**Federating Identities**

Consider the many times an individual accesses services on the Internet in a single day. At work, he uses the company intranet to perform a multitude of tasks including reading and sending email, looking up information in the company phone book, searching internal databases, and submitting expense reports and other online forms. At home, he checks personal email, logs into an online news service, finalizes travel plans via a travel agent’s web site, and shops. Each time he accesses one of these services, he must log in and identify himself.

A **local identity** refers to the set of attributes or information that identify the user to a particular service provider. These attributes typically include a name and password, plus an email address, account number or other identifier. Most users have many local identities. For example, the individual in our scenario might log in at work using an employee number but, at home, he might log in to his travel agent as Joe Smith. He might use an account number to log in to the car rental agency he uses frequently, and he might log in to an airline using a frequent flyer number.

Identity federation allows a user to consolidate the many local identities he has configured among multiple service providers. With a **federated identity**, the individual can log in at one service provider site and move to an affiliated service provider site without having to re-authenticate or re-establish his identity. For example, with a federated identity, the individual might want to access both his personal email account and his business email account from his workplace, and move back and forth between the two services without having to log in each time. Or at home he might want to log in to an online travel agency to book airline tickets and make hotel reservations. It is a convenience for the user to be able to access all of these services without having to provide different user names and passwords at each service site. It is a valuable benefit to the user when he can do so safely, knowing that his identity information is secure.

# How Federation Works

The goal of the Liberty Alliance Project specifications is to enable individuals and multiple organizations to easily conduct network transactions while protecting the individual’s identity. When organizations form a **circle of trust**, they agree to exchange user authentication information using web service technologies. A circle of trust must contain at least one identity provider, a service provider that maintains and manages identity information. It also includes multiple service providers that offer web-based services to users. Once a circle of trust is established, single sign-on is enabled between all the providers and users can federate their multiple identities.

In Access Manager, the circle of trust is referred to as an **authentication domain**. An authentication domain contains entities that are grouped together for the purpose of identity federation. A travel portal is a good example of an authentication domain. Typically, a travel portal is a web site designed to help you access various travel-related service providers from one location. The travel portal forms a partnership with each hotel, airline, and car rental agency displayed on its web site. The user registers with the travel portal which, in effect, is the authentication domain's identity provider. After logging in, the user looks for a flight using the airline service provider. After booking a flight, the user looks for a hotel using the accommodations service provider. This time, because of the agreements established among the travel portal partners, the airline web site shares the authentication information obtained earlier in the user's online session. The user moves from the hotel reservations web site to the airline reservations web site without having to re-authenticate. All of this is transparent to the user who must initially choose to unite his local identities. The following figure illustrates the travel portal example.

##### Figure 5–1 Federation Within a Travel Portal

This figure illustrates how a user's identity
can be shared among many businesses such as airlines, car rental agencies,
and hotels.

**Note –**

Account federation occurs when a user chooses to unite distinct service accounts and identity provider accounts. The user retains individual account information with each provider in the circle. At the same time, the user establishes a link that allows the exchange of authentication information between them. Users can choose to federate any or all identities they might have with the service providers. After account federation, when a user successfully authenticates with one service provider, he can access any of the his accounts within the authentication domain in a single session **without having to reauthenticate**.

# The Web Services Stack

In Access Manager, the Federation framework enables the secure exchange of authentication and authorization information by providing an interface for creating, modifying, and deleting authentication domains and configuring service and identity providers (both remote and hosted types) as entities. Additionally, the implemented web services define a stack to support the Federation framework. The following figure illustrates the architecture of the web services stack and how a web service consumer communicates with the web service provider (Access Manager).

##### Figure 5–2 Web Services Architecture

This figure illustrates the web services architecture
in Access Manager.

## Implemented Services

Access Manager includes the following web services based on the Liberty Alliance Project specifications:

Authentication Web Service

Provides authentication to a WSC, allowing the WSC to obtain security tokens for further interactions with other services at the same provider. Upon successful authentication, the final Simple Authentication and Security Layer (SASL) response contains the resource offering for the Discovery Service.

Discovery Service

A web service that allows a requesting entity, such as a service provider, to dynamically determine a principal's registered attribute provider. Typically, a service provider queries the Discovery Service, which responds by providing a **resource offering** that describes the requested attribute provider. The implementation of the Discovery Service includes Java and web-based interfaces.

SOAP Binding

A set of Java APIs used by the developer of a Liberty-enabled identity service. The APIs are used to send and receive identity-based messages using SOAP, an XML-based messaging protocol.

Liberty Personal Profile Service

A data service that supports storing and modifying a principal's identity attributes. Identity attributes might include information such as first name, last name, home address, and emergency contact information. The Liberty Personal Profile Service is queried or updated by a WSC acting on behalf of the principal.

## Web Services Process

The following figure provides a high-level view of the process between the various components in the web services stack. In this example:

* The web browser represents a user.
* The service provider also acts as a web services consumer (WSC), invoking a web service on behalf of the user. The service provider relies on the identity provider for authentication.
* The identity provider acts as an authentication provider by authenticating the user. It also acts as a trusted authority, issuing security tokens through the Discovery Service.
* The web services provider (WSP) serves requests from web services clients such as the Liberty Personal Profile Service.

##### Figure 5–3 Web Services Stack Process

Illustration depicting the web services process
in Access Manager.

The following process assume that the user, the identity provider, and the service provider have already been federated.

1. The user attempts to access a resource hosted on the service provider server.
2. The service provider redirects the user to the identity provider for authentication.
3. The identity provider authenticates the user successfully and sends the single sign-on assertion to the requesting service provider.
4. The service provider verifies the assertion and the user is issued a session token.
5. The service provider redirects the user to the requested resource.
6. The user requests access to another service hosted on the WSC server.

For example, it might need that value of an attribute from the user’s Liberty Personal Profile Service.

1. The WSC sends a query to the Discovery Service to determine where the user’s Liberty Personal Profile Service instance is hosted.

The WSC bootstraps the Discovery Service with the resource offering from the assertion obtained earlier.

1. The Discovery Service returns a response to the WSC containing the endpoint for the user’s Liberty Personal Profile Service instance and a security token that the WSC can use to access it.
2. The WSC sends a query to the Liberty Personal Profile Service instance.

The query asks for the user’s personal profile attributes, such as home phone number. The required authentication mechanism specified in the Liberty Personal Profile Service resource offering must be followed.

1. The Liberty Personal Profile Service instance authenticates and validates authorization for the requested user or the WSC, or both.

If user interaction is required for some attributes, the Interaction Service will be invoked to query the user for consents or for attribute values. The Liberty Personal Profile Service instance returns a response to the WSC after collecting all required data.

1. The WSC processes the Liberty Personal Profile Service response, and renders the service pages containing the information.

For detailed information about all these components, see the [Sun Java System Access Manager 7.1 Federation and SAML Administration Guide](https://docs.oracle.com/docs/cd/E19462-01/819-4674/index.html).

# SAML Service

SAML defines an eXtensible Markup Language (XML) framework to achieve interoperability across different vendor platforms that provide SAML assertions. SAML is an XML framework for exchanging security information over the Internet. Access Manager SAML Service consists of a web service interface, a SAML core component, and a SAML framework that web services can connect to.

The Access Manager SAML Service enables the following functionality:

* Users can authenticate against Access Manager and access trusted partner sites without having to reauthenticate. This single sign-on process independent of the process enabled by Access Manager user session management.
* Access Manager acts as a policy decision point (PDP), allowing external applications to access user authorization information for the purpose of granting or denying access to their resources.
* Access Manager acts as both an attribute authority (allowing trusted partner sites to query a subject’s attributes) and an authentication authority (allowing trusted partner sites to query a subject’s authentication information.)
* Two parties in different security domains can validate each other for the purpose of performing business transactions.
* Access Manager SAML APIs can be used to build Authentication, Authorization Decision and Attribute Assertions.
* The Access Manager SAML Service provides pluggable XML-based digital signature signing and verifying.

**Identity federation and OpenId**

Identity federation is a hot subject at this moment, at least for my customer.

It began innocently enough: someone who did not belong to the organization needed to access an application. “Not a problem”, let’s register him. Some year later, needs number in thousands of users, and the identity management has outgrown its capacities: the software was sized for the organization, not for the myriad of third-party users around it. And yet, partners, customers and providers, they all have very valid reasons to access internal applications, but still have to be identified and given rights (authentication and authorization), like any other user.

**“A user’s authentication process across multiple IT systems or even organizations.”**

I’d rather define it as a user’s authentication across multiple domains. Domain may refer indifferently to coarse-grained or fine-grained systems.

Other use-cases of identity federation include:

* a startup which has no desire to invest in any identity management software (acquired or developed), and prefer to delegate to a third-party
* an organization that need “fresh” information about identities will delegate to a third-party where this “freshness” is almost garanteed, whether for legal reasons or just because
* an decentralized organization which delegates authentication to its units but still needs its applications to be interoperable
* etc.

A great advantage of identity federation is that it’s the basis of [Single Sign-On](http://en.wikipedia.org/wiki/Single_sign-on). This feature becomes more and more in demand as the number of available applications grow: either users forget their login/password or they note it somewhere which defeats security.

Anyway, identity federation takes a greater importance nowadays, even if previous technologies provided some capabilities oriented toward it. I’m thinking about LDAP here, where you can host some branch of your LDAP tree on another LDAP. This is limited, however, because master and delegate must still share the same schema (if I remember well, I haven’t played with LDAP in a long time).

With such increased need toward identity federation, it’s only natural that there’s plenty of solutions around but no clear standard, either normalized or *de facto*. Available technologies include (but are not limited to): [CAS](http://www.jasig.org/cas), [Shibboleth](http://shibboleth.internet2.edu/) and [OpenId](http://openid.net/" \t "_blank). I recently became interested in the latter, not because it’s better than the others (I’m new in this field and have not enough input to pass such judgement) but because Google is an OpenId provider in front either of Google’s own id management or [Google Apps](http://www.google.com/apps/intl/en/group/index.html)‘s. This means you can manage your identities on Google infrastructure (and it’s free up to 50 accounts) and authentify your users with it using OpenId!

Authentication with OpenId is a multi-step process:

1. Application gets available authentication endpoints from a provider
2. Application redirects the user to one of such authentication endpoint
3. User authenticates himself on the provider infrastructure
4. Provider redirects the stream to the application if successful
5. Finally, application checks whether authentication was successful

Luckily, there’s a OpenSource project available in Java that nicely wraps the gritty details of OpenId inside an API. This project is aptly named [OpenId4Java](https://code.google.com/p/openid4java/) and works like a charm.

The first two steps are handled by the following code:

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
   
 try {  
 // API entry point  
 ConsumerManager manager = new ConsumerManager();  
   
 // Get available endpoints  
 List<?> discoveries = manager.discover("https://www.google.com/accounts/o8/id");  
   
 // Bind to endpoint  
 DiscoveryInformation discovered = manager.associate(discoveries);  
   
 // Create the auth request, providing return URL  
 AuthRequest authReq = manager.authenticate(discovered, request.getRequestURL().toString());  
   
 // Redirects to provider login page  
 response.sendRedirect(authReq.getDestinationUrl(true));  
   
 } catch (Exception e) {  
   
 throw new ServletException(e);  
 }  
}

Just trying the previous code will get our user to a Google’s login page! Once the user is authentified, he’s sent back to our application: we just have to manage the token to verify its integrity. OpenId4Java also does the trick (previous code must be slightly modified, you’ll find it in the sources):

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
   
 ParameterList openidResp = new ParameterList(request.getParameterMap());  
   
 DiscoveryInformation discovered = (DiscoveryInformation) request.getSession().getAttribute("discovered");  
   
 StringBuffer receivingURL = request.getRequestURL();  
   
 String queryString = request.getQueryString();  
   
 if (queryString != null && queryString.length() > 0) {  
   
 receivingURL.append("?").append(request.getQueryString());  
 }  
   
 try {  
   
 VerificationResult verification = manager.verify(receivingURL.toString(), openidResp, discovered);  
   
 Identifier verified = verification.getVerifiedId();  
   
 if (verified != null) {  
   
 AuthSuccess authSuccess = (AuthSuccess) verification.getAuthResponse();  
   
 request.getRequestDispatcher("/WEB-INF/page/welcome.jsp").forward(request, response);  
   
 } else {  
   
 request.getRequestDispatcher("/").forward(request, response);  
 }  
   
 } catch (Exception e) {  
   
 throw new ServletException(e);  
 }  
}

Even better, Google supports an OpenId extension, the Attribute Exchange extension (AX). This let us query information about the user from theOpenId provider, provided we get the user’s agreement, of course. OpenId4Java seamlessly integrates with such an extension, we just have to update somewhat both our request and our response:

FetchRequest fetch = FetchRequest.createFetchRequest();  
   
fetch.addAttribute("Email", "http://schema.openid.net/contact/email", true);  
fetch.addAttribute("FirstName", "http://axschema.org/namePerson/first", true);  
fetch.addAttribute("LastName", "http://axschema.org/namePerson/last", true);  
fetch.addAttribute("Country", "http://axschema.org/contact/country/home", true);  
fetch.addAttribute("Lang", "http://axschema.org/pref/language", true);  
   
authReq.addExtension(fetch);

if (authSuccess.hasExtension(OPENID\_NS\_AX)) {  
   
 MessageExtension ext = authSuccess.getExtension(OPENID\_NS\_AX);  
   
 if (ext instanceof FetchResponse) {  
   
 FetchResponse fetch = (FetchResponse) ext;  
   
 request.setAttribute("EMAIL", fetch.getAttributeValue("Email"));  
 request.setAttribute("FIRST\_NAME", fetch.getAttributeValue("FirstName"));  
 request.setAttribute("LAST\_NAME", fetch.getAttributeValue("LastName"));  
 request.setAttribute("COUNTRY", fetch.getAttributeValue("Country"));  
 request.setAttribute("LANG", fetch.getAttributeValue("Lang"));  
 }  
}

With OpenId and OpenId4Java, using identity federation is really a matter of minutes!

**Public clients vs Credential clients:**

Confidential clients are applications that are able to securely authenticate with the authorization server, for example being able to keep their registered client secret safe. Public clients are unable to use registered client secrets, such as applications running in a browser or on a mobile device.

**Refresh token is a long-lived special kind of token used to obtain a renewed access token.**

**ID token carries identity information encoded in the token itself, which must be a JWT**. It must not contain any authorization information, or any audience information — it is merely an identifier for the user

## [What is an OAuth 2.0 Grant Type?](https://developer.okta.com/blog/2018/04/10/oauth-authorization-code-grant-type#what-is-an-oauth-20-grant-type)

In OAuth 2.0, the term “grant type” refers to the way an application gets an access token. OAuth 2.0 defines several grant types, including the authorization code flow. OAuth 2.0 extensions can also define new grant types.

Each grant type is optimized for a particular use case, whether that’s a web app, a native app, a device without the ability to launch a web browser, or server-to-server applications.

## [The Authorization Code Flow](https://developer.okta.com/blog/2018/04/10/oauth-authorization-code-grant-type#the-authorization-code-flow)

The Authorization Code grant type is used by web and mobile apps. It differs from most of the other grant types by first requiring the app launch a browser to begin the flow. At a high level, the flow has the following steps:

* The application opens a browser to send the user to the OAuth server
* The user sees the authorization prompt and approves the app’s request
* The user is redirected back to the application with an authorization code in the query string
* The application exchanges the authorization code for an access token

The Password grant type is **a way to exchange a user's credentials for an access token**. Because the client application has to collect the user's password and send it to the authorization server, it is not recommended that this grant be used at all anymore.

“PKCE (RFC 7636) is **an extension to the Authorization Code flow to prevent several attacks and to be able to securely perform the OAuth exchange from public clients**.” The OAuth 2.0 spec is the industry standard protocol for authorization and allows users to grant permission for apps to access their Dropbox data