE type.

Of course you need to do this for each and every attribute—and even a simple system is likely to have dozens of attributes that need to migrate.

So all of this is nice, but wouldn't it be better if rather than simply *moving* the data—which still requires a manual intervention at each data source to create—we *create* the data. That's where our metadirectory becomes an IdM system.

Because people usually have phone extensions assigned to them when they are hired, we can define a workflow around the creation of a user. Now, when we add an employee to the system our IdM system will execute a series of actions to ensure the user is provisioned appropriately.

In our example, assume our PBX provides an interface for programmatically adding extensions, configuring voicemail, etc. We can take advantage of this so that when we create a user in the Active Directory, the IdM system executes a workflow that creates an extension in the PBX and in return gets the phone number, which it then distributes to any export-mapped source.

By doing that, we've taken one manual step out of the process of adding a user. More importantly, we've gained insight into the quirks of our IdM system and we've programmed an external action to be triggered based on an action. But why stop there?

If we move the entire account creation system inside the IdM system, then not only can we add phone extensions but we can create a mail login and set its quota, create a home directory, add them to NIS, etc.

At this point, why should we stop with this interface? If you connect it to your HR system, you can automatically trigger the creation of accounts when a person is hired—and in a move that will make your security officer happy, you can disable or delete them when a person leaves the company.

Let's say we want to design a workflow to allow your helpdesk to change someone's quota. But what happens if you have charge-backs for your disk allocations, which require approval for the charge? Rather than update the account immediately, before the change occurs you must send it up the chain to the person's manager who can approve the charge. And then if we discover that there's not enough space for that allocation, we must have the system open a ticket with your system administrators to migrate the mailbox to another server—and automatically update the quota when the ticket is closed.

As you might guess, these workflows can grow very complicated. This is why it is absolutely critical that you understand your data before you start an implementation of an IdM system. This is doubly true if you have a large number of systems or types of users.

Because IdM is so daunting, it's highly unlikely you'll want to grow your own solution. Most of the software currently available comes from the big names: Oracle, Sun, Novell, and Computer Associates. These products have a variety of tools for defining what each data source has versus what it should have and, most importantly, for executing actions based upon that.

Most of these tools have native support for many directory sources, such as LDAP, NIS, and Active Directory. The best also include the ability to write custom code to connect with other sources and the ability to insert custom code to do data transformation.

A Plea

Unfortunately, not all software vendors have realized the importance or magnitude of the problems of account and identity management. Many prefer to work in their own self-contained world, each housing their own user database, passwords, and often even access controls.

Much of what is managed by an IdM system is necessary only because we are trying to connect these worlds together. A better long-term solution is to integrate applications into single identity stores and, where that is not possible, to provide external interfaces for easy integration into the enterprise IdM system.

There's no reason Web-based course management software should have its own database of users, nor your trouble-ticketing system, nor your enterprise calendaring system. This is a house that cannot stand for long, and it is critical that we, the in-the-trenches practitioners, demand more of our vendors.

Conclusion

In the end, the idea of an Identity Management System is a simple one—monitor attributes, and when those attributes change, perform actions based on those attributes' values.

However, this simple concept has very complex implications. To manage this complexity, it's important that your implementation be very well planned and begun only after obtaining a thorough understanding of the data you are working with.

If you're careful, your IdM system will improve data integrity, flexibility, and security and can automate numerous mundane tasks—and that thorny problem of systems management will be greatly reduced.

RESOURCES

Mark Dixon and Pat Patterson of Sun both write interesting blogs on Identity Management: http://blogs.sun.com/roller/page/identity?catname=%2FIdentity; http://blogs.sun.com/roller/page/superpat?catname=%2FIdentity.

Dave Kearns of Network World writes on IdM and security topics: http://www.networkworld.com/topics/identity-management.html.

Kim Cameron writes a more technical blog on IdM, metadirectories, and security: http://www.identityblog.com/.