Access Management and Provisioning

Identity Provisioning is a technology originally designed for enterprise account synchronization and management of identity-related business processes. This technology originated in the enterprise environment but it soon became obvious that it has much broader applicability. Today it is almost impossible to find an identity-related solution that can be complete without any kind of identity provisioning component. However there is still a significant number of engineers that do not realize this fact and continually try and fail to deploy partial identity management solutions. This approach was perhaps justifiable in 2000s when provisioning technologies were unreliable and expensive. But today there is no excuse for omitting provisioning technology from an identity management solution. The following paragraphs explain the reasons.

What is the Problem?

A huge number of identity management solutions are based on access management technologies. This includes Single Sign-On (SSO) solutions, identity federation, Internet "user-centric" identity solutions, etc. However almost all access management technologies have a critical weakness: transfer of user profiles. The very principle of these technologies and their current implementation means that the transfer of user profiles is inherently bound to a user accessing the system (this is "access management" after all). The usual flow goes like this:

- **Step 1:** User is accessing the application or service.
- **Step 2:** The application does not have a session for the user therefore the user is redirected to the authentication services (e.g SSO server or identity provider)
- **Step 3:** Authentication service authenticates the user (if not already done) and forms a *token*. This token usually contains user identifier and may also contain parts of user profile.
- **Step 4:** User is redirected back to the original application. The token reference is part of the redirected request.
- **Step 5:** Application uses the token and extracts user identifier and parts of the user profiles. The application will use such data and works normally.

This is the theory. But it is also often used in practice and even recommended as a satisfactory and complete solution. However the truth is that this approach is satisfactory only for very simple applications. It works only for applications that do not need to do any serious processing with the identity-related data. On the other hand vast majority of non-trivial applications needs to locally store at least some portions of user profile. There are numerous reasons why the applications need to do so. Perhaps the most pressing requirements are:

- **Notifications.** such as mail message or an SMS notifying the user about a certain event. Such events happen asynchronously and user may not be accessing the application at the time when the event happens. Therefore user profile may not be accessible from the access management system. Therefore the application needs to store part of user profile locally or it needs to have (asynchronous) access to a database that maintains the user profile. In enterprise environment this is usually directory service (e.g. LDAP) and chances are that it is readily available. However it may complicate integration for some (typically older) applications. However the problem is much more pronounced in federation and other Internet-based solutions. There usually is no such directory service and the only option is to store parts of user profile on the application side.
- **Reporting**. Even the simplest report that contain user's full name needs local storage of profile data. The full name needs to be stored in the same database as the other data that goes into the report otherwise the report cannot be really efficient. Even the use of directory service (LDAP) is usually out of question in this case. It is next to impossible to do efficient data "join" between regular relational data and data from the directory service without resorting to some kind of data synchronization technique.
- **Integration complexity**. Identity and access management (IAM) is not there for its own sake. The goal of IAM is to make *other* applications work efficiently. But there is a huge number of applications that are difficult to change. Typical business application is based on top of relational database and it expects that all data will be stored in the same data storage. This is often not possible to separate user profile data and still maintain the same performance, scalability, security and other systemic qualities. And this is the usual case. However there may be some cases where it is theoretically possible to store part of the data in the relational database and the other part in directory service (LDAP) without compromising systemic qualities. But as too many applications are built directly on top of the database layer it is very difficult to implement as it requires fundamental changes to application architecture. Such a change is almost always economically infeasible.

Practical experience clearly indicates that vast majority of non-trivial applications needs to store parts of the user profile locally. The applications just cannot efficiently work without it. Therefore there is usually one more step in the access management flow described above:

- **Step 6:** Application stores the relevant part of the user profile into the local database. If there is already a profile for the user then such profile is updated.

This approach apparently solves the problem. It looks like a satisfactory solution applicable to a very broad range of applications. It is sometimes even recommended as a best practice. However the reality has a couple of surprises.

A system that does this "on demand" management of user profiles may actually work very well after the initial deployment. This is the reason why this method is so often used and recommended. However some time after the deployment the things start to turn for the worse. Critical issues begin to appear. The primary cause of the problems is the on-demand synchronization of user profiles which is very far from being perfect. It is actually closer to a "hack" than a real solution. Such synchronization happens only when user is accessing the system. Therefore it works very well for users that access the system for the first time. That explains why such a system works very well when deployed: all the users are accessing the system for the first time. It also works acceptably well if the users are accessing the system frequently. And frequent access is what typically happens when the system is tested and also during the pilot deployment. Therefore the issues may go undetected for quite a long time and such system may pass all the testing with flying colors. But the issues start to appear later in the system lifecycle when there is a longer delay between user visits. The synchronization becomes less frequent and the data quickly get out of date. Even such a benign data item as a user full name changes surprisingly often. There are thousands of marriages *every day* and each of them is likely to end up in the change of user full name. And the wedding is still quite a rare event in the life of an individual. Changes of telephone number, e-mail address, postal address, locality and organization affiliation are much more frequent. User profile is much more dynamic than an average engineer would ever suspect. Locally stored user profile data become outdated surprisingly quickly. Under normal operational circumstances this is usually a matter of days or weeks rather than months or years.

When locally stored data become outdated then a number of severe problems appear:

- **Unreliable notifications**. Notification services take contact information from locally stored user profile. If users are not accessing the system often then the local copy of user profile deteriorates. The notifications are sent to old addresses or phone numbers and the notification system becomes mostly useless. This problem is especially troublesome for applications that are not frequently visited such as marketing-oriented applications, many applications in the utility segment, eGovernment applications and so on.
- **Data deformation**. Reports and similar bulk data processing routines usually has no practical option but to use the locally-stored data. If user profiles are stored locally then the reports become more and more deformed every day. The reports finally become totally useless as they are based on outdated data. Such reporting facility is a plain waste of time and money.
- **Deprovisioning problem**. This is unquestionably the most severe problem of access management technologies. Access management technologies can create user profile on demand and even somehow keep it imperfectly up to date. But there is absolutely no "pure" way how to delete the information. The updates are always bound to user accessing the system (hence "*access* management"). But if user is deleted in the original system (e.g. central directory service or identity provider) there will be no more access. User that does not exist just cannot access the system and therefore cannot convey the information that it was deleted. Local copies of inactive user profiles are just hanging around with no systematic way to clean them up. This is the cause of many severe problems:
    - **Impact on performance**. It is not uncommon that an application contains more inactive users profiles than active ones. Modern applications and databases are built to be scalable. But scalability means that the system resource consumption increases proportionally to system utilization not that it does not increases at all. Such "dead wood" can have severe impact on system performance even if the system is scalable.
    - **Security**. Access management technologies often replace the primary authentication method of the applications. Therefore if a user record is removed from the original data store, such user is not able to authenticate any more. This is often regarded as an acceptable solution. But many applications have secondary authentication mechanisms that a user can enable. This ranges from various "remember me" features, security questions, SSH keys to permanent access tokens (OAuth or other tokens, e.g. for mobile devices). Some applications do not even have the option to completely disable legacy authentication methods. Therefore even if a user's primary authentication is disabled, the user may still be able to access the system using a secondary authentication mechanisms. The security risk is still there, unless user profile is completely removed from all the applications.
    - **Personal data protection and privacy**. Every business that honours user privacy allows a user to completely wipe his data from all the data stores. Parts of this requirement is also given by legislation. But this cannot be done by access management technologies alone. Bits and pieces of user profile are scattered across many applications, and there is may not be a list of where the data is located. Therefore even if the central user profile is delete some traces of user information remain. This is clearly a violation of good privacy practices and it may even be a violation of applicable law and regulations.
    - **Licensing cost.** The cost of licenses and support for many applications are based on the number of application users. But if a large number of inactive identities remain in the application then the licensing cost may be unnecessarily high. This may a significant waste of money even if the inactive identities remain in the application for a relatively short time.

These problems also apply to identity federation and Internet-based (user-centric) identity solutions as these are based on similar approach. Therefore similar reasoning also applies to solutions based on OpenID, SAML and WS-Federation, OAuth and similar technologies. Similar problems also apply to the very popular APIs that form the "API economy" as these also frequently create copies of user profiles using the on-demand fashion.

Some deployers of access management solutions are obviously aware of these problems. They usually try to resolve them with "hacks" such as clean-up of account data that is inactive for several months. But such techniques are usually very unreliable, they cause severe user discomfort for non-frequent users and they still do very little to address the security risks and privacy concerns. Except for a very few cases such "solutions" are absolutely inadequate.

The knowledge about these severe problems is relatively widespread among engineers that work on enterprise solutions. These concerns were documented as early as 2006 and the general knowledge was there ever before that date. However these facts are very little known in the Internet-oriented environments. It almost looks like the Internet-oriented engineering community is ignoring the issues.

Identity Synchronization and Provisioning

It is impossible to avoid duplication of any data in network. Actually any transfer of data over the network creates a copy. Sometimes such copy lives only for a couple of moments but it is not unusual for the copy to exist for months or years. As we cannot avoid copying the data, we need to manage the copies. And that is what we call *synchronization*: a process to create, maintain and dispose of data copies in a manageable way. All the *identity provisioning* systems are built around this concept in one way or another. Some are lightweight and elegant others are heavyweight cumbersome monsters. But all of them essentially synchronize identity-related data. It essentially works like this:

- Provisioning system continually monitors identity information source (or sources) such as a central directory service or a database of identity provider.
- Provisioning system detects a change in the information source. It pulls the change into the provisioning system. Usually a set of rules or scripts is executed to determine what to do.
- Provisioning system replicates the change to all the systems where it should be replicated. Accounts are created, updated or deleted as needed.
- Provisioning system remembers where are the copies of the data. This is crucial. This kind of meta-data can be used to correctly update all the copies if needed. It is also used to delete the data as necessary.

This is fundamentally different principle than those techniques used in access management technologies. It is not event-driven but data-driven. The trigger is not user access but it is the change in the data. Therefore it is much more reliable. Especially when combined with additional techniques such as reconciliation to increase the reliability. Some advanced provisioning systems also have a self-healing capability and can correct the data immediately after a problem is discovered. The data also does not need to be the same. They can be transformed and adjusted as they flow between the systems.

Identity synchronization brings substantial advantages:

- **Security**. Even if user profiles are copied to many systems provisioning system keeps track of them all. Every copy is recorded. The data can be automatically maintained. And this can be done in a consistent policy-based fashion. Also advanced security concepts such as segregation of duties can easily be enforced. As there is place that knows about all the data then it is easy to use such data for automated risk analysis.
- **Audit**. Provisioning system records the changes in the data: where the change originated, who caused it, where it was propagated. Therefore compliance is faster and considerable less costly.
- **Personal data protection and privacy**. Data copies are recorded. Therefore the provisioning system is able to remove the data completely from all the system if needed. Also all the data in all the system can be updated immediately making it easy correct data errors therefore reducing the impact of misleading or leaked data.
- **Time to market**. Applications are passive during data synchronization. Applications just wait for the data to be picked up or updated. Provisioning system is the active part. Therefore the applications usually do not need to be changed to integrate them into a solution. Integrating applications using provisioning requires less time and it is less expensive as compared to technologies based on access management technologies.

Provisioning technologies are a great success in some environments such as the enterprise, eGovernment and inside cloud solutions. But they cannot go alone into the Internet. It is clear that both *access* and *provisioning* parts of the solutions are essential for such solutions. However while it is quite well accepted that an Internet-based identity solution should have an *access* part the *provisioning* part is almost always neglected. But it is essential. The solution just cannot be complete without it.

In the past the provisioning systems used to be very expensive toys. And many of the products from that era still survive on the market. But there are also new products. These are built on newer technologies and on years of experience. These are not toys anymore and they are not expensive either. Some of the new products have business models that are feasible for management of large number of identities. These can be deployed in cloud solutions, used to manage customer identities, social network identities and so on. These products are built for the Internet age.

Example: Evolveum midPoint

Couple of years ago there was no suitable solution for the Internet scale. Provisioning was strictly confined to enterprise boundaries and it was very expensive. But all of that is over.

MidPoint is one of several open source identity provisioning systems. The project went through several years of very rapid development and created a very unique, flexible and efficient system. MidPoint is a next generation provisioning system. It is built on lightweight technologies that the Internet has produced. Technologically it has very little in common with its predecessors. However midPoint is designed to accomplish everything that the previous generations of products did - and much more. MidPoint comes with a very interesting pricing model especially suitable for cloud deployments, management of customer identities or similar large-scale deployments.

MidPoint has feature suitable for **Internet** environment:

- **Scalability** in both the engineering and business sense. Technologically midPoint is built on top database abstraction layer which allows it to run on non-traditional data stores such as noSQL databases. Business-wise midPoint comes with a pricing model that is very suitable for cloud solutions, federated deployments, Internet portals, service providers and so on.
- **Automation** is essential when dealing with large number of identities. MidPoint is designed to automate propagation of changes over the applications - and to do it quickly and efficiently. MidPoint can be deployed in a zero-administration mode which reduces the cost and time to market for new applications. MidPoint is **self-healing**. It can correct errors in data transparently as soon as they are detected.
- **REST** and SOAP interfaces are a norm. MidPoint interfaces provide a complete control over all the midPoint features. Therefore midPoint can be used as an part of larger solution or even as an embedded system.

There are also features essential for the **enterprise**:

- **Advanced RBAC** model that supports very complex structures with just a handful of roles. It is as close to **ABAC** as a provisioning system can go. This efficiently avoids role explosion problem.
- **Organizational structure** is natively supported in midPoint. It can be used for **delegated administration** but is also can be easily synchronized to other applications. Other objects such as groups can be synchronized as well.
- **Rich set of connectors** allow easy integration with existing applications. MidPoint can use connectors from several other projects and it also adds its own connectors.
- **Entitlements** can be directly managed by midPoint. MidPoint can manage group membership, access control lists (ACLs), privileges or similar fine-grain entitlements.

MidPoint is an **open source** system with a full commercial support. This is essential because:

- **Economic feasibility** is given by zero licensing cost and very reasonable subscription cost.
- **Flexibility** is inherent in midPoint. MidPoint configuration is very flexible and several scripting languages can be used as part of midPoint configuration. But it is also possible for a partner to modify midPoint source code. Unlike most software vendors we are able to provide support even for midPoint deployments with modified source code. Our partners are empowered rather than intimidated.
- **Innovative business** is possible. Our partners use midPoint to create cloud and SaaS deployments. There are plans for identity-in-the-box and pre-configured solutions. MidPoint allows business models that are simply not feasible with other type of software.

See Also

- Enterprise Identity Management

- Best Practice
- Antipatterns