Sun Federated Access Manager 8.0 Technical Overview

Beta

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–3740–10
June 2008

List of Remarks

Contents

Preface

Sun Java™ System Access Manager is a component of the Sun Java Enterprise System (Java ES), a set of software components that provide services needed to support enterprise applications distributed across a network or Internet environment. The *Sun Java System Access Manager 7.1 Technical Overview* describes Access Manager features, explains what Access Manager does, and illustrates how Access Manager works.

Before You Read This Book

This book is intended for use by IT administrators and software developers who implement a web access platform using Sun Java System servers and software. Readers of this guide should be familiar with the following:

- Web containers in which Access Manager can be deployed:
  - Sun Java System Application Server
  - Sun Java System Web Server
  - BEA WebLogic
  - IBM WebSphere Application Server
- Technologies:
  - Lightweight Directory Access Protocol (LDAP)
  - Java
  - JavaServer Pages™ (JSP)
  - HyperText Transfer Protocol (HTTP)
  - HyperText Markup Language (HTML)
  - eXtensible Markup Language (XML)
  - SOAP
  - HyperText Transfer Protocol (HTTP)
  - Liberty Alliance Project specifications

Related Books

Related documentation is available as follows:

Access Manager Core Documentation

The Access Manager core documentation set contains the following titles:

- The *Sun Java System Access Manager 7.1 Release Notes* will be available online after the product is released. It gathers an assortment of last-minute information, including a description of what is new in this current release, known problems and limitations, installation notes, and how to report issues with the software or the documentation.
- The *Sun Federated Access Manager 8.0 Technical Overview* (this guide) provides an overview of how Access Manager components work together to protect enterprise assets and web-based applications. It also explains basic Access Manager concepts and terminology.
- The *Sun Java System Access Manager 7.1 Deployment Planning Guide* provides planning and deployment solutions for Sun Java System Access Manager based on the solution life cycle
- The *Sun Java System Access Manager 7.1 Postinstallation Guide* provides information for configuring Access Manager after running the Java ES installer.
- The *Sun Java System Access Manager 7.1 Performance Tuning and Troubleshooting Guide* provides information on how to tune Access Manager and its related components for optimal performance.
- The *Sun Java System Access Manager 7.1 Administration Guide* describes various administrative tasks such as Realms Management, Policy Management, Authentication and Directory Management. Most of the tasks described in this book are performed through the Access Manager console as well as through the command line utilities.
- The *Sun Java System Access Manager 7.1 Administration Reference* is a look-up guide containing information about the command line interfaces, configuration attributes, Access Manager files, and error codes.
- The *Sun Java System Federated Access Manager 8.0 Developer's Guide* offers information on how to customize Access Manager and integrate its functionality into an organization's current technical infrastructure. It also contains details about the programmatic aspects of the product and its API.
- The *Sun Java System Federated Access Manager 8.0 C API Reference* provides summaries of data types, structures, and functions that make up the public Access Manager C APIs.
- The *Federated Access Manager 8.0 Java API Reference* provides information about the implementation of Java packages in Access Manager.

- The *Sun Java System Federated Access Manager Policy Agent 3.0 User's Guide* provides an overview of the policy functionality and the policy agents available for Federated Access Manager.

Updates to the *Release Notes* and links to modifications of the core documentation can be found on the Access Manager page at the Sun Java Enterprise System documentation web site. Updated documents will be marked with a revision date.

Agent Documentation

Useful information can be found in the documentation for the following products:

- Sun Java System Directory Server Enterprise Edition 6.0
- Sun Java System Web Server 7.0
- Sun Java System Application Server Enterprise Edition 8.2
- Sun Java System Web Proxy Server 4.0.4

Adjunct Product Documentation

A full list of the Java Enterprise System documentation is documented in the following table.

TABLE P–1 Sun Java Enterprise System Documentation Listing

Document Title	Contents
Sun Java Enterprise System 5 Release Notes for UNIX Sun Java Enterprise System 5 Release Notes for Microsoft Windows	Contains the latest information about Java ES, including known problems. In addition, components have their own release notes listed in the Release Notes Collection.
Sun Java Enterprise System 5 Update 1 Technical Overview	Introduces the technical and conceptual foundations of Java ES. Describes components, the architecture, processes, and features.
Sun Java Enterprise System 2006Q3 Deployment Planning Guide	Provides an introduction to planning and designing enterprise deployment solutions based on Java ES. Presents basic concepts and principles of deployment planning and design, discusses the solution life cycle, and provides high-level examples and strategies to use when planning solutions based on Java ES.
Sun Java Enterprise System 5 Installation Planning Guide	Helps you develop the implementation specifications for the hardware, operating system, and network aspects of your Java ES deployment. Describes issues such as component dependencies to address in your installation and configuration plan.

TABLE P–1 Sun Java Enterprise System Documentation Listing *(Continued)*

Document Title	Contents
Sun Java Enterprise System 5 Installation Guide for UNIX Sun Java Enterprise System 5 Installation Guide for Microsoft Windows	Guides you through the process of installing Java ES. Also shows how to configure components after installation, and verify that they function properly.
Sun Java Enterprise System 5 Installation Reference for UNIX	Gives additional information about configuration parameters, provides worksheets to use in your configuration planning, and lists reference material such as default directories and port numbers on the Solaris Operating System and Linux operating environment.
Sun Java Enterprise System 2006Q3 Upgrade Guide Sun Java Enterprise System 5 Upgrade Guide for Microsoft Windows	Provides instructions for upgrading to Java ES 5 from previously installed versions.
Sun Java Enterprise System 5 Monitoring Guide	Gives instructions for setting up the Monitoring Framework for each product component and using the Monitoring Console to view real-time data and create monitoring rules.
Sun Java Enterprise System Glossary	Defines terms that are used in Java ES documentation.

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com[SM] web site, you can use a search engine by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for "broker," type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use `sun.com` in place of `docs.sun.com` in the search field.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (http://www.sun.com/documentation/)
- Support (http://www.sun.com/support/)
- Training (http://www.sun.com/training/)

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to http://docs.sun.com and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the title of this book is *Sun Java System Access Manager 7.1 Technical Overview*, and the part number is 819–4669–10.

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–2 Typographic Conventions

Typeface	Meaning	Example
AaBbCc123	The names of commands, files, and directories, and onscreen computer output	Edit your .login file. Use ls -a to list all files. machine_name% you have mail.

TABLE P–2 Typographic Conventions *(Continued)*

Typeface	Meaning	Example
AaBbCc123	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Shell Prompts in Command Examples

The following table shows default system prompts and superuser prompts.

TABLE P–3 Shell Prompts

Shell	Prompt
C shell on UNIX and Linux systems	`machine_name%`
C shell superuser on UNIX and Linux systems	`machine_name#`
Bourne shell and Korn shell on UNIX and Linux systems	`$`
Bourne shell and Korn shell superuser on UNIX and Linux systems	`#`
Microsoft Windows command line	`C:\`

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–4 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.

TABLE P–4 Symbol Conventions *(Continued)*

Symbol	Description	Example	Meaning
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

PART I

An Overview of Federated Access Manager

This part of the Sun Federated Access Manager Technical Overview contains introductory material concerning Federated Access Manager. It includes the following sections:

◆ ◆ ◆ CHAPTER 1

Introducing Federated Access Manager

Sun Java™ System Federated Access Manager integrates authentication and authorization services, single sign-on, and open, standards-based federation protocols to provide a comprehensive solution for protecting network resources by preventing unauthorized access to web services, applications and web content, and securing identity data. This introductory chapter contains a high-level description of Federated Access Manager and what it does. It contains the following sections:

What is Federated Access Manager?

Sun Java System Federated Access Manager is a single product that combines the features of Sun Java System Access Manager, Sun Java System Federation Manager, and the Sun Java System SAML v2 Plug-in for Federation Services; additionally, it is enhanced with new functionality developed specifically for this release. Federated Access Manager provides access management by allowing the implementation of authentication, policy-based authorization, federation, SSO, and web services security from a single, unified framework. The core application is delivered as a simple web archive (WAR) that can be easily deployed in a supported web container.

Note – Federated Access Manager is Sun Microsystems' commercial distribution of the open source code available at OpenSSO.

To assist the core application, policy agents, the Client SDK, and (possibly) other disparate pieces must be installed remotely and be able to communicate with the Federated Access

Manager server. See "What Does Federated Access Manager Do?" on page 22 for a high-level explanation of the deployment architecture and Chapter 2, "Dissecting Federated Access Manager," for more specific information.

What Does Federated Access Manager Do?

The following types of interactions occur daily in a corporate environment.

- An employee looks up a colleague's phone number in the corporate phone directory.
- A manager retrieves employee salary histories to determine an individual's merit raise.
- An administrative assistant adds a new hire to the corporate database, triggering the company's health insurance provider to add the new hire to its enrollment.
- An engineer sends an internal URL for a specification document to another engineer who works for a partner company.
- A customer logs into a company's web site and looks for a product in their online catalog.
- A vendor submits an invoice to the company's accounting department.
- A corporate human resources administrator accesses an outsourced benefits application.

For each of these transactions, the company must determine who is allowed to view the information or use the application. Some information such as product descriptions and advertising can be made available to everyone in a public online catalog. Other information such as accounting and human resources data must be restricted to employees only. And other sensitive information such as pricing models and employee insurance plans is appropriate to share only with partners, suppliers, and employees. This need for access determination is met by Sun Java System Federated Access Manager, an access management product with authentication, authorization, and single sign-on (SSO) services provided out of the box.

When a user or an external application requests access to content stored on a company's server, a *policy agent* (available in a separate download and installed on the same machine as the resource you want to protect) intercepts the request and directs it to Federated Access Manager which, in turn, requests credentials (such as a username and password in the case of a user) for authentication. If the credentials returned match those stored in the appropriate identity data store, Federated Access Manager determines that the user is authentic. Following authentication, access to the requested content is determined by the policy agent which evaluates the policies associated with the authenticated identity. Policies are created using Federated Access Manager and identify which identities are allowed to access a particular resource, specifying the conditions under which this authorization is valid. Based upon the results of the policy evaluation, the policy agent either grants or denies the user access. Figure 1–1 illustrates a high-level deployment architecture of Federated Access Manager.

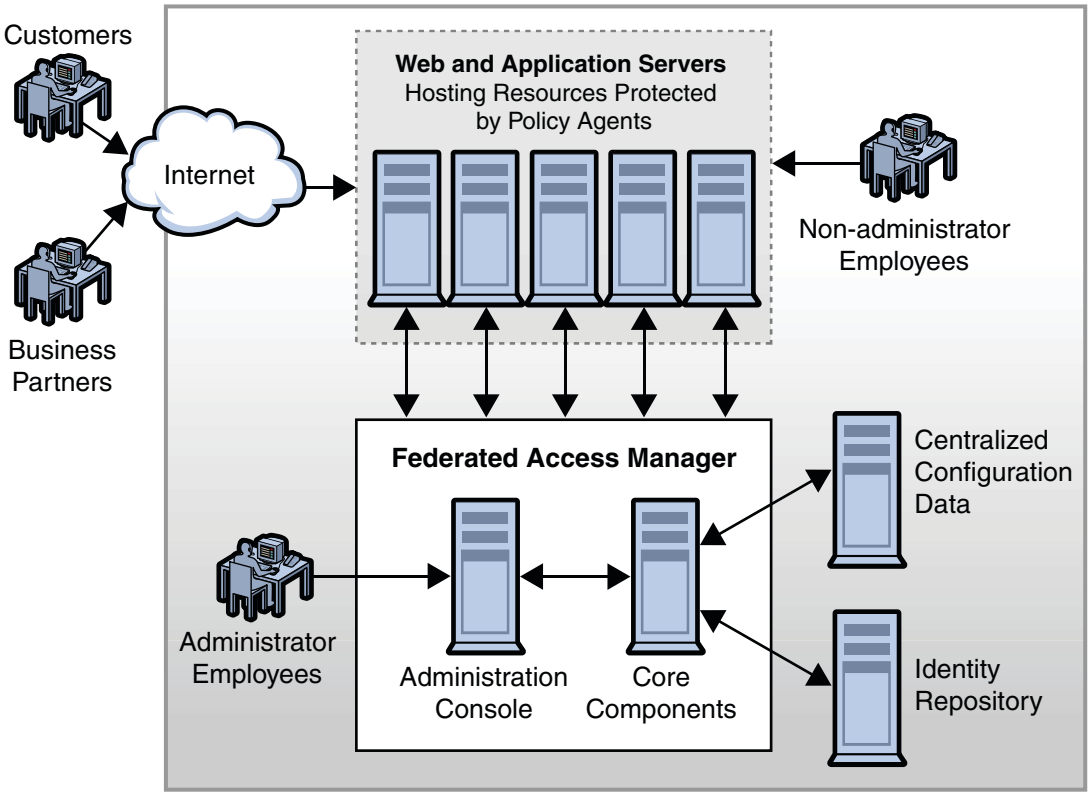


FIGURE 1–1 High-level Deployment Architecture of Federated Access Manager

What Are the Functions of Federated Access Manager?

The following sections contain an overview of the functions of Federated Access Manager.

Access Management

Federated Access Manager manages authorized access to network services and resources. By implementing authentication and authorization, Federated Access Manager (along with an installed policy agent) ensures that access to protected resources is restricted to authorized users. In a nutshell, a policy agent intercepts a request for access to a resource and communicates with Federated Access Manager to authenticate the requestor. If the user is

successfully authenticated, the policy agent then evaluates the policies associated with the requested resource and the user to determine if the authenticated user is authorized to access the resource. If the user is authorized, the policy agent allows access to the resource, also providing identity data to the resource to personalize the interaction.

For more information on access management, see Part II.

Federation Management

With the introduction of federation protocols into the process of access management, identity information and entitlements can be communicated across security domains, spanning multiple trusted partners. By configuring a *circle of trust* and defining applications and services as *entity providers* in the circle (either identity providers or service providers), users can opt to associate, connect or bind the various identities they have configured locally for these providers. The linked local identities are federated and allow the user to log in to one identity provider site and click through to an affiliated service provider site without having to reauthenticate; in effect, single sign-on (SSO). Federated Access Manager supports several open federation technologies (including the Security Access Markup Language [SAML] versions 1 and 2, WS-Federation, and the Liberty Alliance Project Identity Federation Framework (Liberty ID-FF), therefore encouraging an interoperable infrastructure among providers. See Part III for more information.

Web Services and Web Services Security

A *web service* is a component service or application that exposes some type of business or infrastructure functionality through a language-neutral and platform-independent, callable interface; enterprises might use this web service to build larger service-oriented architectures. In particular, the service defines its interface using the Web Services Description Language (WSDL), and communicates using SOAP and eXtensible Markup Language (XML) messages. The web service client (WSC) communicates with the web service provider (WSP) through an intermediary — usually a firewall or load balancer.

Note – The message being exchanged is defined by the WSDL as published by the WSP.

Although web services enable open, flexible, and adaptive interfaces, its openness creates security risks. Without proper security protections, a web service can expose vulnerabilities that might have dire consequences. Hence, ensuring the integrity, confidentiality and security of web services through the application of a comprehensive security model is critical for both enterprises and consumers. A successful security model associates identity data with the web services and creates secure service-to-service interactions. The security model adopted by Federated Access Manager identifies the user and preserves that identity through multiple

interactions, maintains privacy and data integrity, uses existing technologies, and logs the interactions. In Federated Access Manager, the following web service security standards are implemented:

- Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF)
- WS-I Basic Security Profile
- WS-Trust (from which the Security Token Service was developed)

See Part IV for more information.

Identity Services

Federated Access Manager also exposes its back-end access management functions as simple identity services that are accessible using SOAP, WSDL and REST. This allows developers to easily invoke them when developing an application using one of the supported integrated development environments (IDE). This access management solution does not require deployment of an agent or a proxy and includes the following capabilities:

- Authentication
- Authorization
- Provisioning
- Logging

For more information on access management and the available identity services, see Part II.

What Else Does Federated Access Manager Offer?

Federated Access Manager allows for:

1. **Ease of Deployment:** Federated Access Manager is delivered as a WAR that can be easily deployed as a Java EE application in different web containers. All configuration files and required libraries are inside the WAR to avoid the manipulation of the classpath in the web container's configuration file. The Federated Access Manager WAR is supported on:
   - Sun Java System Web Server 7.0 — Update 1 &2
   - Sun Java System Application Server 9.1 (and Glassfish v2)
   - BEA WebLogic Application Server 9.2 &10
   - IBM WebSphere Application Server 6.1
   - Oracle Application Server 10g
   - JBoss 4.2.x
   - Tomcat 5.5.x & 6.x
   - Geronimo (supported on the Sun Solaris™ 10 Operating Environment for SPARC, x86 & x64 and the Sun Solaris 9 Operating Environment for SPARC & x86 systems only)

Note – Geronimo can install Jetty Application Server and Tomcat Application Server; Federated Access Manager supports only Tomcat.

See the XXXXXX FAM8 Release Notes for updates to this list.

2. **Portability:** Federated Access Manager is supported on the following operating systems:
   - Sun Solaris 10 Operating Environment for SPARC, x86 & x64 systems
   - Sun Solaris 9 Operating Environment for SPARC & x86 systems
   - Windows Server 2003 and Windows XP (development only) operating systems
   - Red Hat Enterprise Linux 4 Server (Base)
   - Red Hat Enterprise Linux 4 Advanced Platform
   - Red Hat Enterprise Linux 5 Server (Base)
   - Red Hat Enterprise Linux 5 Advanced Platform
   - Windows 2003 Standard Server
   - Windows 2003 Enterprise Server
   - Windows 2003 Datacenter Server
   - Windows Vista
   - IBM AIX 5.3 (supported with the IBM Websphere Application Server 6.1 container only)
3. **Open Standards:** Federated Access Manager is built using open standards and specifications as far as possible. For example, features designed for federation management and web services security are based on the Security Assertion Markup Language (SAML), the Liberty Alliance Project specifications, and the WS-Security standards.
4. **Ease of Administration:** Federated Access Manager contains a web-based, graphical administration console as well as command line interfaces for configuration tasks and administrative operations. Additionally, an embedded, centralized data store allows for one place to store server and agent configuration data.
5. **Security:**
   - Runtime security enables an enterprise's resources to be protected as configured and Federated Access Manager services to be accessed by authorized entities only.
   - Administration security ensures only authorized updates are made to the Federated Access Manager configuration data.
   - Deployment security implements best practices for installing Federated Access Manager on different operating systems, web containers, and so forth.

   Additionally, all security actions are logged.

6. **Embedded Configuration Data Store:** Federated Access Manager writes server configuration data to an embedded configuration data store. You can also point to instances of Sun Java System Directory Server 5.1, 5.2 & 6.2 during configuration of Federated Access Manager for use as a configuration data store. See "Data and Data Stores" on page 53 for more information.
7. **User Data Store Independence:** Federated Access Manager allows you to view and retrieve user information without making changes to an existing user database. Supported directory servers include Directory Server 5.1, 5.2 & 6.2, Tivoli Directory Server 6.1, and Microsoft Active Directory 2003. See "Data and Data Stores" on page 53 for more information.

Note – The configuration data store embedded with Federated Access Manager should only be used as a user data store for proof of concepts and deployments in development.

---

8. **Web and Non-Web-Based Resources:** The core design of Federated Access Manager caters to SSO for both web and non-web applications.
9. **Performance, Scalability and Availability:** Federated Access Manager can be scaled horizontally and vertically to handle increased workloads, and as security needs change over time. There is no single point of failure.
10. **Distributed Architecture** Server and client components can be deployed across the enterprise or across domain boundaries as all application programming interfaces (API) provide remote access to Federated Access Manager based on a service-oriented architecture.
11. **Flexibility and Extensibility:** Many Federated Access Manager services expose a service provider interface (SPI) allowing expansion of the framework to provide for specific deployment needs.
12. **Internationalization** Federated Access Manager contains a framework for multiple language support. Customer facing messages, API, command line interfaces, and user interfaces are localized in the supported languages.

◆ ◆ ◆ CHAPTER 2

Dissecting Federated Access Manager

Federated Access Manager provides a pluggable architecture to deliver access management, secure web services, and federation capabilities. This chapter contains information on the internal architecture and features of Federated Access Manager.

Federated Access Manager Architecture

Federated Access Manager is written in Java, and leverages industry standards, including the HyperText Transfer Protocol (HTTP), the eXtensible Markup Language (XML), the Security Assertion Markup Language (SAML), and SOAP, to deliver access management, secure web services, and federation capabilities in a single deployment. It consists of client application programming interfaces (a Client SDK), a framework of services that implement the business logic, and service provider interfaces (SPI) that are implemented by concrete classes and can be used to extend the functionality of Federated Access Manager as well as retrieve information from data stores. Figure 2–1 illustrates the internal architecture of Federated Access Manager.

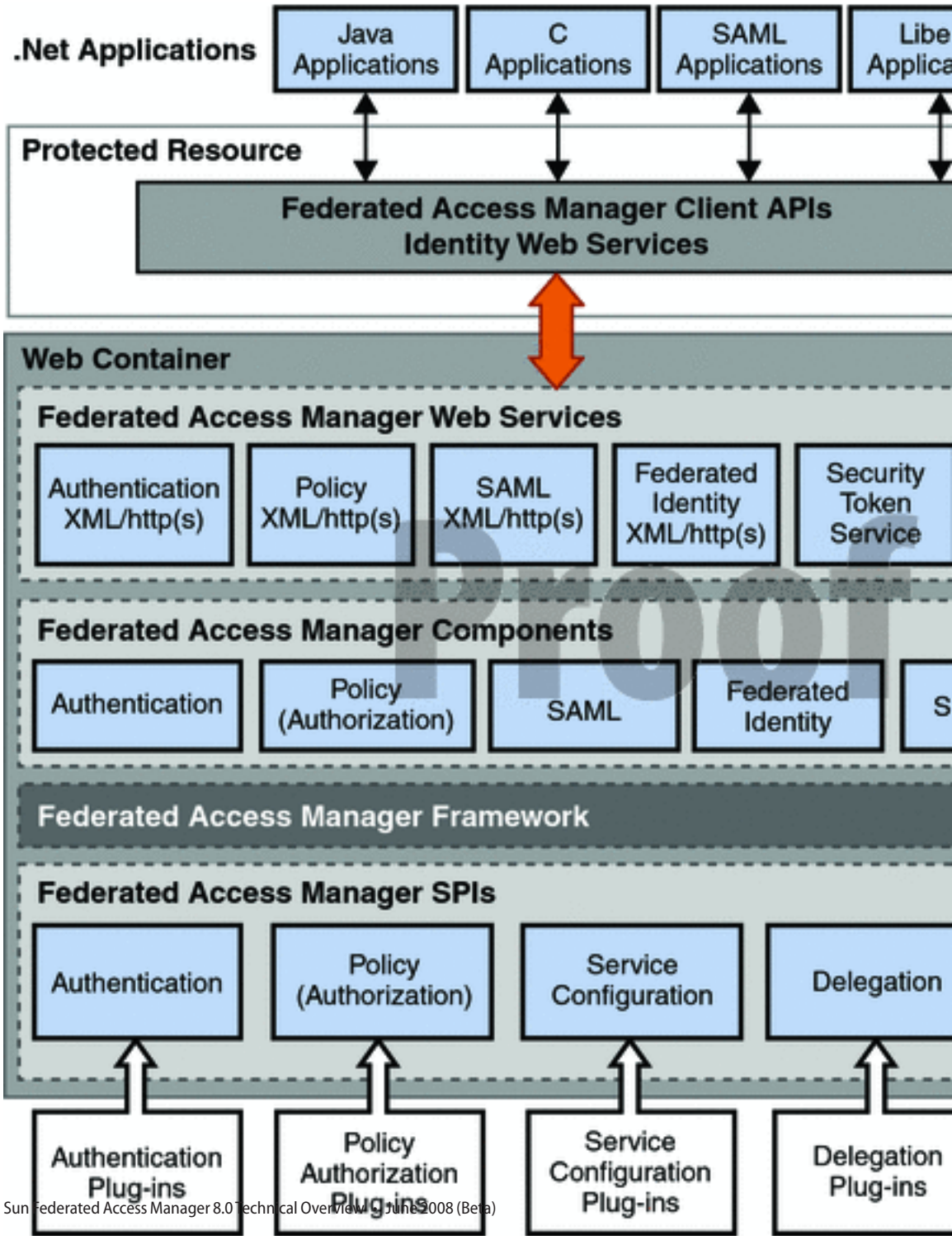


FIGURE 2–1 Internal Architecture of Federated Access Manager

Each component of Federated Access Manager uses its own framework to retrieve customer data from the plug-in layer and to provide data to other components. The Federated Access Manager framework integrates all of the business logic into one layer that is accessible to all components and plug-ins. The Client SDK and Identity Web Services are installed on a machine remote to the Federated Access Manager server that holds the resource to be protected. (The policy agent, also installed on the remote machine, is basically a client written using the Client SDK and Identity Web Services.) Applications on the remote machine access Federated Access Manager using the Client SDK. Custom plug-in modules are installed on the machine local to Federated Access Manager and interact with the Federated Access Manager SPI to retrieve required information from the appropriate data store and deliver it to the plug-ins and, in turn, the Federated Access Manager framework for processing.

How Federated Access Manager Works

When Federated Access Manager starts up, it is initialized with configuration data from various service plug-ins including those for the Policy Service, the Identity Repository Service, as well as the embedded service configuration data store. When someone (using a browser) sends an HTTP request for access to a protected resource, a policy agent (separately downloaded and installed on the same machine as the resource you want to protect) intercepts the request and examines it. If no valid session token is found, the policy agent contacts Federated Access Manager which will then invoke the authentication and authorization processes. Figure 2–2 illustrates one way in which the policy agents can be situated to protect an enterprise's servers by directing HTTP requests to a centralized Federated Access Manager for processing.

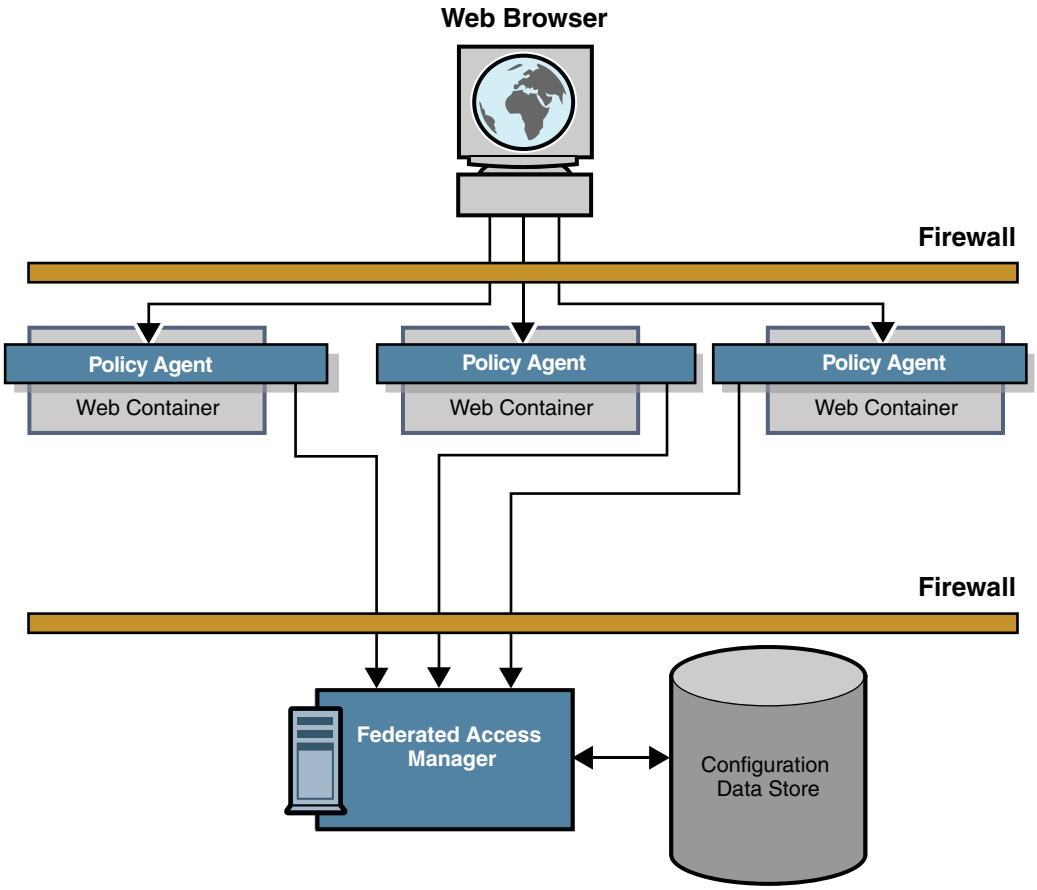


FIGURE 2–2 Basic Federated Access Manager Deployment

Federated Access Manager integrates core features such as access management and authorization. These functions can be configured using the administration console. Figure 2–3 is a high-level illustration of the interactions that occur between Federated Access Manager, a policy agent, browser, and protected application during authentication and authorization.

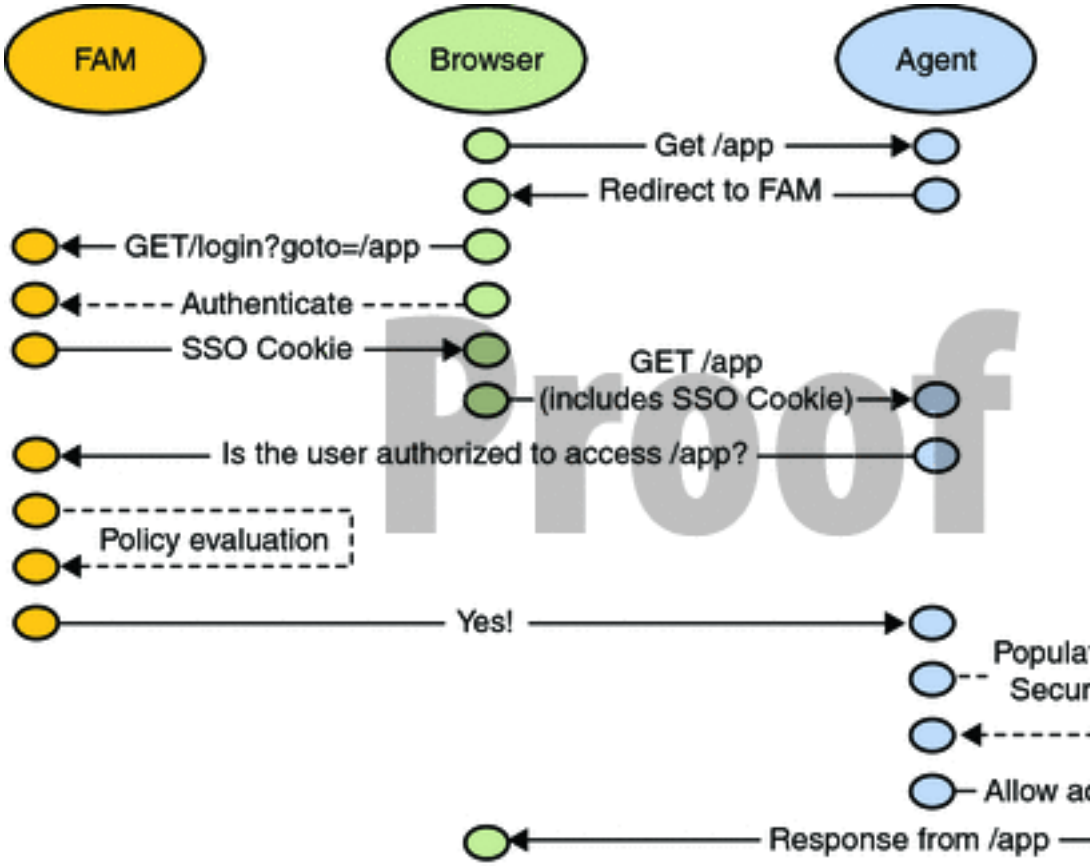


FIGURE 2–3 Federated Access Manager Authentication and Authorization Interactions

For more information on the core functions of Federated Access Manager, see "Core Services" on page 33.

Core Services

Services developed for Federated Access Manager generally contain both a server component and a client component. The server component is a simple Java servlet developed to receive XML requests and return XML responses. (The deployment descriptor `web.xml` defines the servlet name and description, the servlet class, initialization parameters, mappings, and other startup information.) The client component is provided as Java application programming interfaces (API), and in some cases C API, that allow remote applications and other Federated Access Manager services to communicate with and consume the particular functionality.

Each core service uses its own framework to retrieve customer and service data and to provide it to other Federated Access Manager services. The Federated Access Manager framework integrates all of these service frameworks to form a layer that is accessible to all product components and plug-ins. The following sections contain information on the Federated Access Manager core services.

- "Authentication Service" on page 34
- "Policy Service" on page 37
- "Session Service" on page 39
- "Logging Service" on page 42
- "Identity Repository Service" on page 44
- "Federation Services" on page 45
- "Web Services" on page 48
- "Web Services Security" on page 49
- "Identity Services" on page 50

Note – Many services also provide a public SPI that allows the service to be extended. See the *Sun Java System Federated Access Manager 8.0 Developer's Guide*, the *Sun Java System Federated Access Manager 8.0 C API Reference*, and the *Federated Access Manager 8.0 Java API Reference* for information.

Authentication Service

The Authentication Service provides the functionality to request user credentials and validate them against a specified authentication data store. Upon successful authentication, it creates a session data structure for the user that can be validated across all web applications participating in an SSO environment. Several authentication modules are supplied with Federated Access Manager, and new modules can be plugged-in using the Java Authentication and Authorization Service (JAAS) SPI.

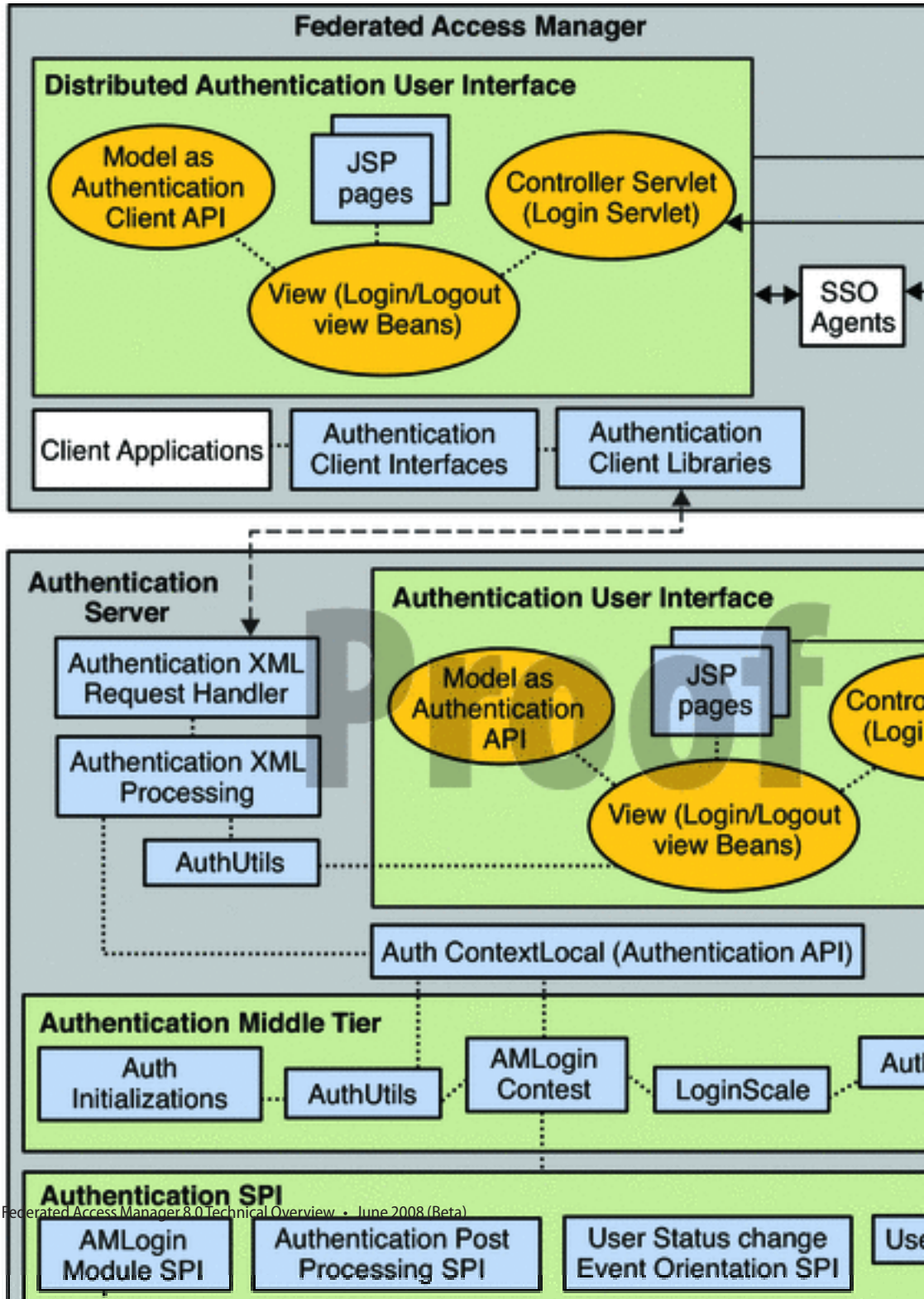
Note – The Authentication Service is based on the JAAS specification, a set of API that enables services to authenticate and enforce access controls upon users. See the Java Authentication and Authorization Service Reference Guide for more information.

Components of the Authentication Service include:

- The Distributed Authentication User Interface allows the Authentication Service user interface to be deployed separately from Federated Access Manager, if desired. By deploying this authentication proxy in the DMZ and using the authentication interfaces provided in the Client SDK to pass user credentials back and forth, you can protect Federated Access Manager data (for example, the login URL information and hence the host information). JavaServer Pages (JSP) represent the interface displayed to users for authentication and are completely customizable.

- The Core Authentication Service executes common processes across all authentication modules. Key responsibilities of this service include identification of the appropriate plan to authenticate the user (identify the authentication module, load the appropriate JSP) and creation of the appropriate session for the authenticated user.
- The Authentication API are *remoteable* interfaces that don't need to reside on the same machine as the Federated Access Manager server. This allows remote clients to access the Authentication Service. remote-auth.dtd defines the structure for the XML communications that will be used by the Authentication Service, providing definitions to initiate the process, collect credentials and perform authentication.
- A number of authentication modules are installed and configured (including, but not limited to, LDAP, Unix, Windows Desktop, Certificate, and Active Directory). A configured authentication level for each module is globally defined. Mechanisms are also provided to upgrade a user's session after authenticating the user to an additional authentication module that satisfies the authentication level of the resource. New modules can be plugged-in using the JAAS SPI.

The Authentication Service interacts with both the database that stores user credentials (authentication data store) to validate the user, and with the Identity Repository Service plug-ins to retrieve user profile attributes. When the Authentication Service determines that a user's credentials are genuine, a valid user session token is issued, and the user is said to be *authenticated*. The following figure illustrates how the authentication subcomponents interact within the infrastructure.


Federated Access Manager
Distributed Authentication User Interface
Model as Authentication Client API
JSP pages
Controller Servlet (Login Servlet)
View (Login/Logout view Beans)
SSO Agents
Client Applications
Authentication Client Interfaces
Authentication Client Libraries
Authentication Server
Authentication XML Request Handler
Authentication XML Processing
AuthUtils
Authentication User Interface
Model as Authentication API
JSP pages
View (Login/Logout view Beans)
Auth ContextLocal (Authentication API)
Authentication Middle Tier
Auth Initializations
AuthUtils
AMLogin Contest
LoginScale
Authentication SPI
AMLogin Module SPI
Authentication Post Processing SPI
User Status change Event Orientation SPI

More information on the architecture of the Authentication Service can be found in the Authentication Service Architecture document on the OpenSSO web site.

Policy Service

Authorization is the process with which Federated Access Manager evaluates the policies associated with an authenticated user's identity, and determines whether the user has permission to access a protected resource. (A *policy* defines the rules that specify a user's access privileges to a protected resource.) The Policy Service provides the authorization functionality using a rules-based engine. It interacts with the Federated Access Manager configuration data store, a delegation plug-in (which helps to determine a network administrator's scope of privileges), and Identity Repository Service plug-ins to verify that the user has access privileges from a recognized authority. Policy is configured using the Administration Console, and comprises the following:

- A *Schema* for the policy type (normal or referral) that describes the syntax of policy.
- A *Rule* which defines the policy itself and is made up of a *Resource*, an *Action* and a *Value*.
- *Condition(s)* to define constraints on the policy.
- *Subject(s)* to define the user or collection of users which the policy affects.
- A *ResponseProvider(s)* to send requested attribute values, typically based on the user profile, with the policy decision.

Figure 2–5 illustrates the framework of the Policy Service. Note that the `PolicyServiceRequestHandler` maps to the `PolicyRequest` XML element.

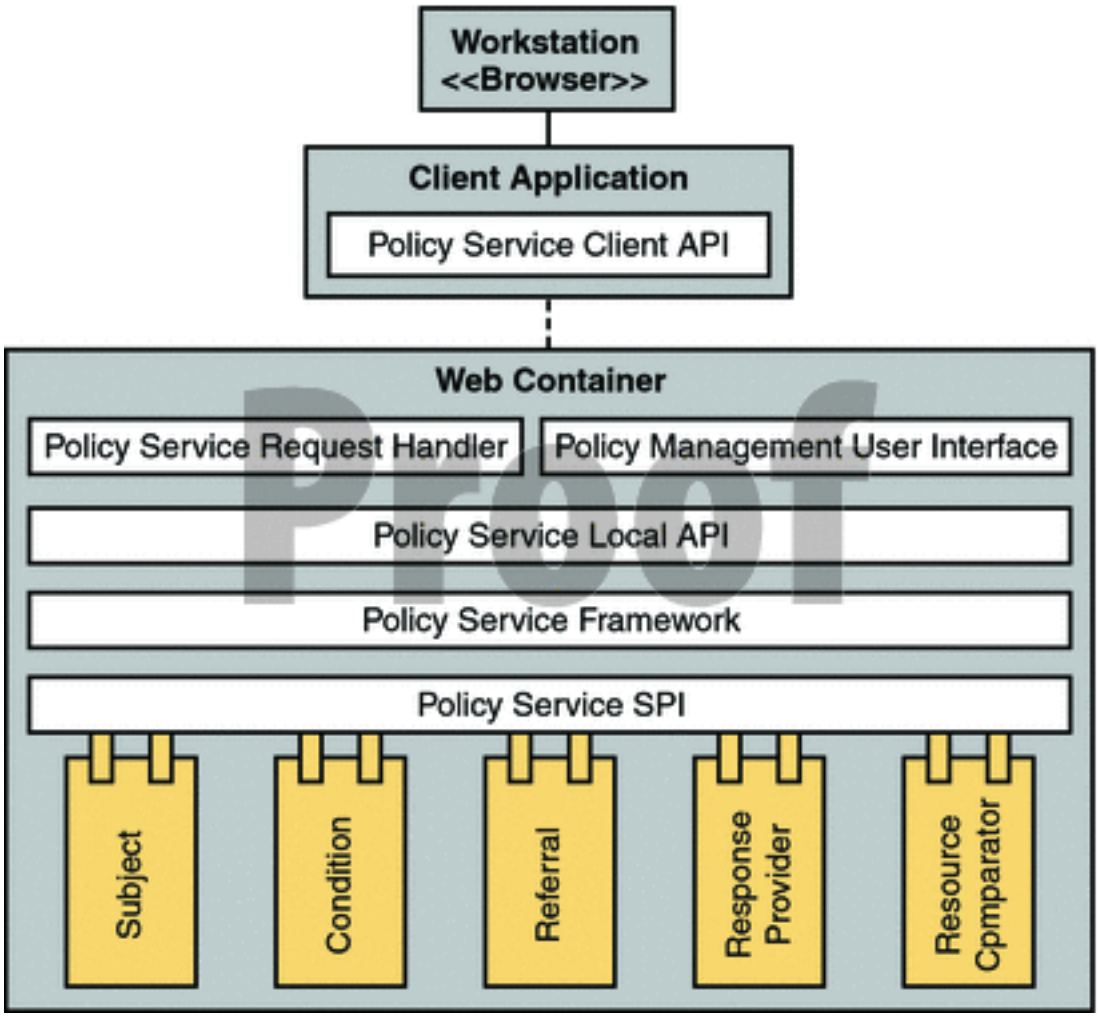


FIGURE 2–5 Policy Components within the Policy Service Framework

Policy agents are an integral part of authorization. They are programs, available for installation separate from Federated Access Manager, that police the web container which hosts the protected resources. When a user requests access to the protected resource (such as a server or an application), the policy agent intercepts the request and redirects it to the Federated Access Manager Authentication Service. Following authentication, the policy agent will enforce the authenticated user's assigned policies. Federated Access Manager supports two types of policy agents:

- The *web agent* enforces URL-based policy for C applications.

- The *Java Platform, Enterprise Edition (Java EE) agent* enforces URL-based policy and Java EE-based policy for Java applications on Java EE containers.

Note – When policy agents are implemented, all HTTP requests are implicitly denied unless explicitly allowed by the presence of two things:

1. A valid session
2. A policy allowing access

Note – If the resource is in the Not Enforced list defined for the policy agent, access is allowed even if there is no valid session.

For an overview of the available policy agents and links to specific information on installation, see the *Sun Java System Federated Access Manager Policy Agent 3.0 User's Guide*.

Session Service

The mission of the Federated Access Manager Session Service is to maintain information about an authenticated user's session across all web applications participating in a user session. Additionally, Federated Access Manager provides continuous proof of the user's identity, enabling the user to access multiple enterprise resources without having to provide credentials each time. This enables the following types of user sessions.

- **Basic user session.** The user provides credentials to log in to one application, and then logs out of the same application.
- **SSO session.** The user provides credentials once, and then accesses multiple applications within the same DNS domain.
- **Cross domain SSO (CDSSO) session.** The user provides credentials once, and then accesses applications among multiple DNS domains.

A *user session* is the interval between the time a user successfully authenticates through Federated Access Manager and is issued a session token, and the moment the user logs out of the session. In what might be considered a typical user session, an employee accesses the corporate benefits administration service. The service, monitored by Federated Access Manager, prompts the user for a username and password. With the credentials Federated Access Manager can *authenticate*, or verify that the user is who he says he is. Following authentication, Federated Access Manager allows the user access to the service providing authorization is affirmed. Successful authentication through Federated Access Manager results in the creation of a *session data structure* for the user or entity by the Session Service. Generally speaking, the Session Service performs some or all of the following:

- Generates unique session identifiers, one for each user's session data structure

---

Note – A session data structure is initially created in the `INVALID` state with default values for certain attributes and an empty property list. Once the session is authenticated, the session state is changed to `VALID` and session data is updated with the user's identity attributes and properties.

- Maintains a master copy of session state information

Note – The session state maintained on the client side is a cached view of the actual session data structure. This cache can be updated by either the active polling mechanism or the session notification triggered by the Session Service.

- Implements time-dependent behavior of sessions — for example, enforces timeout limits
- Implements session life cycle events such as logout and session destruction
- Notifies all participants in the same SSO environment of session state changes
- Enables SSO and cross-domain single sign-on (CDSSO) among applications external to Federated Access Manager by providing continued proof of identity.
- Allow participating clients to share information across deployments
- Implement high availability facilities

Figure 2–6 illustrates the components of the Session Service.

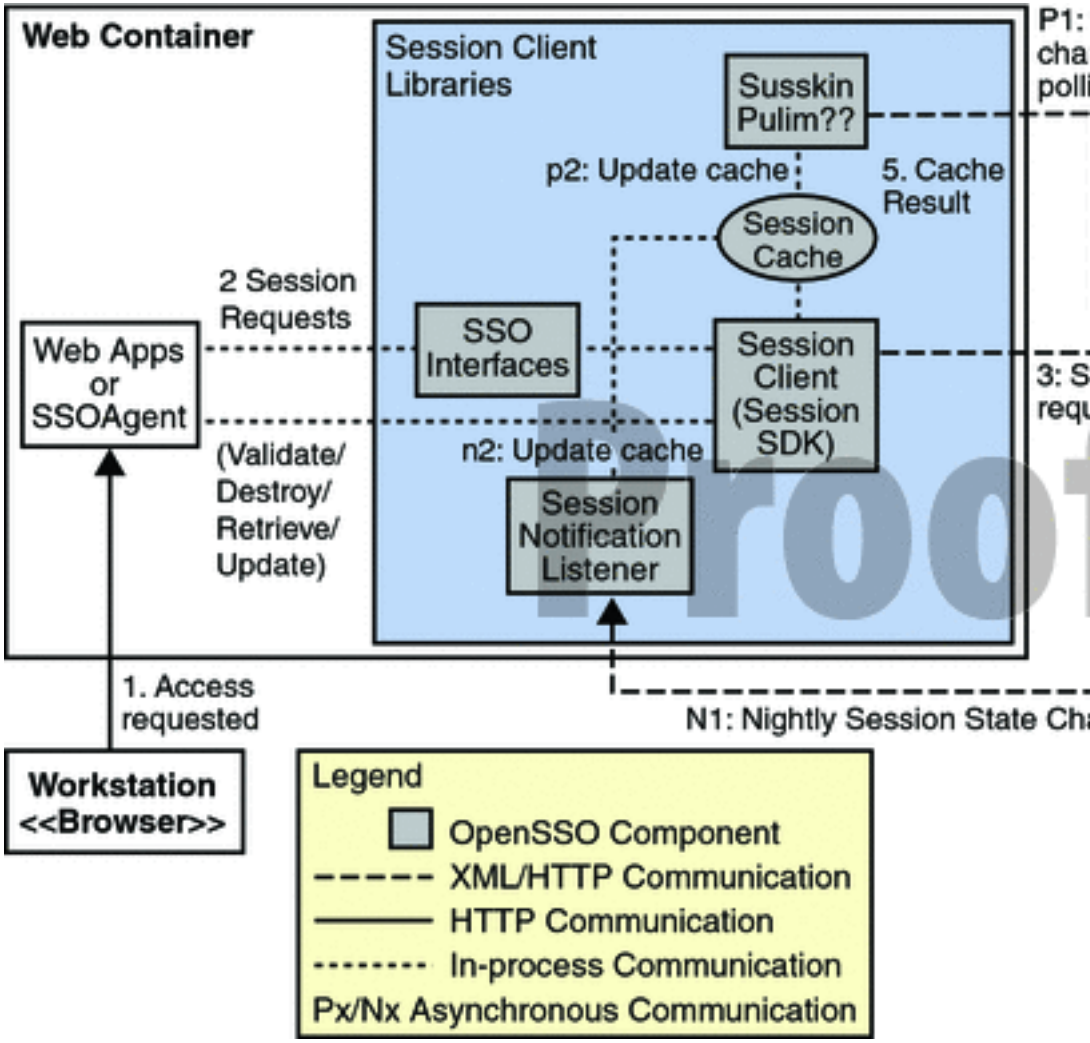


FIGURE 2–6 Components within the Session Service Framework

Figure 2–7 illustrates how the messaging capabilities of Message Queue are used to push session information to a persistent store based on the Berkeley DataBase (DB).

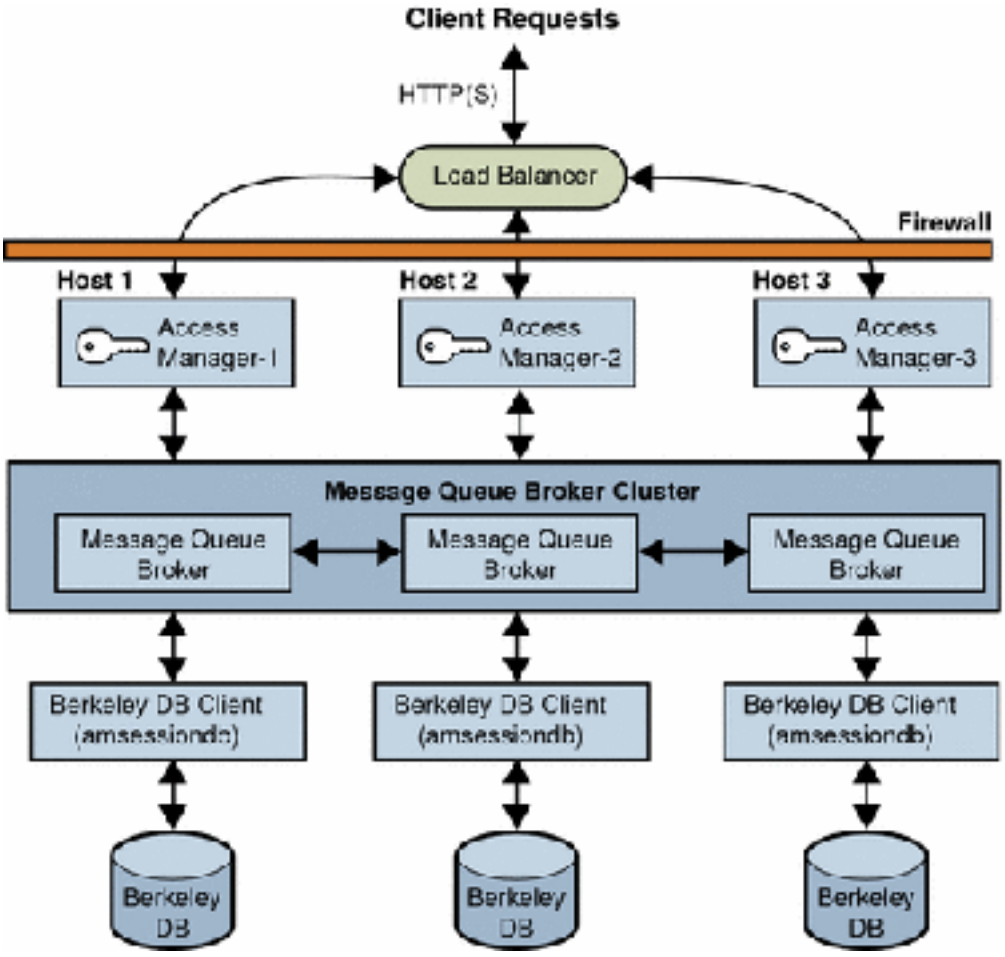


FIGURE 2–7 Session Persistence Deployment Architecture

Using Federated Access Manager in this manner enables the following key feature:

- Session Failover allows an alternative Federated Access Manager server to pick up a given user session when the server owning the original session fails.
- Session Constraints allow deployments to specify constraints on a sessions, such as one session per user.

Logging Service

When a user logs in to a resource protected by Federated Access Manager, the Logging Service records information about the user's activity. The common Logging Service can be invoked by

components residing on the same server as Federated Access Manager as well as those on the client machine, allowing the actual mechanism of logging (such as destination and formatting) to be separated from the contents which are specific to each component. You can write custom log operations and customize log plug-ins to generate log reports for specific auditing purposes.

Administrators can control log levels, authorize the entities that are allowed to create log entries and configure secure logging. Logged information includes the name of the host, an IP address, the identity of the creator of the log entry, the activity itself, and the like. Currently, the fields logged as a *log record* are controlled by the Configurable Log Fields selected in the Logging Configuration page located under the System tab of the Federated Access Manager console. The Logging Service is dependent on the client application (using the Logging APIs) creating a programmatic `LogRecord` to provide the values for the log record fields. The logging interface sends the logging record to the Logging Service which determines the location for the log record from the configuration. A list of active logs can also be retrieved using the Logging API. Figure 2–8 illustrates logging communications.

Caution – Generally speaking, writing log records can be done remotely, using the Client SDK, but anything involving the reading API can only be done on the machine on which Federated Access Manager is installed. Using the reading API is not recommended as it uses a lot of system resources, especially when database logging is involved (btw, DB logging is not mentioned here, and not shown in figure 2-8 on page 31). logging to a DB doesn't have a notion of "history" tables as flatfile logging does, so requesting all records from a table could be disasterous.

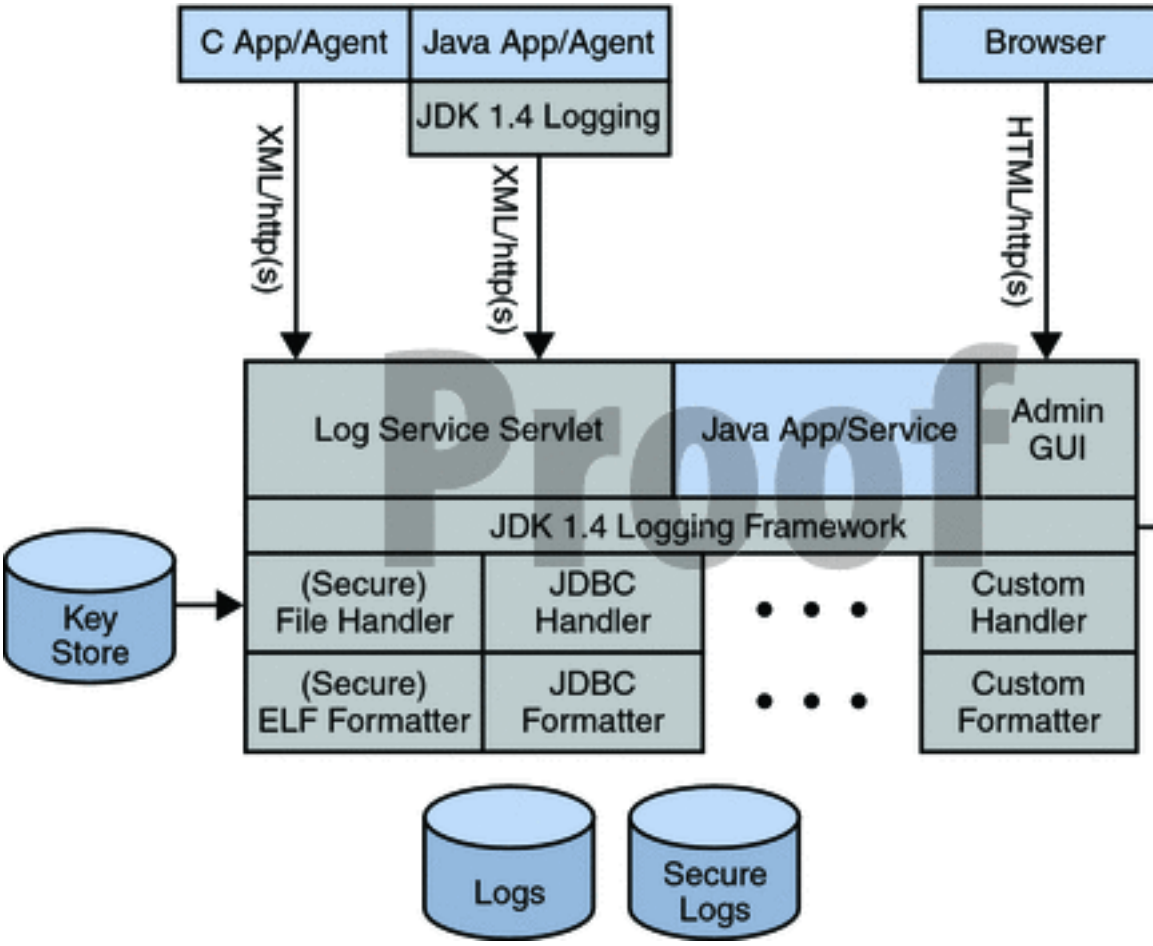


FIGURE 2–8 Components within the Logging Service Framework

See Chapter 11, "Logging and the Java Enterprise System Monitoring Framework," for more information.

Identity Repository Service

The Identity Repository Service allows Federated Access Manager to integrate an existing user data store (such as a corporate LDAP server) into the environment. The Identity Repository Service is able to access user profiles (as well as group and role assignments if supported) and is capable of spanning multiple repositories — even of different types. The current implementation supports Sun Java System Directory Server, IBM Tivoli Directory and Active Directory.

Access to the Identity Repository Service is provided by the `com.sun.identity.idm` Java package. The `AMIdentityRepository` class represents a realm that has one or more identity repositories configured and provides interfaces for searching, creating and deleting identities. The `AMIdentity` class represents an individual identity such as a user, group or role and provides interfaces to set, modify and delete identity attributes and assign and unassign services. `IdRepo` is an abstract class that contains the methods that need to be implemented by plug-ins when building new adapters for repositories not currently supported.

The Identity Repository Service is configured per realm under the Data Stores tab and it's main functions are:

- To specify an identity repository that will store service configurations and attributes for users, groups and roles.
- To provide a list of identity repositories that can provide user attributes to the Policy Service and Federation Services frameworks.
- To combine the attributes obtained from different repositories.
- To provide interfaces to create, read, edit, and delete identity objects such as a realm, role, group, user, and agent.
- To map identity attributes using the Principal Name from the `SSOToken` object.

Note – Access control for the Identity Repository Service is controlled by the Delegation Service using the Policy Service framework. The Delegation Service defines permissions that can be assigned to groups or roles. Those configured permissions become realm and policy privileges, and can be further assigned to administrator roles.

Federation Services

Federated Access Manager provides an open and extensible framework for identity federation and associated web services to resolve the problems of identity-enabling web services, web service discovery and invocation, security, and privacy. Federation Services are built on the following standards:

- Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) 1.1 and 1.2
- OASIS Security Assertion Markup Language (SAML) 1.0 and 1.1
- OASIS Security Assertion Markup Language (SAML) 2.0
- WS-Federation (Passive Requestor Profile)

Federation Services allows organizations to share a identity information (for example, which organizations and users are trusted, and what types of credentials are accepted) securely. Once this data can be exchanged securely, identity federation is possible, allowing a user to consolidate the many local identities configured among multiple service providers. With one federated identity, the user can log in at one service provider's site and move to an affiliated site

without having to re-establish identity. The following figure illustrates the actions and services common to federation and how they interact with other non-federation FAM components.

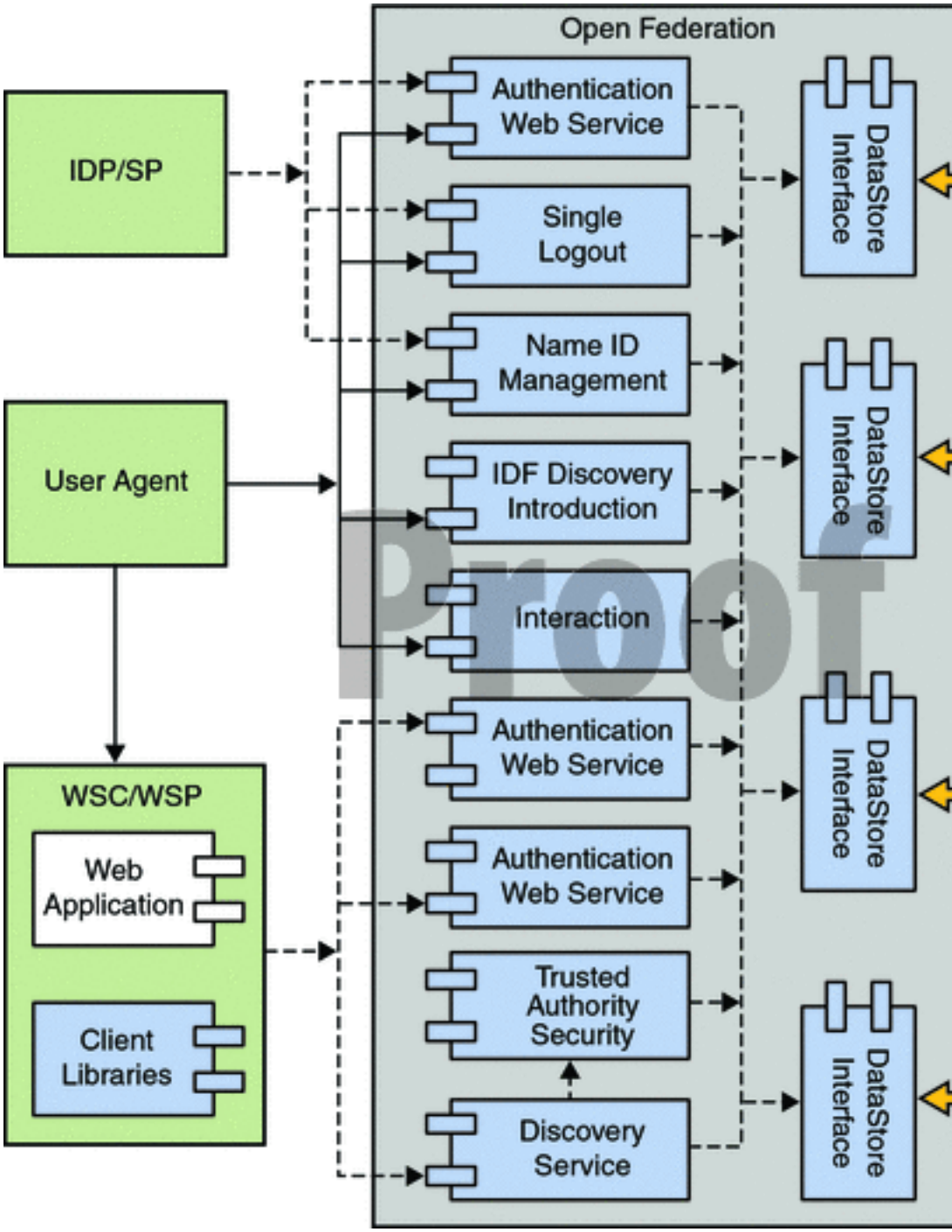


FIGURE 2–9 Federation Components Interaction within the Federated Access Manager Infrastructure

See the Federation Use Case documentation for more information.

Web Services

Web services follow a standardized way of integrating web-based applications using XML, SOAP, and other open standards over an Internet protocol backbone. They enable applications from various sources to communicate with each other because they are not tied to any one operating system or programming language. Businesses use web services to communicate with each other and their respective clients without having to know detailed aspects of each other's IT systems. Federated Access Manager provides web services that primarily use XML and SOAP over HTTP. These web services are designed to be centrally provided in an enterprise's network for convenient access by client applications. Federated Access Manager implements the follow web service specifications.

- Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF) 1.0, 1.1, and 2.0
- Web Services-Interoperability (WS-I) Basic Security Profile

The following table lists the Federated Access Manager web services.

TABLE 2–1 Federated Access Manager Web Services

Web Service Name	Description
Authentication Web Service	Provides authentication to a web service client (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.
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.
Security Token Service	The centralized Security Token Service that issues, renews, cancels, and validates security tokens.

TABLE 2–1 Federated Access Manager Web Services *(Continued)*

Web Service Name	Description
SOAP Binding Service	A set of Java APIs implemented 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.

Federated Access Manager uses both XML files and Java interfaces to manage web services and web services configuration data. A Federated Access Manager XML file is based on the structure defined in the Federated Access Manager Document Type Definition (DTD) files located in *path-to-context-root*`/fam/WEB-INF`. The main `sms.dtd` file defines the structure for all Federated Access Manager service files (located in *path-to-context-root*`/fam/WEB-INF/classes`).

Caution – Do not modify any of the DTD files. The Federated Access Manager API and their internal parsing functions are based on the default definitions and alterations to them may hinder the operation of the application.

Web Services Security

In message security, security information is applied at the message layer and travels along with the web services message. Message layer security differs from transport layer security in that it can be used to decouple message protection from message transport so that messages remain protected after transmission, regardless of how many hops they travel on. This message security is available as Web Services Security in Federated Access Manager and through the installation of an *authentication agent*. Web Services Security is the implementation of the WS-Security specifications and the Liberty Alliance Project Identity Web Services Framework (Liberty ID-WSF). Web Services Security allows communication with the Security Token Service to insert security tokens in outgoing messages and evaluate incoming messages for the same. Towards this end, authentication agents based on the Java Specification Request (JSR) 196 must be downloaded and installed on the web services client (WSC) machine and the web services provider (WSP) machine.

To secure web services communications, the requesting party must first be authenticated with a security token which is added to the SOAP header of the request. Additionally, the WSC needs to be configured to supply message level security in their SOAP requests and the WSP needs to be configured to enable message level security in their SOAP responses. Figure 2–10 illustrates the components used during a secure web services interaction.

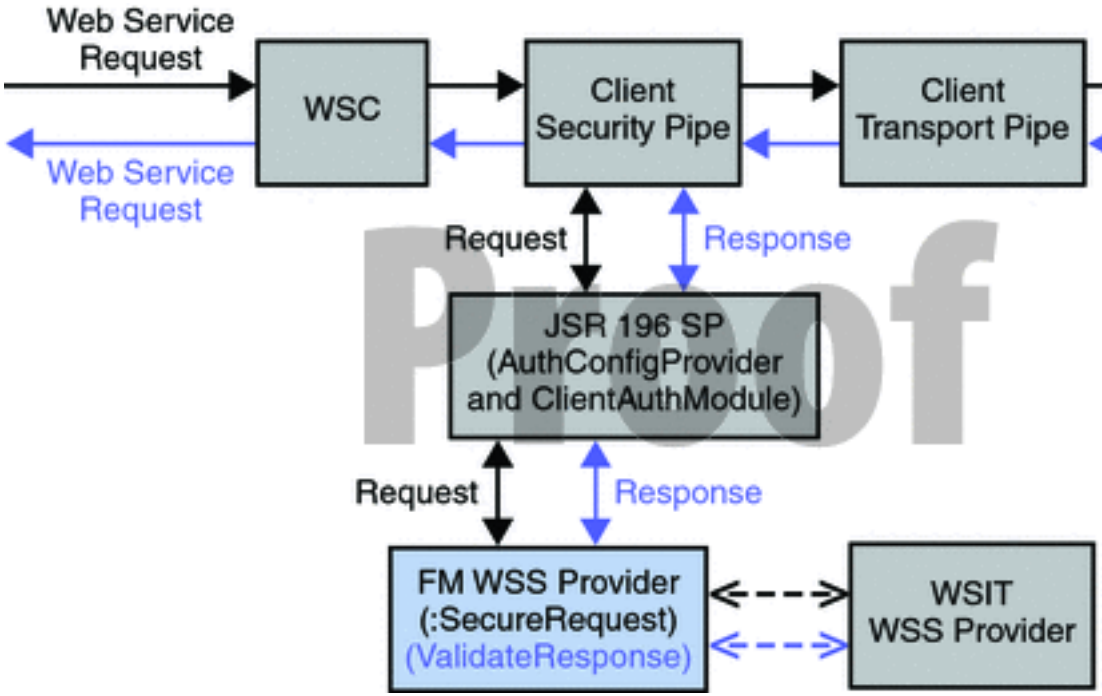


FIGURE 2–10 Components within the Web Services Security Interactions

Note – The stand alone applications can directly invoke the interfaces (secure request by WSC, and validate response by WSP) from the WS-Security Library and establish message-level end-to-end web service security. Standalone Java applications do not need the WS-Security Provider Plugin.

Identity Services

Federated Access Manager contains client interfaces for authentication, authorization, session management, and logging in Java, C, and C++ (using the proprietary XML and SOAP over HTTP or HTTPs). These interfaces are used by policy agents and custom applications. Development using these interfaces, though, is labor-intensive. Additionally, the interfaces cause dependencies on Federated Access Manager. Therefore, Federated Access Manager now has simple interfaces that can be used for efficient development of:

- Authentication (verification of user credentials, password management)
- Authorization (policy evaluation for access to secured resources)
- Provisioning (self-registration, creating or deleting identity profiles, retrieve or update identity profile attributes)

- Logging (auditing, recording operations)
- Token validation
- Search (return a list of identity profile attributes that match a search filter)

These Identity Services are offered using either SOAP and the Web Services Description Language (WSDL) or Representational State Transfer (REST). They are implemented by pointing an integrated development environment (IDE) application project to the appropriate URL and generating the stub code that wraps the function calls to the services.

Note – Federated Access Manager supports Eclipse, NetBeans, and Visual Studio.

The user interacts with the presentation logic of the application which could be, for example, a calendar, a human resources applications, or a banking account. The application calls either of the Identity Services to authenticate and authorize the identity, create personalized services, and log the actions. When contacted at the respective URL, Federated Access Manager obtains the user profile from the appropriate identity repository for authentication and the policy configuration from the appropriate configuration data store, and writes the actions to the configured log file. Figure 2–11 illustrates the components of the Identity Services.

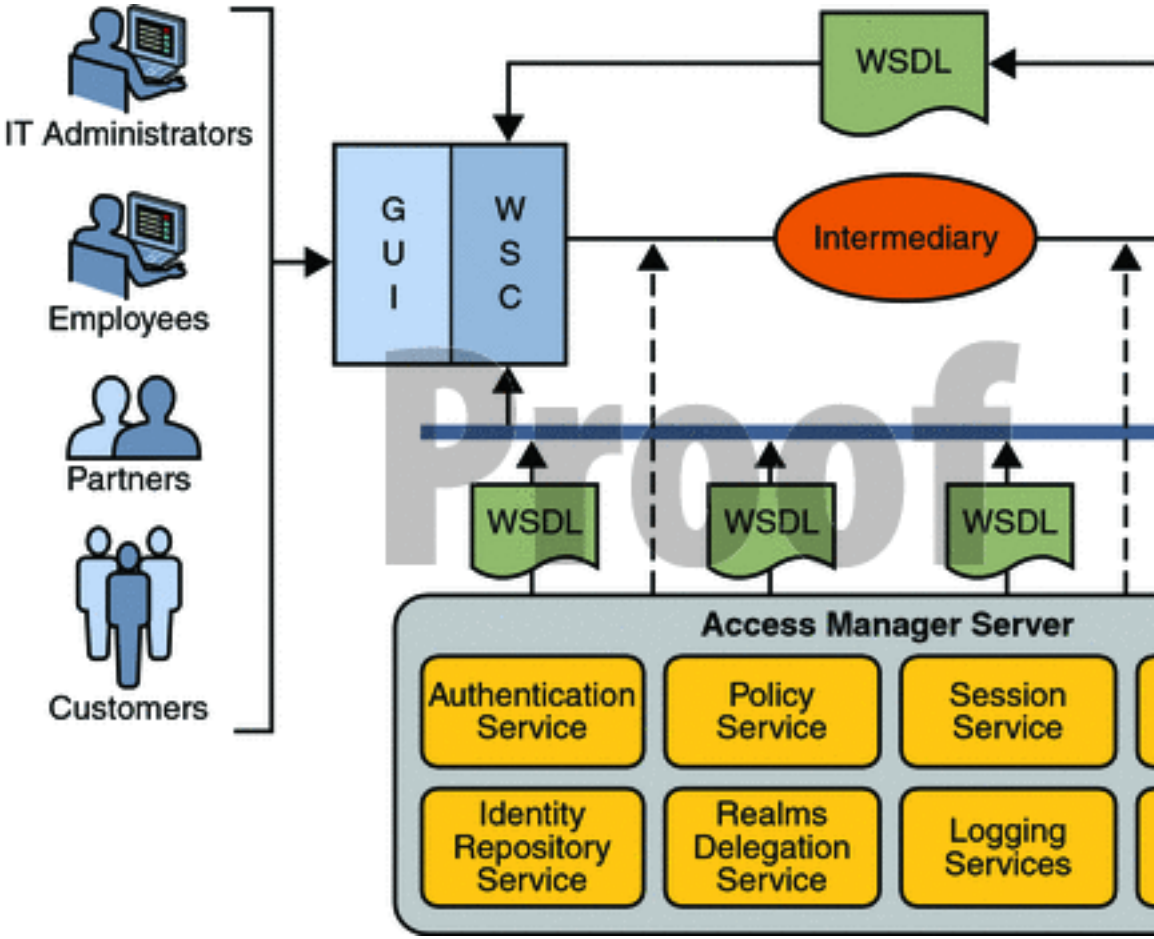


FIGURE 2–11 Components within the Identity Services Interactions

Note – Identity Services does not require the Client SDK or deployment of an agent or proxy to protect a resource.

Infrastructure

The following sections provide information on the components that define the infrastructure of a deployed Federated Access Manager server.

Data and Data Stores

Federated Access Manager services need to interact with a number of different data stores. The following distinct repositories can be configured.

- A configuration repository provides server and service specific data.
- One or more identity repositories provide user profile information.
- Authentication repositories provide authentication credentials to a particular module of the Authentication Service.

A common LDAP connection pooling facility allows efficient use of network resources. In the simplest *demonstration* environment, a single LDAP repository is sufficient for all data however, the typical *production* environment tends to separate configuration data from other data. The following sections contain more specific information.

Configuration Data

The default configuration of Federated Access Manager creates a branch in an embedded configuration data store for storing service configuration data and other information pertinent to the server's operation. Federated Access Manager components and plug-ins access the configuration data and use it for various purposes including:

- Accessing policy data for policy evaluation.
- Finding location information for identity data stores and Federated Access Manager services.
- Retrieving authentication configuration information that define how users and groups authenticate.
- Finding which partner servers can send trusted SAML assertions.

Federated Access Manager supports Microsoft Active Directory, Sun Java System Directory Server, and the open source OpenDS as configuration data stores. Flat files (supported in previous versions of the product) are no longer supported but configuration data store failover is by using `bootstrap`. (See "The `bootstrap` File" on page 62.) Figure 2–12 illustrates how configuration data in the embedded data store is accessed.

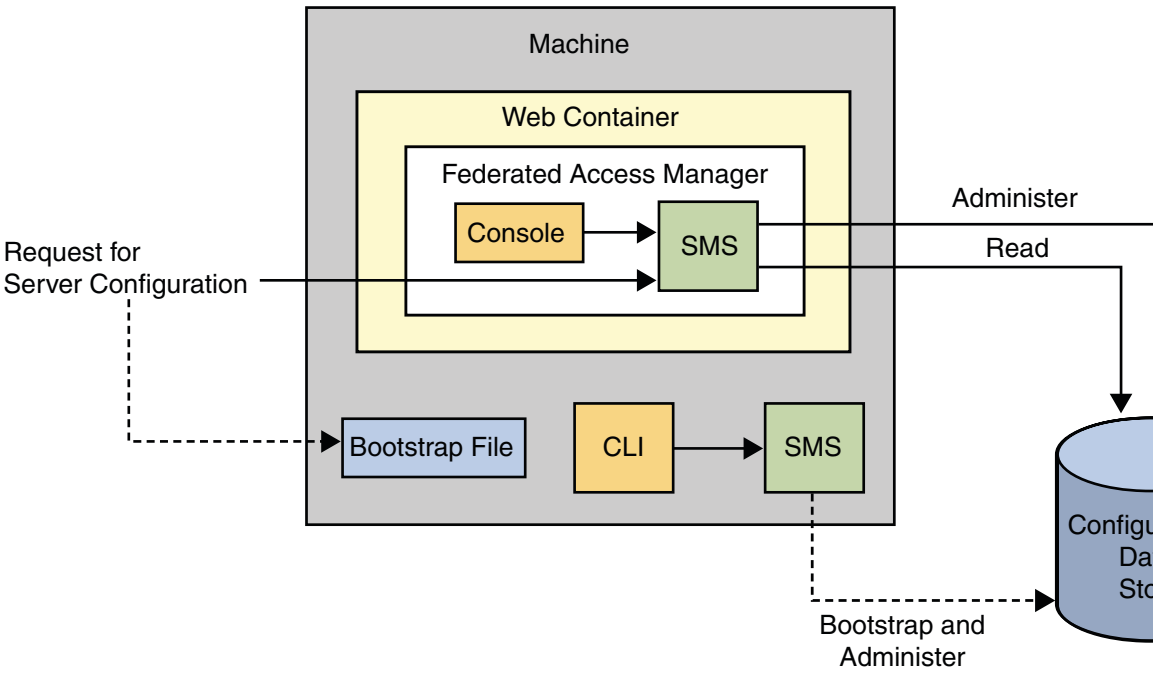


FIGURE 2–12 Accessing Configuration Data

Previous releases of Access Manager and Federation Manager stored product configuration data in a property file named `AMConfig.properties` that was installed local to the product instance directory. This file is deprecated for Federated Access Manager although supported for backward comparability. See the *Sun Federated Access Manager 8.0 Installation and Configuration Guide* for more information.

Configuration data comprises the attributes and values in the Federated Access Manager configuration services, as well as default Federated Access Manager users like `amadmin` and `anonymous`. It is stored under **`ou=services,`***ROOT SUFFIX* in the configuration data store. Following is a partial listing of the XML service files that contribute to the data. They can be found in the *path-to-context-root*`/fam/WEB-INF/classes` directory.

Note – The data in this node branch is private and is mentioned here for information purposes only.

- `AgentService.xml`
- `amAdminConsole.xml`
- `amAgent70.xml`
- `amAuth.xml`

- amAuth-NT.xml
- amAuthAD.xml
- amAuthAnonymous.xml
- amAuthCert.xml
- amAuthConfig.xml
- amAuthDataStore.xml
- amAuthHTTPBasic.xml
- amAuthJDBC.xml
- amAuthLDAP.xml
- amAuthMSISDN.xml
- amAuthMembership.xml
- amAuthNT.xml
- amAuthRADIUS.xml
- amAuthSafeWord-NT.xml
- amAuthSafeWord.xml
- amAuthSecurID.xml
- amAuthWindowsDesktopSSO.xml
- amClientData.xml
- amClientDetection.xml
- amConsoleConfig.xml
- amDelegation.xml
- amEntrySpecific.xml
- amFilteredRole.xml
- amG11NSettings.xml
- amLogging.xml
- amNaming.xml
- amPasswordReset.xml
- amPlatform.xml
- amPolicy.xml
- amPolicyConfig.xml
- amRealmService.xml
- amSession.xml
- amUser.xml
- amWebAgent.xml
- idRepoEmbeddedOpenDS.xml
- idRepoService.xml
- identityLocaleService.xml
- ums.xml

Caution – By default, the Federated Access Manager configuration data is created and maintained in the embedded configuration data store apart from any identity data. Although users can be created in the configuration data store this is only recommended for demonstrations and development environments.

For more information, see "Embedded Configuration Data" on page 69.

Identity Data

An *identity repository* is a data store where information about users, roles, and groups in an organization is stored. User profiles can contain data such as a first name, a last name, a phone number, or an e-mail address; an identity profile template is provided out-of-the-box but it can be modified to suit specific deployments.

Identity data stores are defined per realm. Because more than one identity data store can be configured per realm Federated Access Manager can access the many profiles of one identity across multiple data repositories. Sun Java System Directory Server, Microsoft Active Directory and IBM Tivoli Directory are the supported identity repositories. Plug-ins can be developed to integrate other types of repositories (for example, a relational database). Figure 2–13 illustrates a Federated Access Manager deployment where the identity data and the embedded configuration data are kept in separate data stores.

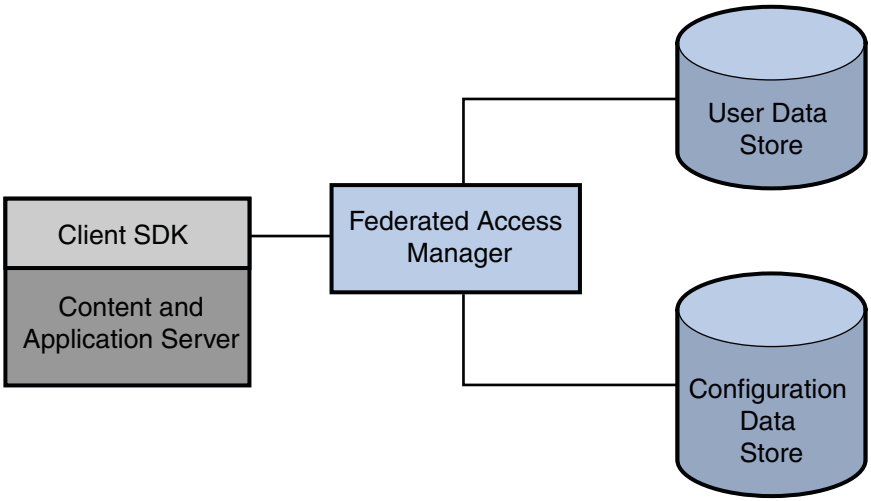


FIGURE 2–13 Federated Access Manager Deployment with Two Data Stores

Note – The information in an identity repository is maintained by provisioning products separate from Federated Access Manager. The supported provisioning product is Sun Java System Identity Manager.

Federated Access Manager provides out-of-the-box plug-in support for different types of identity repositories. Each default plug-in configuration includes details about what operations are supported on the underlying data store. Once a realm is configured to use a plug-in, the framework will instantiate it and execute the operations on the appropriate identity repository. Each new plug-in developed must have a corresponding service management schema defining its configuration attributes. This schema would be integrated into `idRepoService.xml` (the service management file for the Identity Repository Service which controls the identity data stores under a realm's Data Stores tab) as a sub schema. The following sections contain information on the out-of-the-box plug-ins.

- "Generic Lightweight Directory Access Protocol (LDAP) version 3" on page 57
- "Active Directory" on page 58
- "Sun Directory Server With Access Manager Schema" on page 58
- "Access Manager Repository Plug-in" on page 58

Note – For more information see the "Identity Repository Service" on page 44.

this one seems to be AM 7.1 information (it talks about JES installer). You can have the following for identity repositories: Generic LDAPv3 plugin, LDAPv3 plugin for AD, SUN DS with AM schema. AM repository plugin is used with SUN DS with AM schema. In addition to these, there is also a possibility of extending user repository(SUN DS with generic ldapv3 plugin or AD or IBM tivoli directory with additional schema to support things like session service, federation, etc). This information should also be covered may be elsewhere.

AMSDk plugin will not be an option in configurator. You can still do it later with manual configuration. So following are the user repository options. Generic LDAPv3 - AD (Console provides a way to configure it, schema needs to be loaded manually) Tivoli (Console provides a way to configure it, schema needs to be loaded manually) SUN DS with FAM core services schema (configurator option) SUN DS with full schema including legacy (manual configuration).

Generic Lightweight Directory Access Protocol (LDAP) version 3

The Generic LDAPv3 identity repository plug-in can be used with either Sun Java System Directory Server or Tivoli Directory Server. (It may work with other LDAPv3 repositories but they are not currently supported. This includes the open-source OpenDS directory server.) reside on an instance of any directory that complies with the LDAPv3 specifications. The directory can not make use of features that are not part of the LDAP version 3 specification, and

no specific DIT structure can be assumed as LDAPv3 identity repositories are simply DIT branches that contain user and group entries. The identity repositories might or might not reside in the same instance of Sun Java System Directory Server as the Access Manager information tree. Each data store has a name that is unique among a realm's data store names, but not necessarily unique across all realms in the Access Manager information tree. The `com.sun.identity.idm.plugins.ldapv3.LDAPv3Repo` class provides the default LDAPv3 identity repository implementation.

needs no schema from FAM, supports Sun DS and Tivoli Directory (configurator options)

Note – This configuration is not compatible with previous versions of Access Manager.

Active Directory

This data store type uses the LDAP version 3 specification to write identity data to an instance of Microsoft® Active Directory®.

Sun Directory Server With Access Manager Schema

This repository resides in an instance of Sun Java System Directory Server and holds the Access Manager information tree. It differs from the Access Manager Repository Plug-in in that more configuration attributes allow for better customization.

Access Manager Repository Plug-in

The Access Manager Repository can reside only in Sun Java System Directory Server. During installation, the repository itself is created in the same instance of Sun Java System Directory Server that holds the Access Manager information tree. (This is the default installation mode when using the Sun Java Enterprise System installer.) The two information trees are configured in separate nodes under a single directory suffix. The Access Manager Repository Plug-in is designed to work with Sun Java System Directory Server as it makes use of features specific to the server including *roles* and *class of service*. It uses a DIT structure similar to that of previous versions of Access Manager.

Note – Previously, the functionality of this plug-in was provided by the AMSDK component. In Access Manager 7.1, the AMSDK functionality still exists, but as a plug-in only. Thus, the Access Manager Repository is compatible with previous versions of Access Manager.

When you configure an instance of Access Manager in realm mode for the first time, the following occurs:

- An Access Manager Repository is created under the top-level realm.
- The Access Manager Repository is populated with internal Access Manager users.

Note – The Java Enterprise System installer does not set up an Access Manager Repository when you configure an Access Manager instance in legacy mode. Legacy mode requires an identity repository that is mixed with the Access Manager information tree under a single directory suffix.

Authentication Data

Authentication data contains authentication credentials for Federated Access Manager users. An authentication data store is aligned with a particular authentication module, and might include:

- RADIUS servers
- SafeWord authentication servers
- RSA ACE/Server systems (supports SecurID authentication)
- LDAP directory servers

Note – Identity data may include authentication credentials although authentication data is generally stored in a separate authentication repository.

Realms

A *realm* is the unit that Federated Access Manager uses to organize configuration information. Authentication properties, authorization policies, data stores, subjects (including a user, a group of users, or a collection of protected resources) and other data can be defined within the realm. The data stored in a realm can include, but is not limited to:

- One or more subjects (a user, a group of users, or a collection of protected resources)
- A definition of one or more data stores to store subject (user) data
- Authentication details identifying, for example, the location of the authentication repository, and the type of authentication required.
- Policy information that will be used to determine which resources protected by Federated Access Manager the subjects can access.
- Responder configurations that allows applications to personalize the user experience, once the user has successfully authenticated and been given access.
- Administration data for realm management

You create a top-level realm when you deploy Federated Access Manager. The top-level realm (by default `fam`) is the root of the Federated Access Manager instance and contains Federated Access Manager configuration data; it cannot be changed after it is created. All other realms are configured under the `fam` realm. These sub-realms may contain other sub-realms and so on. Sub-realms identify sets of users and groups that have different authentication or authorization requirements.

The Federated Access Manager framework aggregates realm properties as part of the configuration data. Figure 2–14 illustrates how configuration data can use a hierarchy of realms to distribute administration responsibilities. Region 1, Region 2, and Region 3 are realms; Development, Operations, and Sales are realms sub to Region 3.

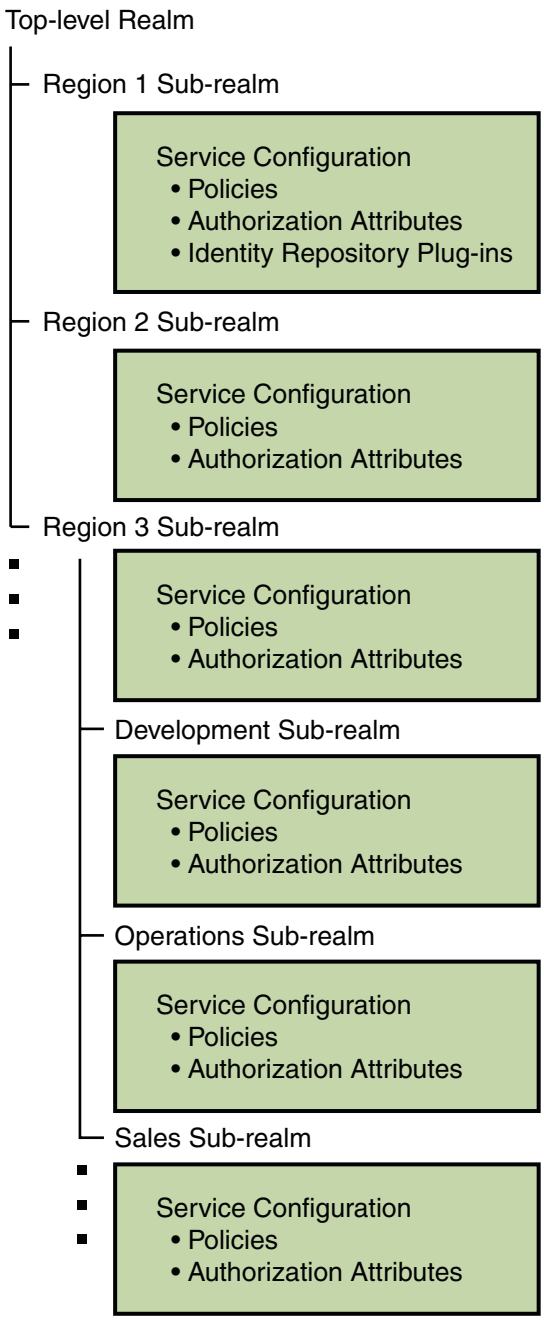


FIGURE 2–14 Realm Hierarchy for Configuration Data

Note – Federated Access Manager 8.0 supports the Sun Java System Access Manager Legacy mode (which contains no realms) with a provided interface.

The `bootstrap` File

Federated Access Manager uses a file to bootstrap itself. Previously, `AMConfig.properties` held configuration information to bootstrap the server but now a file named `bootstrap` points to the configuration data store allowing the setup servlet to retrieve the bootstrapping data. After deploying the Federated Access Manager WAR and running the configuration wizard, configuration data is written to the configuration data store by the service management API contained in the Java package, `com.sun.identity.sm`. The setup servlet creates `bootstrap` in the top-level `/fam` directory. The content in bootstrap can be either of the following:

- A directory local to Federated Access Manager (for example, `/export/SUNWam`) indicating the server was configured with a previous release. The directory is where `AMConfig.properties` resides.
- An encoded URL that points to a directory service using the following format:

  `ldap://`*ds-host*`:`*ds-port*`/`*server-instance-name*`?pwd=`*encrypted-amadmin-password*`&embeddedds=`*path-to-directory-service-installation*`&basedn=`*base-dn*`&dsmgr=`*directory-admin*`&dspwd=`*encrypted-directory-admin-password*

  For example:

  ```
  ldap://ds.samples.com:389/http://owen2.red.sun.com:8080/fam?
  pwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8&embeddedds=/fam/opends&
  basedn=dc=fam,dc=java,dc=net&dsmgr=cn=Directory+Manager
  &dspwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8
  ```

  where

  - `ds.samples.com:389` is the host name and port of the machine on which the directory is installed.
  - `http://owen2.red.sun.com:8080/fam` is the instance name.
  - `AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8` is the encrypted password of the OpenSSO administrator.
  - `/fam/opends` is the path to the directory installation.
  - `dc=fam,dc=java,dc=net` is the base DN.
  - `cn=Directory+Manager` is the directory administrator.
  - `BQIC5xM2LY4SfcximdVZEdtfwar4vhWNkmG7` is the encrypted password for the directory administrator.

If more than one URL is present in the file and Federated Access Manager is unable to connect or authenticate to the data store at the first URL, the bootstrapping servlet will try the second (and so on). Additionally, the number sign [#] can be used to exclude a URL as in:

```
# ldap://ds.samples.com:389/http://owen2.red.sun.com:8080/fam?
pwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8&embeddedds=/fam/opends&
basedn=dc=fam,dc=java,dc=net&dsmgr=cn=Directory+Manager
&dspwd=AQIC5wM2LY4Sfcxi1dVZEdtfwar2vhWNkmS8
```

Global Services

Global services take configuration values and perform functions for Federated Access Manager on a global basis. The following table lists the global services with brief descriptions.

TABLE 2–2 Global Federated Access Manager Services

Service	What it Does
Common Federation Configuration	XXXXX
Liberty ID-FF Service Configuration	XXXXX
Liberty ID-WSF Security Service	XXXXX
Liberty Interaction Service	XXXXX
Multi-Federation Protocol	XXXXX
Password Reset	XXXXX
Policy Configuration	XXXXX
SAML v2 Service Configuration	XXXXX
SAML v2 SOAP Binding	XXXXX
Security Token Service	XXXXX
Session	XXXXX
User	XXXXX

Policy Agents

Policy agents are an integral part of SSO and CDSSO sessions. They are programs that police the web container on which protected resources are hosted. When a user requests access to a protected resource such as a server or an application, the policy agent intercepts the request and redirects it to the Federated Access Manager Authentication Service for authentication.

Following this, the policy agent requests the authenticated user's assigned policy and evaluates it to allow or deny access. (A *policy* defines the rules that specify a user's access privileges to a protected resource.) Federated Access Manager supports two types of policy agents:

- The *web agent* enforces URL-based policy for C applications.
- The *Java EE agent* enforces URL-based policy and Java-based policy for Java applications on Java EE containers.

Both types of agents are available for you to install as programs separate from Federated Access Manager. Policy agents are basically clients written using the Client SDK and Identity Services.

Note – All HTTP requests are implicitly denied unless explicitly allowed by the presence of a valid session and a policy allowing access. If the resource is defined in the Not Enforced list for the policy agent, access is allowed even if there is no valid session.

For an overview of the available policy agents and links to specific information on installation, see the *Sun Java System Federated Access Manager Policy Agent 3.0 User's Guide*.

Authentication Agents

Authentication agents plug into web containers to provide message level security for web services, and supports both Liberty Alliance Project token profiles as well as Web Services-Interoperability Basic Security Profiles (WS-I BSP). (A *profile* defines the HTTP exchanges required to transfer XML requests and responses between web service clients and providers.) Authentication agents use an instance of Federated Access Manager for all authentication decisions. Web services requests and responses are passed to the appropriate authentication modules using standard Java representations based on the transmission protocol. An HTTP Authentication Agent or a SOAP Authentication Agent can be used. For more information, see "Web Services Security" on page 49.

Client SDK

Enterprise resources cannot be protected by Federated Access Manager until the Federated Access Manager Client SDK is installed on the machine that contains the resource that you want to protect. (The Client SDK is automatically installed with a policy agent.) The Client SDK allows you to customize an application by enabling communication with Federated Access Manager for retrieving user, session, and policy data.

Service Provider Interfaces for Plug-ins

The Federated Access Manager service provider interfaces (SPI) work as plug-ins to provide customer data to the Federated Access Manager framework for back-end processing. Some customer data comes from external data base applications such as identity repositories while other customer data comes from the Federated Access Manager plug-ins themselves. You can develop additional custom plug-ins to work with the SPI. For a complete list of the SPI, see the *Federated Access Manager 8.0 Java API Reference*. The following sections contain brief descriptions.

Authentication Service SPI

The `com.sun.identity.authentication.spi` package provides interfaces and classes for writing a supplemental authentication module to plug into Federated Access Manager. The `com.sun.identity.authentication` package provides interfaces and classes for writing a remote client application that can access user data in a specified identity repository to determine if a user's credentials are valid.

Delegation Service Plug-in

The Delegation Service plug-in aggregates policies and roles to determine the scope of a configured Federated Access Manager realm (including top-level) administrator's authority. The Authentication Service and the Policy Service use the aggregated data to perform authentication and authorization processes. The Delegation Service plug-in works together with the Identity Repository Service plug-in (where default administrator roles are defined) to form rules that describe the scope of privileges for each administrator, and specifies the roles to which these rules apply. The following is a list of roles defined by the Identity Repository Service plug-in, and the default rule the Delegation Service plug-in applies to each.

TABLE 2–3 Administrator Roles and Scope of Privileges

Administrator Role	Delegation Rule
Realm Administrator	Can access all data in all configured realms.
Subrealm Administrator	Can access all data within the specified realm.
Policy Administrator	Can access all policies in all configured realms.

TABLE 2–3 Administrator Roles and Scope of Privileges *(Continued)*

Administrator Role	Delegation Rule
Policy Realm Administrator	Can access policies only within the specified realm.

Federation Service SPI

The `com.sun.identity.federation.services` package provides plug—ins for customizing the Liberty ID-FF profiles implemented by Federated Access Manager. The `com.sun.identity.federation.plugins` package provides an interface that can be implemented to perform user specific processing on the service provider side during the federation process. The `com.sun.identity.saml2.plugins` package provides the SAML v2 service provider interfaces (SPI). The `com.sun.identity.wsfederation.plugins` package provides the WS-Federation based SPI.

Identity Repository Service SPI

The `com.sun.identity.idm` package contains the `IdRepo` interface that defines the abstract methods which need to be implemented or modified by Identity Repository Service plug-ins. `com.sun.identity.plugin.datastore` package contains interfaces that search for and return identity information such as user attributes and membership status for purposes of authentication.

Policy Service SPI

The `com.sun.identity.policy.interfaces` package provides interfaces for writing custom policy plug-ins for Conditions, Subjects, Referrals, Response Providers and Resources.

Service Configuration Plug-in

The `com.sun.identity.plugin.configuration` package provides interfaces to store and manage configuration data required by the core Federated Access Manager components and other plug-ins.

Note – In previous releases, the functionality provided by the Service Configuration plug-in was known as the Service Management Service (SMS).

◆ ◆ ◆ CHAPTER 3

Simplifying Federated Access Manager

This chapter contains information on the usability and manageability features of Federated Access Manager. It includes the following sections:

Installation and Configuration

Previous versions of Sun Microsystems' access management server were built for multiple hardware platforms, and different web containers. The complexity of this development process led to the release of separate platform and container patches. To alleviate this extraneous development, Federated Access Manager is now available as a single ZIP file which can be downloaded, unzipped, and quickly deployed; there will be no separate installations for each hardware platform. The ZIP file will contain the full Federated Access Manager web archive (WAR), layouts for the generation of other specific WARs, libraries, the Java API reference documentation, and samples. (Tools for use with Federated Access Manager, including the command line interfaces and policy and authentication agents, can be downloaded separately.) Figure 3–1 illustrates the process for deployment, installation and configuration of a new Federated Access Manager WAR and a patched WAR.

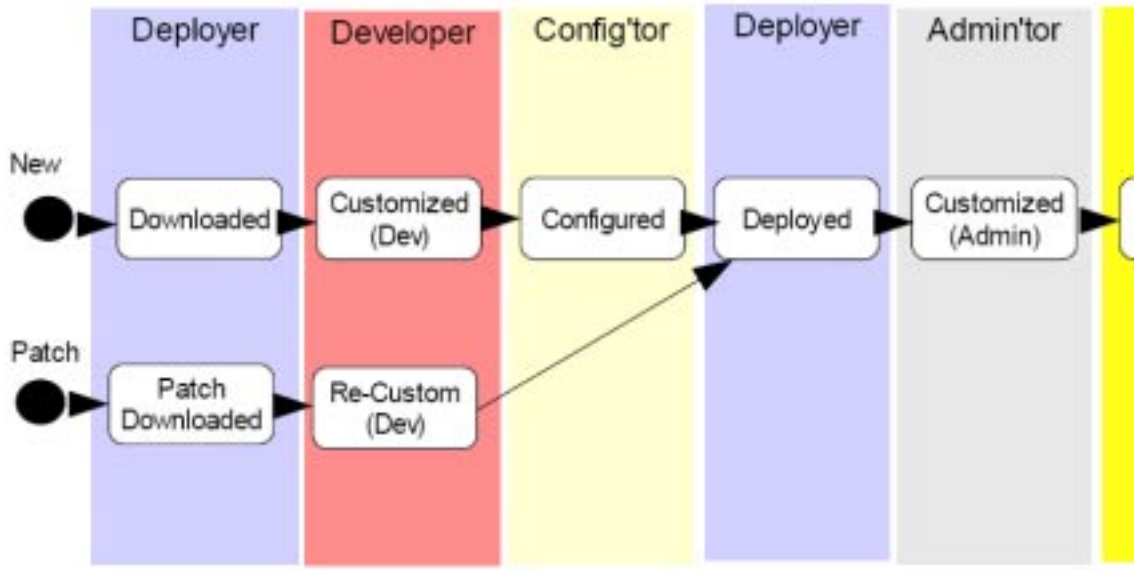


FIGURE 3–1 Installation and Configuration Process

Note – When patched, a full patched version of the Federated Access Manager WAR will be included in the download, assuring that there is always a single download to get the latest bits.

The intent of this new process is to allow the administrator to download Federated Access Manager and deploy it on the container or platform of choice, using the web container's administration console or command line interfaces. After the initial launch of the deployed WAR, the user is directed to a JavaServer Page (JSP) called the `Configurator` that prompts for configuration parameters (including, but not limited to, the host name, port number, URI, and repositories), provides data validation for the parameter values to prevent errors, and eliminates post-installation configuration tasks. Once successfully configured, any further changes to the configuration data store must be made using the Federated Access Manager console or command line interfaces.

Note – When deploying Federated Access Manager against an existing legacy installation, the Directory Management tab will be enabled in the new console.

For more information including a directory layout of the ZIP, see the *Sun Federated Access Manager 8.0 Installation and Configuration Guide*.

Embedded Configuration Data

Federated Access Manager has implemented an embedded configuration data store to replace the `AMConfig.properties` and `serverconfig.xml` files which had been the storage files for server configuration data. Previously, each instance of the server installed had separate configuration files but now when deploying more than one instance of Federated Access Manager, all server configuration data is stored centrally, in one embedded configuration data store per instance. After the Federated Access Manager WAR is configured, a sub configuration is added under the Platform Service to store the data and a bootstrap file that contains the location of the configuration data store is created in the installation directory. Figure 3–2 illustrates how Federated Access Manager is bootstrapped.

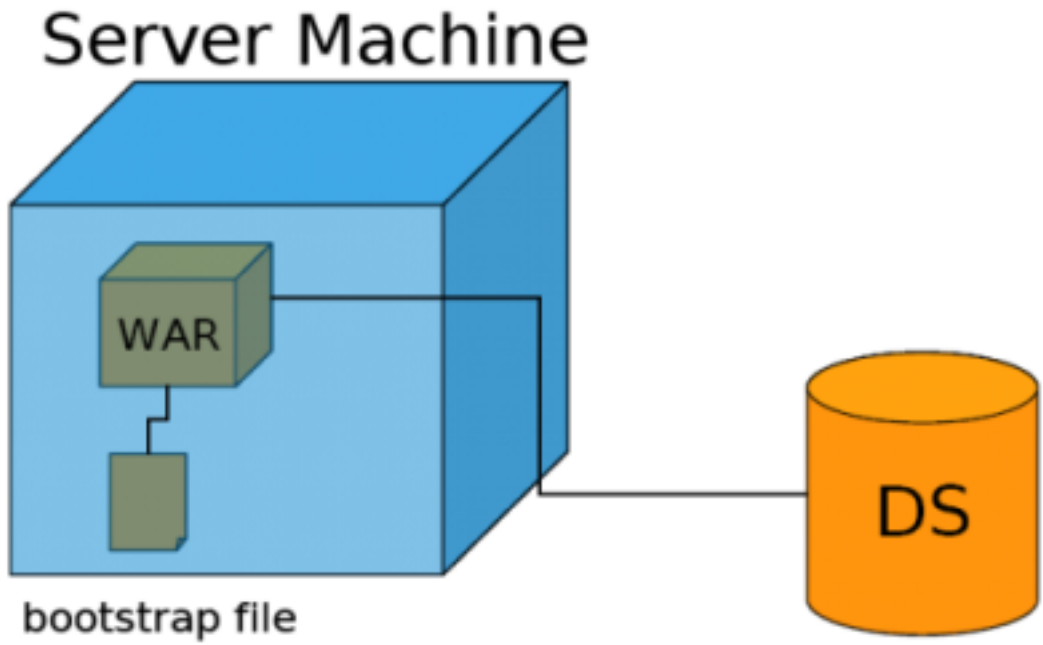


FIGURE 3–2 Bootstrapping Federated Access Manager

Post-installation, the configuration data can be reviewed and edited using the Federated Access Manager console or the command line interface. For more information see the XXXXXX Admin Guide.

Note – Federated Access Manager also supports an LDAPv3–based solution that uses an existing directory server for configuration data storage. This is configured during installation. Supported directories include Sun Java System Directory Server, Microsoft Active Directory, and IBM Tivoli Directory.

Centralized Agent Configuration

Policy agents function based on a set of configuration properties. Previously, these properties were stored in the `AMAgent.properties` file, residing on the same machine as the agent. With Centralized Agent Configuration, Federated Access Manager moves most of the agent configuration properties to the embedded configuration data store. Now agent profiles can be configured to store properties locally (on the machine to which the agent was deployed) or centrally (in the embedded configuration data store), making this new function compatible with both older 2.x agents and newer 3.0 agents. Following is an explanation of the local and central agent configuration repositories.

- Local agent configuration is supported for backward compatibility. Agent configuration data is stored in a property file named `FAMAgentConfiguration.properties` that is stored on the agent machine. It is only used by agent profiles configured locally.
- Centralized Agent Configuration stores agent configuration data in a centralized data store managed by Federated Access Manager. When an agent starts up, it reads its bootstrapping file to initialize itself. `FAMAgentBootstrap.properties` is stored on the agent machine and indicates the location from where the configuration properties need to be retrieved. It is used by agent profiles configured locally or centrally. Based on the repository setting in `FAMAgentBootstrap.properties`, it retrieves the rest of its configuration properties. If the repository is local, it reads the agent configuration from a local file; if the repository is remote, it fetches its configuration from Federated Access Manager.

Thus, Centralized Agent Configuration separates the agent configuration properties into two places: a bootstrapping file stored local to the agent and either a local (to the agent) or central (local to Federated Access Manager) agent configuration data store. `FAMAgentBootstrap.properties` is the bootstrapping file used by agent profiles configured locally or centrally. It is stored on the agent machine and indicates the local or central location from where the agent's configuration properties are retrieved. If the repository is local to the agent, it reads the configuration data from a local file; if the repository is remote, it fetches its configuration from Federated Access Manager. Choosing Centralized Agent Configuration provides an agent administrator with the means to manage multiple agent configurations from a central place using either the Federated Access Manager console or command line interface. Figure 3–3 illustrates how an agent retrieves bootstrapping and local configuration data, and configuration data from the configuration data store.

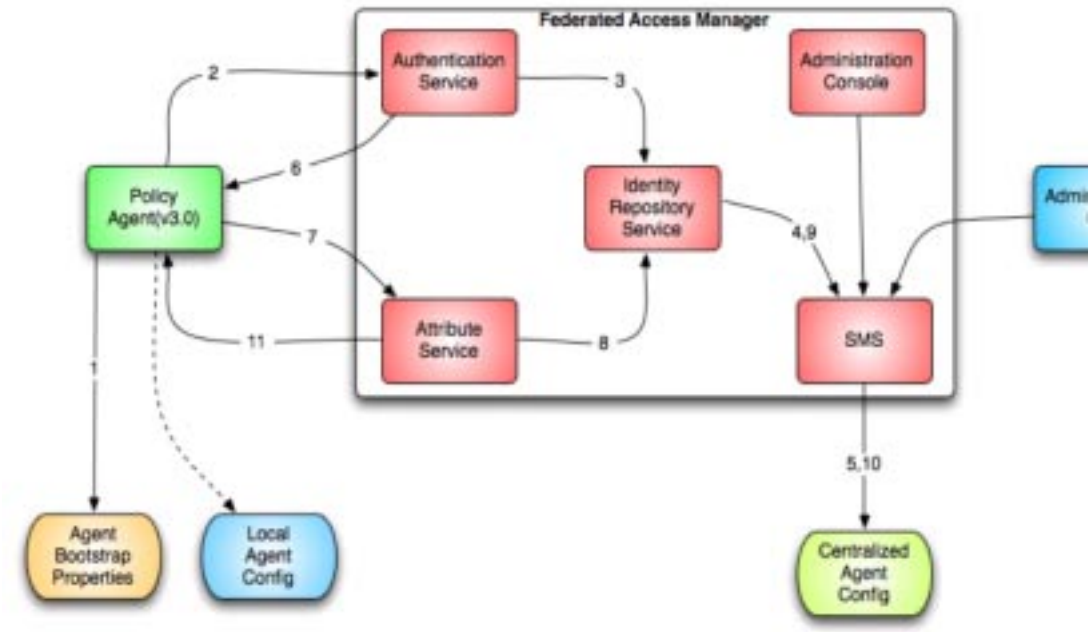


FIGURE 3–3 Retrieving Agent Configuration Data

An agent fetches its configuration properties periodically to determine if there have been any configuration changes. Any agent configuration changes made centrally are conveyed to the affected agents which will react accordingly based on the nature of the updated properties. If the properties affected are *hot swappable*, the agent can start using the new values without a restart of the underlying agent web container. Notification of the agent when configuration data changes and polling by the agent for configuration changes can be enabled. Agents can also receive notifications of session and policy changes.

Note – A agent configuration data change notification does not contain the actual data; it is just a ping that, when received, tells the agent to make a call to Federated Access Manager and reload the latest. Session and policy notifications, on the other hand, contain the actual data changes. Also, when using a load balancer, the notification is sent directly to the agent whose configuration has been changed. It does not go through the load balancer.

For more information see the XXXXXX.

Common Tasks

Federated Access Manager has implemented a Common Tasks tab that allows an administrators to create federation-based objects using console wizards. The wizards offer simplified entity provider configuration with metadata input using the URL `http://fam.sun.com:8080/fam/saml2/jsp/exportmetadata.jsp` or a metadata file. The following things can be done using a Common Task wizard:

- **Create SAML v2 Providers** They can be hosted or remote provider; and identity or service provider. To create them, you just need to provide some basic information about the providers.
- **Create Fedlet** A *Fedlet* is a small ZIP file that can be given to a service provider to allow for immediate federation with an identity provider configured with Federated Access Manager. It is ideal for an identity provider that needs to enable a service provider with no federation solution in place. The service provider simply adds the Fedlet to their application, deploys their application, and they are federation enabled.
- **Test Federation Connectivity** This test will show if federation connections are being made successfully by identifying where the troubles, if any, are located.
- **Access Documentation** This link opens the OpenSSO documentation page. View frequently asked questions, tips, product documentation, engineering documentation as well as links to the community blogs.
- **Register Your Product** This link allows you to register your product with Sun Connection. You must have a Sun Online Account in order to complete the registration. If you do not already have one, you may request one as part of this process.

Multi-Federation Protocol Hub

Federated Access Manager allows a configured circle of trust to contain entities speaking different federation protocols thus supporting cross protocol single sign-on and logout among hosted identity providers in the same circle-of-trust. For example, you can create a circle of trust containing one identity provider instance that communicates with multiple federation protocol and three service provider instances that speak, respectively, Liberty ID-FF, SAML v2 and WS-Federation. Figure 3–4 illustrates the process of multi-federation single sign-on and single logout.

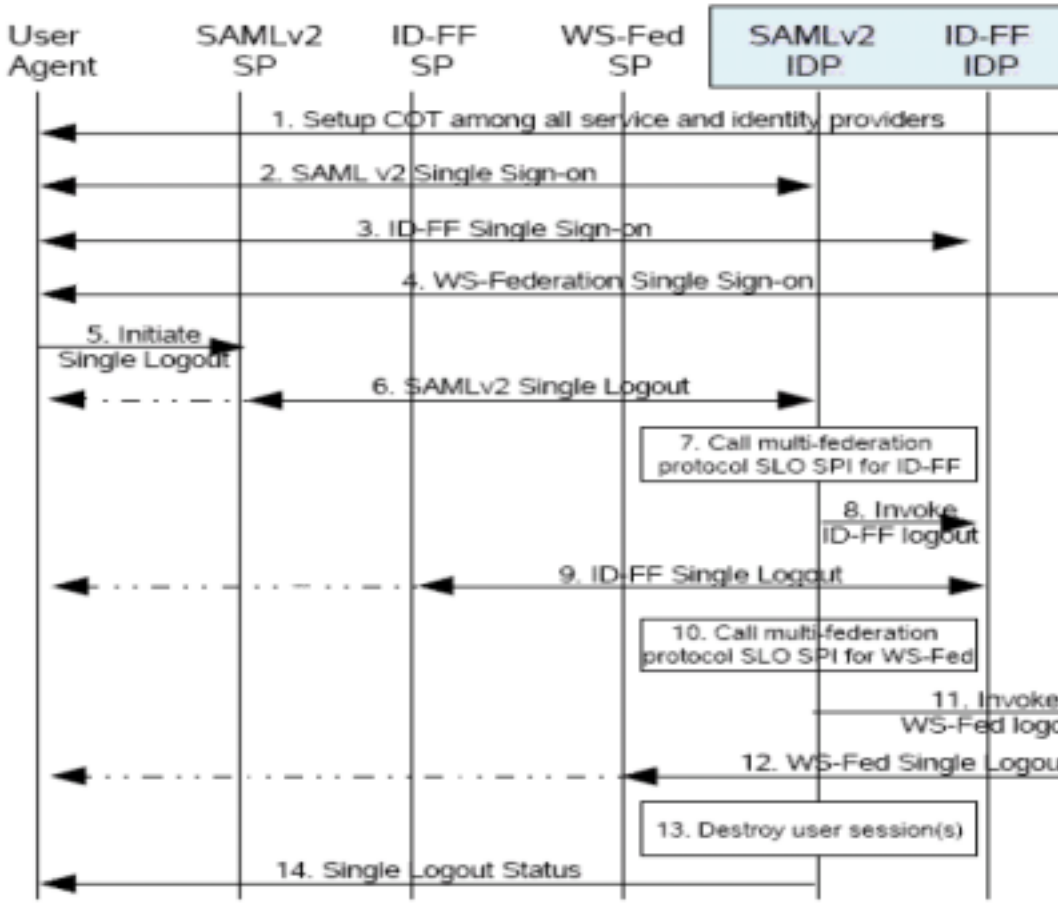


FIGURE 3–4 Multi-federation Single Sign-on and Single Logout

Federation and Federation Configuration Validation

XXXXXX

Third Party Integration

XXXXXX

PART II

Access Management Using Federated Access Manager

Access management is one of the core functions of Sun Federated Access Manager. User authentication, authorization for access to protected resources, and programmatically defining user sessions are all aspects of access management. Federated Access Manager offers access management features programmatically using the Client SDK, over the wire using HTTP and the Federated Access Manager console, and incorporating Representational State Transfer (REST) calls and Web Services Definition Language (WSDL) files using an integrated development environment (IDE) application. The chapters in this part contain information on these aspects of access management.

◆ ◆ ◆ CHAPTER 4

Understanding Sessions and the Session Service

The Session Service in Sun Federated Access Manager tracks a user's interaction with web applications through the use of session data structures, session tokens, cookies, and other objects. This chapter explains these concepts and other components of a user's session and contains the following sections:

About the Session Service

The Session Service in Sun Federated Access Manager tracks a user's interaction with web applications. For example, the Session Service maintains information about how long a user has been logged in to a protected application, and enforces timeout limits when necessary. Additionally, the Session Service:

- Generates session identifiers.
- Maintains a master copy of session state information.
- Implements time-dependent behavior of sessions.
- Implements session life cycle events such as logout and session destruction.
- Generates session life cycle event notifications.
- Generates session property change notifications.
- Implements session quota constraints.
- Implements session failover.
- Enables single sign-on and cross-domain single sign-on among applications external to Federated Access Manager.

The state of a particular session can be changed by user action or timeout. Figure 4–1 illustrates how a particular session is created as invalid prior to authentication, how it is activated following a successful authentication, and how it can be invalidated (and destroyed) based on different timeout values.

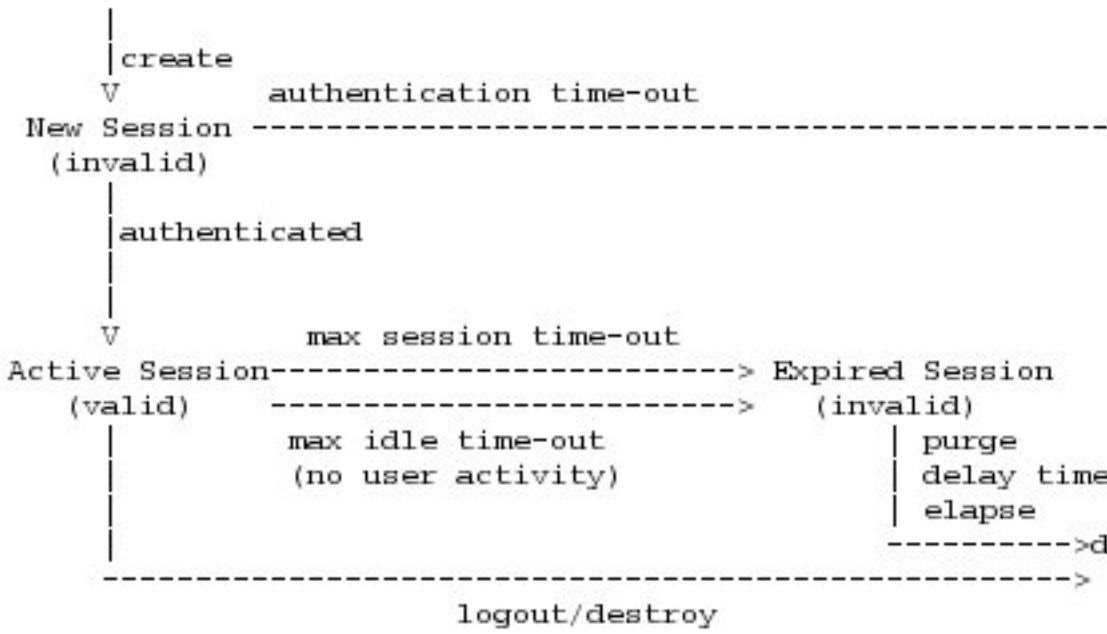


FIGURE 4–1 Session Timeouts

Remote access to the Session Service is offered by the session libraries in the Client SDK by which the user sessions can be validated, updated, and destroyed remotely.

User Sessions and Single Sign-on

A *user session* is the interval between the moment a user attempts to log in to a resource protected by Federated Access Manager, and the moment the user logs out of Federated Access Manager. As an example of a user session, an employee attempts to access the corporate benefits administration application protected by Federated Access Manager. A new invalid session is created, and the Authentication Service prompts the user for a username and password to verify the user's identity. Following a successful authentication, the Policy Service and policy agent work together to check that the user has the appropriate permissions to access the protected application and allows or denies access based on the outcome.

Oftentimes, in the same user session (without logging out of the corporate benefits application), the same employee might attempt to access a corporate expense reporting application. Because the expense reporting application is also protected by Federated Access Manager, the Session Service provides proof of the user's authentication, and the employee is allowed to access the expense reporting application (based on the outcome of another authorization check). If access is granted, the employee has accessed more than one application in a single user session without having to reauthenticate. This is called *single sign-on* (SSO). When SSO occurs among applications in more than one DNS domain, it is called *cross-domain single sign-on* (CDSSO). For more detailed explanations, see any of the following sections:

In the Single Sign-On environment,

- "Basic User Session" on page 81
- "Single Sign-On Session" on page 91
- "Cross-Domain Single Sign-On Session" on page 93

All clients participating in SSO must know the identifier of the appropriate session data structure to be able to perform actions and interact with other components within the system. See "Session Data Structures and Data Structure Identifiers" on page 79 for information.

Session Data Structures and Data Structure Identifiers

The result of a successful authentication results in the validation of a *session data structure* for the user or entity and a *session token*. The session token, also referred to as a *sessionID* and programmatically as an `SSOToken`, is an encrypted, unique string that identifies the session data structure. If the session token is known to a protected resource such as an application, the application can access all user and session information identified by it. With Federated Access Manager, a session token is carried in a *cookie*, an information packet generated by a web server and passed to a web browser.

Note – The generation of a cookie for a user by a web server does not guarantee that the user is allowed access to protected resources. The cookie simply points to user information in a data store from which an access decision can be made.

The Session Service programmatically creates a session data structure to store information about a user session. The session data structure minimally stores the following information.

Maximum Idle Time	Maximum number of minutes without activity before the session will expire and the user must reauthenticate.
Maximum Session Time	Maximum number of minutes (activity or no activity) before the session expires and the user must reauthenticate.
Maximum Caching Time	Maximum number of minutes before the client contacts Access Manager to refresh cached session information.

Internally, these session attributes are used to enforce timeout limits. But a session can also contain additional attributes and properties which can be used by other applications. For example, a session data structure can store information about a user's identity, or about a user's browser preferences. You can configure Federated Access Manager to include the following types of data in a session:

- Fixed session attributes
- Protected properties
- Custom properties

For a detailed summary of information that can be included in a session, see Chapter 10, "Configuring Federated Access Manager Sessions," in *Sun Federated Access Manager 8.0 Installation and Configuration Guide*.

As previously noted, the Session Service also generates a session data structure identifier to point to the new session data structure. The *session token*, also known as a *sessionID* and programmatically as an `SSOToken`, is an encrypted string that is carried in a cookie and uniquely identifies a specific session data structure. As the user visits different protected resources using the browser, the `SSOToken` is propagated to these resources. The resource uses the `SSOToken` to retrieve the user's credentials and validate them by sending a back-end request (via an embedded client API or agent) to Federated Access Manager. The server then returns an error or the session's prior authentication information (including the authentication level, authentication module, time of authentication, and idle time). Access to some Federated Access Manager services, such as the Policy Service and the Logging Service, require presentation of both the `SSOToken` of the application as well as the `SSOToken` of the user, allowing only designated applications to access these services. Sessions (and hence the `SSOToken`) are invalidated when a user logs out, the session expires, or a user in an administrative role invalidates it.

◆ ◆ ◆ CHAPTER 5

User Session and Single Sign-On Processes

This chapter traces events in a basic user session, a single sign-on session (SSO), a cross-domain single sign-on session (CDSSO), and session termination to give you an overview of the features and processes being invoked. It contains the following sections:

Basic User Session

The following sections describe the process behind a basic user session by tracing what happens when a user logs in to a resource protected by Federated Access Manager. In these examples, the server which hosts an application is protected by a policy agent. The Basic User Session includes the following phases:

Initial HTTP Request

When a user initiates a user session by using a browser to log in to a protected web-based application, the events illustrated in Figure 5–1 occur. The accompanying text describes the process.

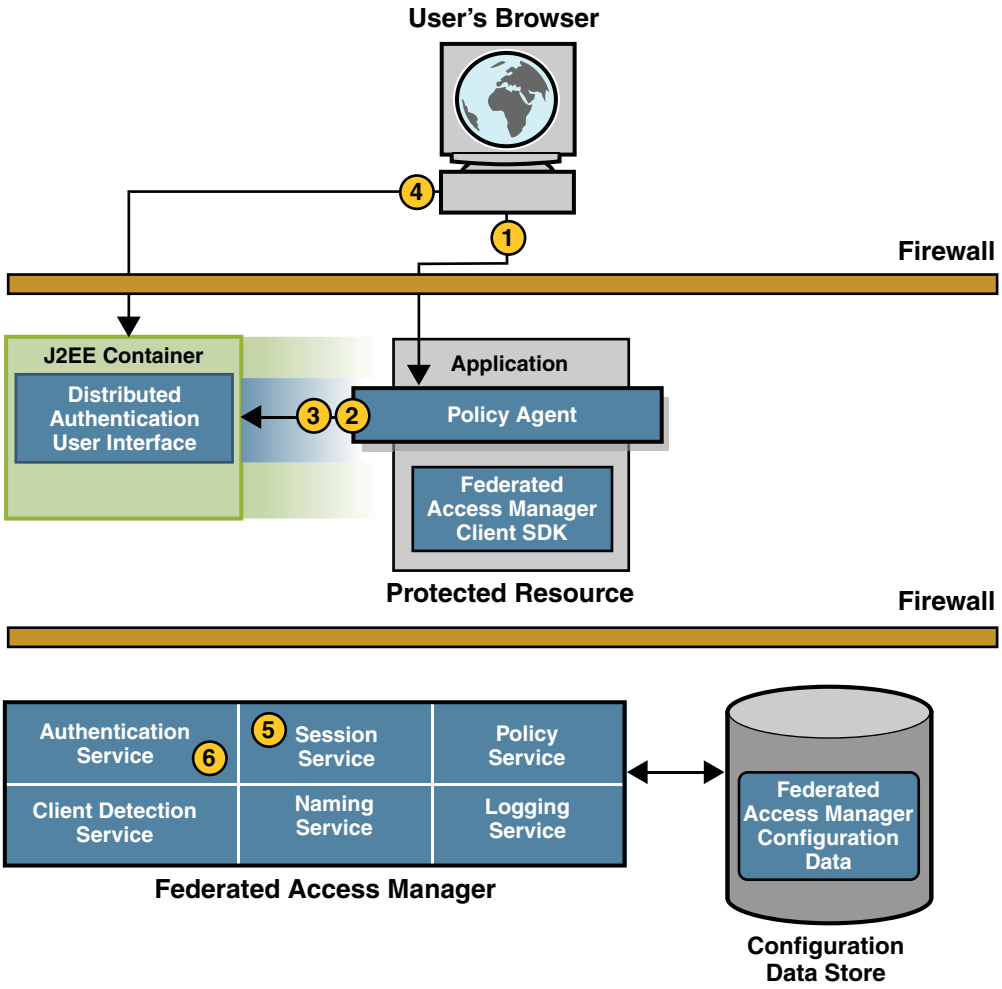


FIGURE 5–1 Initial HTTP Request

1. The user's browser sends an HTTP request to the protected resource.
2. The policy agent inspects the user's request and finds no session token.
3. The policy agent contacts the configured authentication URL.

   In this example, the authentication URL it is set to the URL of the Distributed Authentication User Interface.
4. The browser sends a GET request to the Distributed Authentication User Interface.
5. The Session Service creates a new session (session data structure) and generates a session token (a randomly-generated string that identifies the session).

6. The Authentication Service sets the session token in a cookie.

The next part of the user session is "User Authentication" on page 83.

User Authentication

When the browser sends a GET request to the Distributed Authentication User Interface, the events illustrated in Figure 5–2 occur.

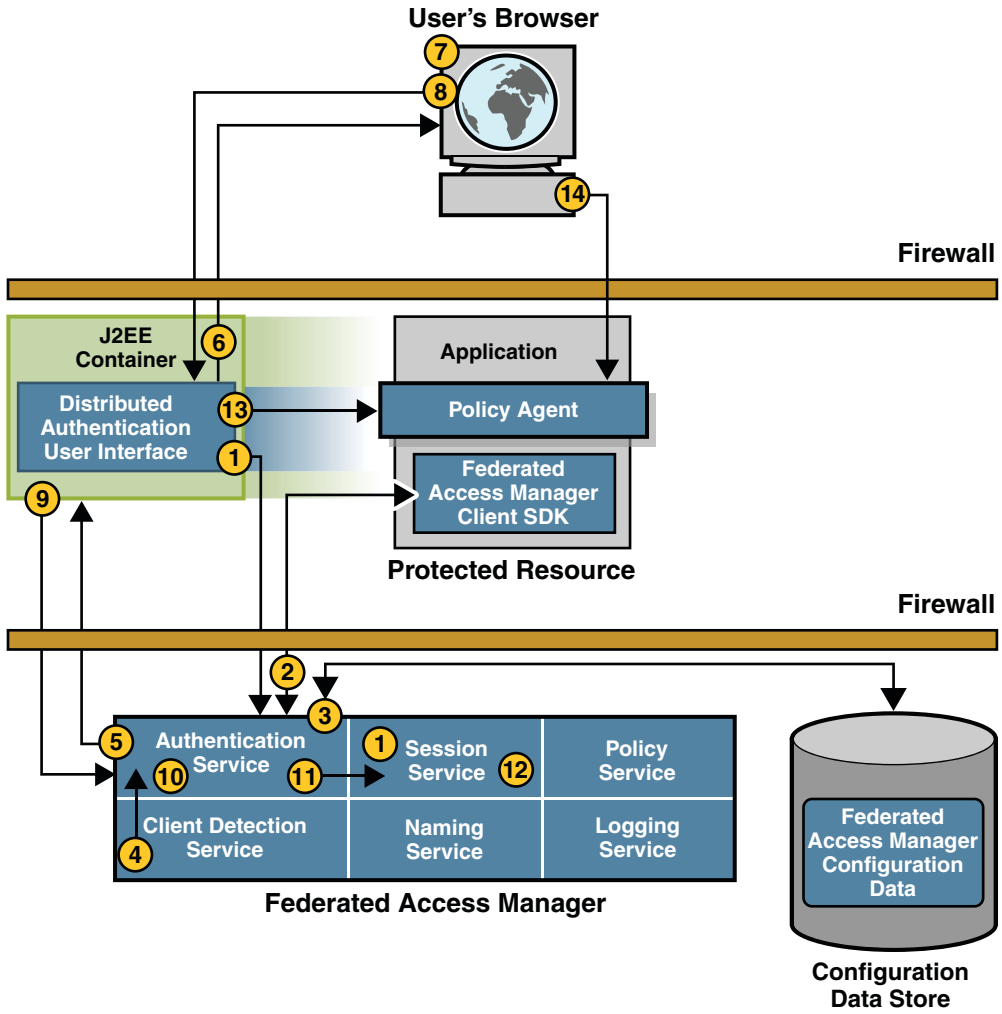


FIGURE 5–2 User Authentication

1. Using the parameters in the GET request, the Distributed Authentication User Interface contacts the Federated Access Manager Authentication Service.
2. The Authentication Service determines the appropriate authentication module to use based upon configuration and the request parameters are passed by the Distributed Authentication User Interface using the Authentication client APIs.

   For example, if Federated Access Manager is configured to use the LDAP Authentication, the Authentication Service determines that the LDAP Authentication login page will be displayed.

3. The Authentication Service determines what should be presented to the user, and sends back to the Distributed Authentication User Interface all necessary credentials, requirements, and callbacks for use by the presentation framework layer.
4. The Client Detection Service determines which protocol, such as HTML or WML, to use to display the login page.
5. The Distributed Authentication User Interface returns a dynamic presentation extraction page along with the session cookie to the browser.

   The presentation extraction page contains the appropriate credentials request and callbacks info obtained from Federated Access Manager.
6. The user's browser displays the login page.
7. The user enters information in the fields of the login page and the browser replies to the Distributed Authentication User Interface with a POST that contains the required credentials.
8. The Distributed Authentication User Interface uses the Authentication client APIs to pass credentials to Federated Access Manager.
9. The Authentication Service uses the appropriate authentication module to validate the user's credentials.

   For example, if the LDAP authentication module is used, the Authentication Service verifies that the username and password provided exist in the LDAP directory. Other authentication module types have different requirements.
10. Assuming authentication is successful, the Authentication Service activates the session by calling the appropriate methods in the Session Service.

    The Authentication Service stores information such as login time, Authentication Scheme, and Authentication Level in the session data structure.
11. Once the session is activated, the Session Service changes the state of the session token to valid.
12. The Distributed Authentication User Interface replies to the protected resource with the valid `SSOToken` in a set-cookie header.
13. Now, the browser makes a second request to the original resource protected by a policy agent.

    This time, the request includes a valid session token, created during the authentication process.

The next part of the user session is "Session Validation" on page 86.

Session Validation

After successful authentication, the user's browser redirects the initial HTTP request to the server a second time for validation. The request now contains a session token in the same DNS domain as Federated Access Manager. The events in Figure 5–3 illustrate this process.

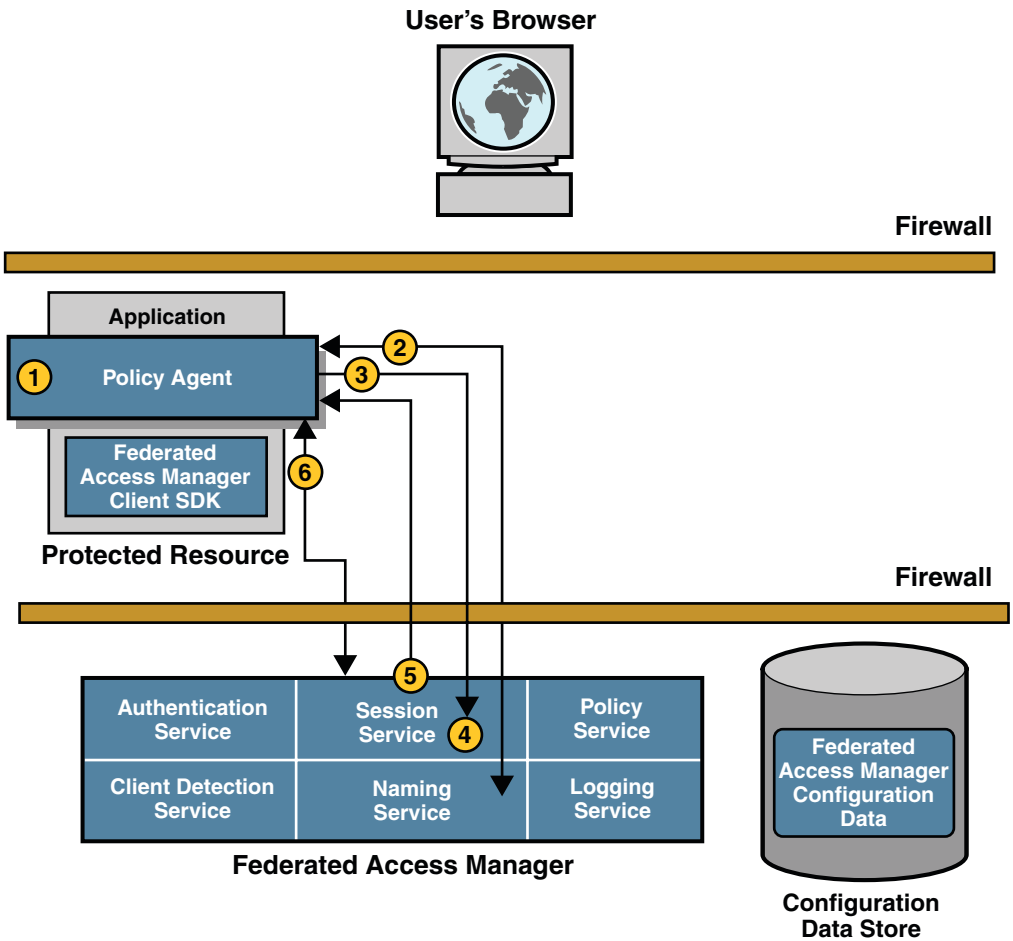


FIGURE 5–3 Session Validation

1. The policy agent intercepts the second access request.
2. The policy agent determines the validity of the session token by contacting the Naming Service to learn where the session token originated

The Naming Service allows clients to find the service URL for Federated Access Manager internal services. When contacted, the Naming Service decrypts the session token and returns the corresponding URLs which will be used by other services to obtain information about the user session.

3. The policy agent, using the information provided by the Naming Service, makes a POST request to the Session Service to validate the included session token.
4. The Session Service receives the request and determines whether the session token is valid based on the following criteria:
   a. Has the user been authenticated?
   b. Does a session data structure associated with the session token exist?
5. If all criteria are met, the Session Service responds that the session token is valid.

   This assertion is coupled with supporting information about the user session itself.
6. The policy agent creates a Session Listener and registers it with the Session Service, enabling notification to be sent to the policy agent when a change in the session token state or validity occurs.

The next part of the user session is "Policy Evaluation and Enforcement" on page 87.

Policy Evaluation and Enforcement

After a session token has been validated, the policy agent determines if the user can be granted access to the server by evaluating it's defined policies. Figure 5–4 illustrates this process.

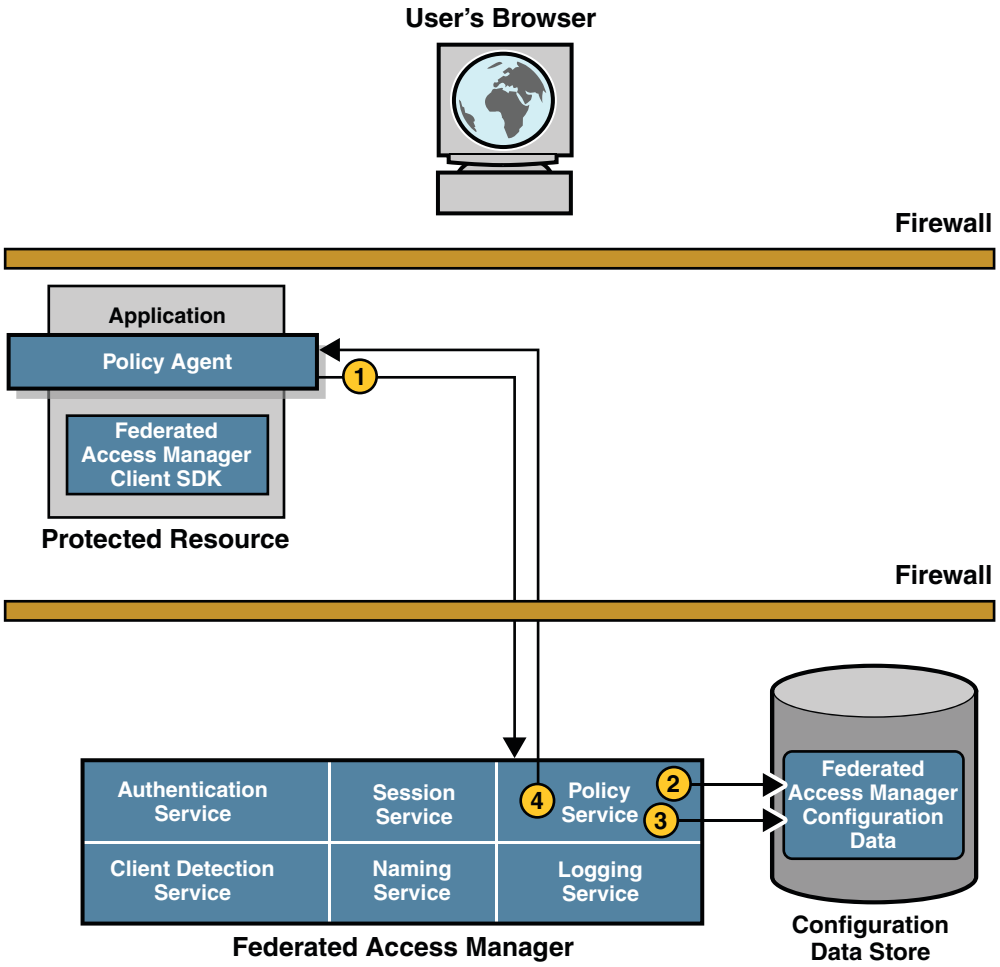


FIGURE 5–4 Policy Evaluation

1. The policy agent sends a request to the Policy Service, asking for decisions regarding resources in it's portion of the HTTP namespace.

   The request also includes additional environmental information. For example, IP address or DNS name could be included in the request because they might impact conditions set on a configuration policy.

2. The Policy Service checks for policies that apply to the request.

   Policies are cached in Federated Access Manager. If the policies have not been cached already, they are loaded from Federated Access Manager.

3. If policies that apply to the request are found, the Policy Service checks if the user identified by the session token is a member of any of the Policy Subjects.

a. If no policies that match the resource are found, the user will be denied access.

b. If policies are found that match the resource, and the user is a valid subject, the Policy Service evaluates the conditions of each policy. For example, *Is it the right time of day?* or *Are requests coming from the correct network?*

   - If the conditions are met, the policy applies.
   - If the conditions are not met, the policy is skipped.

4. The Policy Service aggregates all policies that apply, encodes a final decision to grant or deny access, and responds to the policy agent.

The next part of the basic user session is "Logging the Results" on page 89.

Logging the Results

When the policy agent receives a decision from the Policy Service, the events illustrated in Figure 5–5 occur.

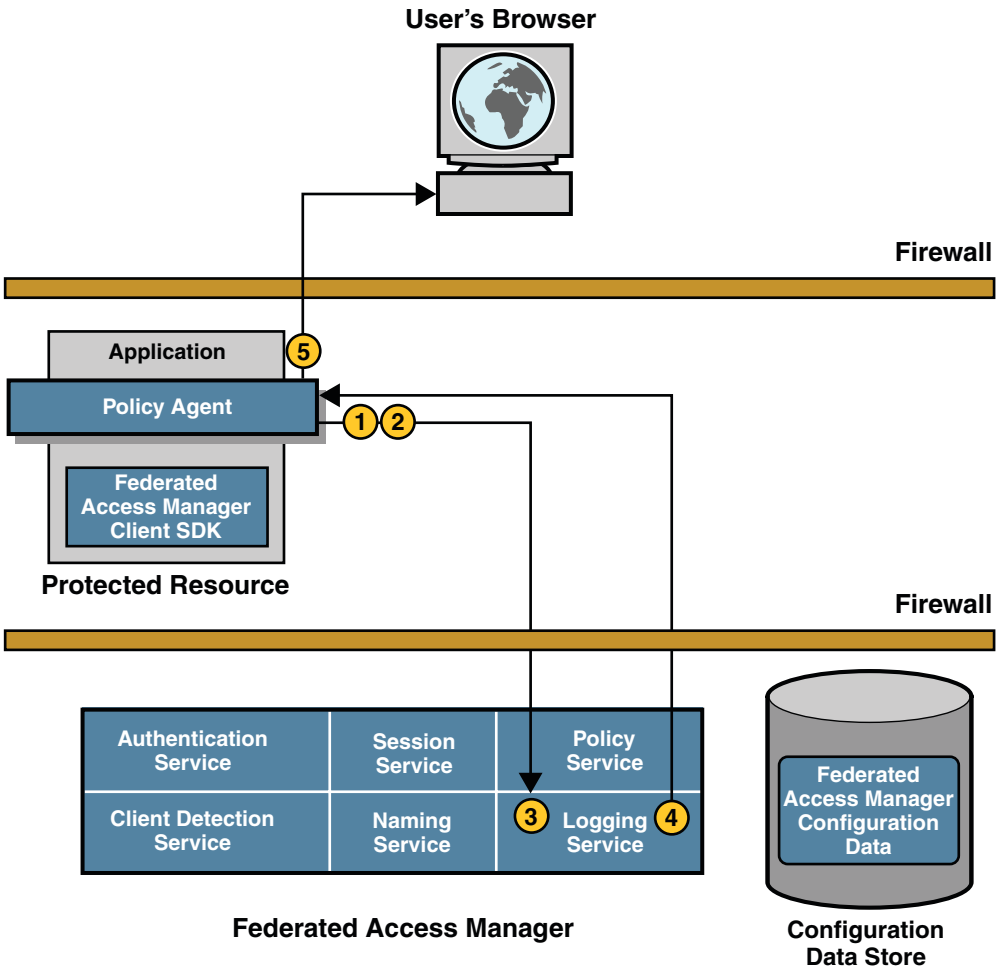


FIGURE 5–5 Logging the Policy Evaluation Results

1. The decision and session token are cached by the policy agent so subsequent requests can be checked using the cache (without contacting Federated Access Manager).

   The cache will expire after a (configurable) interval has passed or upon explicit notification of a change in policy or session status.
2. The policy agent issues a logging request to the Logging Service.
3. The Logging Service logs the policy evaluation results to a flat file (which can be signed) or to a JDBC store, depending upon the log configuration.
4. The Logging Service notifies the policy agent of the new log.
5. The policy agent allows or denies the user access to the application.

a. If the user is denied access, the policy agent displays an "access denied" page.

b. If the user is granted access, the resource displays its access page.

Assuming the browser displays the application interface, this basic user session is valid until it is terminated. See "Session Termination" on page 95 for more information. While logged in, if the user attempts to log into another protected resource, the "Single Sign-On Session" on page 91 begins.

Single Sign-On Session

SSO is always preceded by a basic user session in which a session is created, its session token is validated, the user is authenticated, and access is allowed. SSO begins when the authenticated user requests a protected resource on a second server in the **same** DNS domain. The following process describes an SSO session by tracking what happens when an authenticated user accesses a second application in the same DNS domain as the first application. Because the Session Service maintains user session information with input from all applications participating in an SSO session, in this example, it maintains information from the application the user was granted access to in "Basic User Session" on page 81.

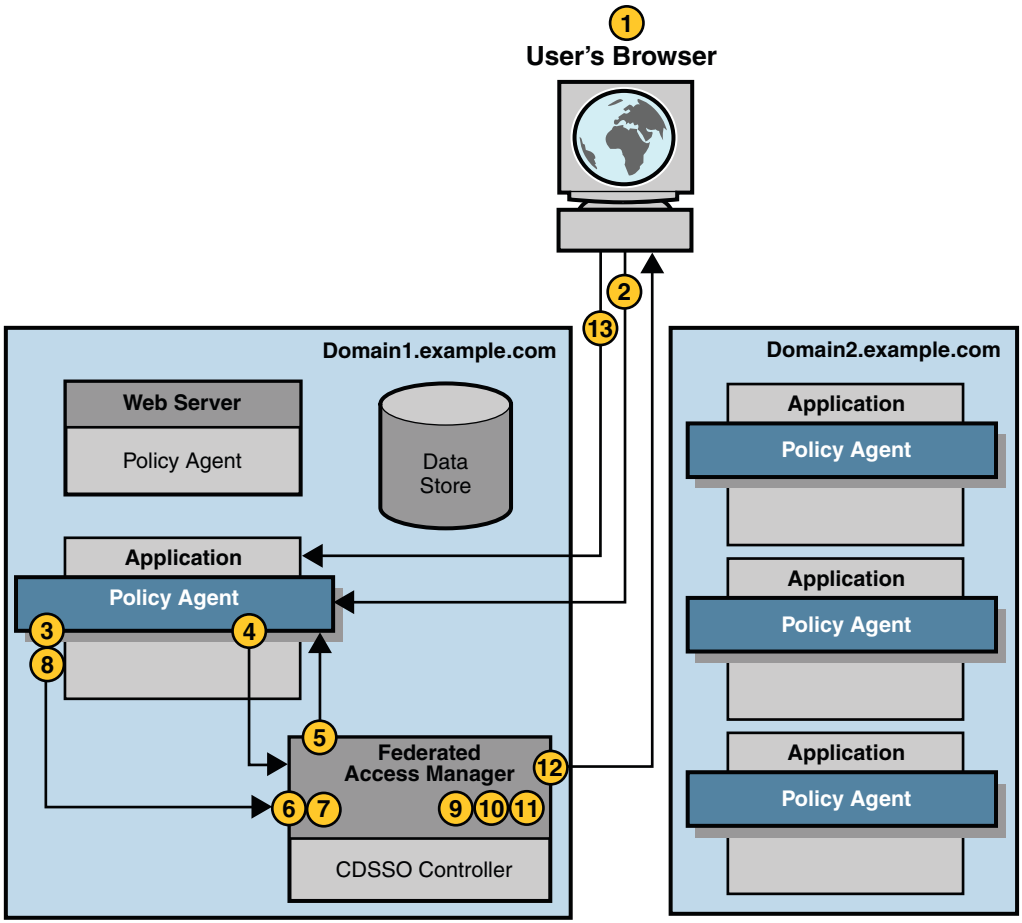


FIGURE 5–6 Single Sign-On Session

1. The user attempts to access a second application hosted on a server in the same domain as the first application to which authentication was successful.
2. The user's browser sends an HTTP request to the second application that includes the user's session token.
3. The policy agent intercepts the request and inspects it to determine whether a session token exists.

   A session token indicates the user is already authenticated. Since the user was authenticated, the Authentication Service is not required at this time. The SSO APIs retrieve the session data structure using the session token imbedded in the cookie. The session data structure is referred to as the *SSOToken* by the SSO APIs. The session token is referred to as the *SSOTokenID*.

4. The policy agent determines the validity of the session.

   For detailed steps, see "Session Validation" on page 86.

5. The Session Service sends a reply to the policy agent indicating whether the *SSOToken* is valid.
   - If the *SSOToken* is not valid, the user is redirected to the Authentication page.
   - If the *SSOToken* is valid, the Session Service creates a Session Listener.

     A Session Listener allows notification to the policy agent when a change in the *SSOToken* state or validity occurs.

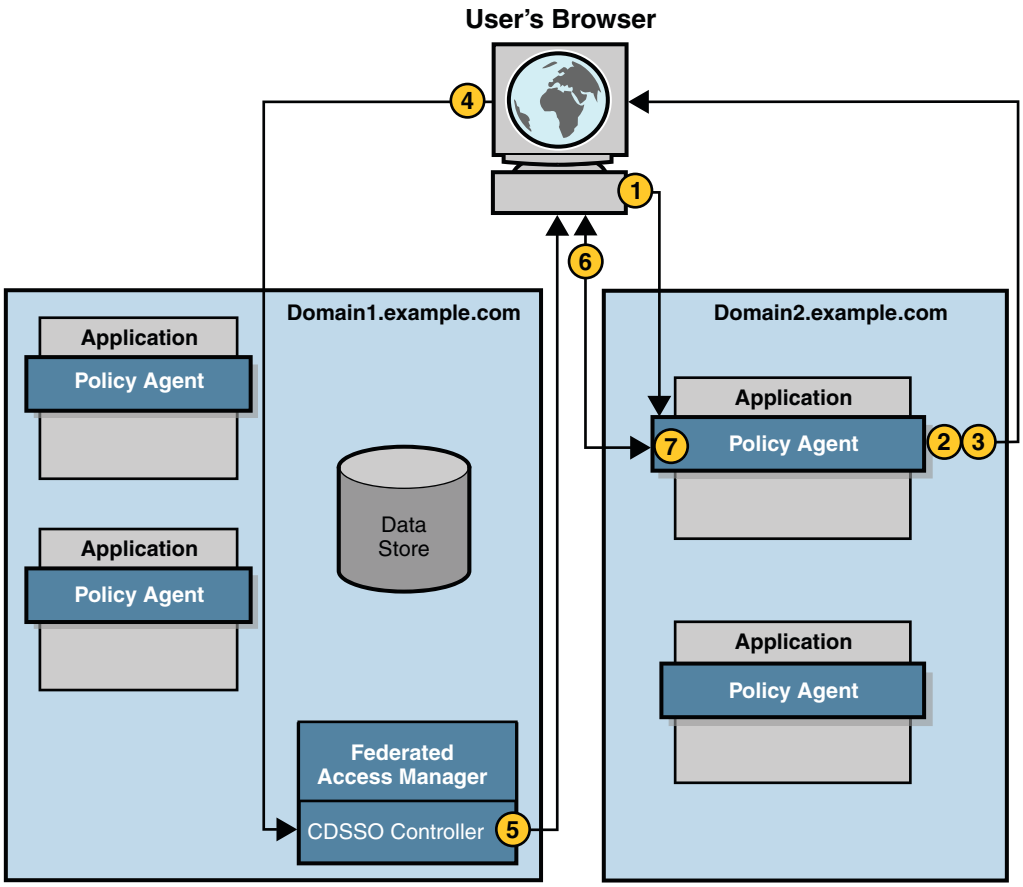
6. The policy agent sends a request for a decision regarding resources in the policy agent's portion of the HTTP namespace to the Policy Service.
7. The Policy Service checks for policies that apply to the request.
   - If Policy Service does not find policy allowing access to the protected resource, the user is denied access and the following events occur:
     a. The Logging Service logs a denial of access.
     b. The policy agent issues a *Forbidden* message to the user.

        The user can then be redirected to an administrator-specified page indicating the user was denied access.
   - If the Policy Service finds policy allowing access to the protected resource, the user is granted access to the protected resource and the SSO session is valid until it is terminated. See "Session Termination" on page 95.

While still logged in, if the user attempts to log in to another protected resource located in a different DNS domain, the "Cross-Domain Single Sign-On Session" on page 93 begins.

Cross-Domain Single Sign-On Session

CDSSO occurs when an authenticated user requests a protected resource on a server in a **different** DNS domain. The user in the previous sections, "Basic User Session" on page 81 and "Single Sign-On Session" on page 91, for example, accessed applications in one DNS domain. In this scenario, the CDSSO Controller within Federated Access Manager transfers the user's session information from the initial domain, making it available to applications in a second domain.



1. The authenticated user's browser sends an HTTP request to the application in a different DNS domain.
2. The policy agent intercepts the request and inspects it to determine if a session token exists for the domain in which the requested application exists. One of the following occurs:
   - If a session token is present, the policy agent validates the session.
   - If no session token is present, the policy agent (which is configured for CDSSO) will redirect the HTTP request to the CDSSO Controller.

   Note – The CDSSO Controller uses Liberty Alliance Project protocols to transfer sessions so the relevant parameters are included in the redirect.

   In this example, no session token for the second domain is found.
3. The policy agent redirects the HTTP request to the CDSSO Controller.

4. The user's browser allows the redirect to the CDSSO Controller.

   Recall that earlier in the user session the session token was set in a cookie in the first domain which is now part of the redirect.

5. The CDC Servlet (in the CDSSO Controller) receives the session token from the first domain, extracts the user's session information, formulates a Liberty POST profile response containing the information, and returns a response to the browser.

6. The user's browser automatically submits the response to the policy agent in the second domain.

   The POST is based upon the Action and the Javascript included in the Body tags `onLoad`.

7. The policy agent in the second domain receives the response, extracts the session information, validates the session, and sets a session token in the cookie for the new DNS domain.

   For detailed information, see "Session Validation" on page 86.

8. The process continues with policy evaluation and results logging.

   See the following sections for detailed information.

   - "Policy Evaluation and Enforcement" on page 87
   - "Logging the Results" on page 89

9. The policy agent allows or denies the user access to the application.

   a. If the user is denied access, the policy agent displays an "access denied" page.

   b. If the user is granted access, the resource displays its access page.

Assuming proper authorization, the policy agent responds to the user by presenting the requested application. The new cookie can now be used by all agents in the new domain, and the session is valid until it is terminated. (See "Session Termination" on page 95.)

Session Termination

A user session can be terminated in any of following ways:

User Ends Session

When a user explicitly logs out of Federated Access Manager by clicking on a link to the Logout Service the following events occur:

1. The Logout Service receives the Logout request, and performs the following steps:

a. Marks user's session as destroyed.
b. Destroys session.
c. Returns a successful logout page to the user.

2. The Session Service notifies applications which are configured to interact with the session.

   In this case, each of the policy agents was configured for Session Notification, and each is sent a document instructing the agent that the session is now invalid.

3. The policy agents flush the session from cache and the user session ends.

Administrator Ends Session

Federated Access Manager administrators with appropriate permissions can terminate a user session at any time. When an administrator uses the Sessions tab in the Federated Access Manager console to end a user's session, the following events occur:

1. The Logout Service receives the Logout request, and performs the following steps:

   a. Marks user's session as destroyed.
   b. Destroys session.

2. The Session Service notifies applications which are configured to interact with the session.

   In this case, each of the policy agents was configured for Session Notification, and each is sent a document instructing the agent that the session is now invalid.

3. The policy agents flush the session from cache and the user session ends.

Federated Access Manager Enforces Timeout Rules

When a session timeout limit is reached, the Session Service completes the following steps:

1. Changes session status to `invalid`.
2. Displays time-out message to user.
3. Starts timer for purge operation delay (default is 60 minutes).
4. When purge operation delay time is reached, purges or destroys the session.
5. If a session validation request comes in after the purge delay time is reached, displays login page to user.

Session Quota Constraints

Federated Access Manager allows administrators to constrain the amount of sessions one user can have. If the user has more sessions than the administrator will allow, one (or more) of the existing sessions can be destroyed.

◆ ◆ ◆ CHAPTER 6

Understanding the Authentication Service

The Sun Federated Access Manager Authentication Service determines whether a user is the person he claims to be. User authentication is the first step in controlling access to web resources within an enterprise. This chapter explains how the Authentication Service works with other components to authenticate a user, or prove that the user's identity is genuine. Topics covered include:

Authentication Service Overview

The function of the Authentication Service is to request information from an authenticating user, validate it against the specified authentication module and, after successful authentication, activate a user session that can be validated across all web applications participating in an SSO environment. The Authentication Service framework has a pluggable architecture for authentication modules that have different user credential requirements. Together with the Session Service, the Authentication Service establishes the fundamental infrastructure for SSO.

From the perspective of the user being authenticated, this is how the Authentication Service works. A company employee wants to look up a colleague's phone number, so he uses a browser to access the company's online phone book. To log in to the phone book service, the employee provides a user name and password. Federated Access Manager compares the user's input with data stored in the appropriate identity data repository. If Federated Access Manager finds a match for the user name, and if the given password matches the stored password, the user's identity is authenticated. The "Basic User Session" on page 81 section in the previous chapter contains a detailed description of the authentication process itself.

The following sections explain some of the features of the Authentication Service.

Account Locking

The Authentication Service provides account locking to prevent a user from completing the authentication process after a specified number of failures. Federated Access Manager sends email notifications to administrators when account lockouts occur. Account locking activities are also logged. Federated Access Manager supports:

Physical Locking. By default, user accounts are `active` or physically unlocked. You can initiate physical locking by changing the value of the Lockout Attribute Name attribute in the user's profile to `inactive`. The account remains physically locked until the attribute is changed to `active`.

Memory Locking. You can enable memory locking by changing the Login Failure Lockout Duration attribute to a value greater then `0`. The user's account is locked in memory for the number of minutes you specified. The account is unlocked after the time period elapses. You can configure Memory Locking so that a user account is locked in memory after a specified number of authentication attempts.

The account locking feature is disabled by default. For information on how to enable it, see XXXXXX "Account Locking" in *Sun Java System Access Manager 7.1 Administration Guide*.

Note – Only authentication modules that throw an `Invalid Password Exception` can leverage the Account Locking feature.

Authentication Chaining

Federated Access Manager allows the configuration of an authentication process in which a user must pass credentials to one or more authentication modules before validation is accomplished. This is called *authentication chaining*. Federated Access Manager uses the Java Authentication and Authorization Service (JAAS) framework (integrated in the Authentication Service) to implement authentication chaining. The JAAS framework validates all user identifiers used during the authentication process, and maps them to the one user. (The mapping is based on the configuration of the User Alias List attribute in the user's profile.) If all the maps are correct,

a user's session token is validated. If all the maps are not correct, the user is denied a valid session token. Once authentication to all modules in the chain succeeds or fails, control is returned to the Authentication Service from the JAAS framework.

You configure an authentication chain by realm, user, role, or service. Determining validation is based upon one of the following control flags being configured for each authentication module instance defined in the chain. The flags are:

Requisite	Authentication to this module instance is required to succeed. If it succeeds, authentication continues down the module instance list. If it fails, control immediately returns to the application.
Required	Authentication to this module instance is required to succeed. If any of the required module instances defined in the chain fails, the whole authentication chain will fail.
Sufficient	The module instance is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the module instance list). If it fails, authentication continues down the list.
Optional	The module instance is not required to succeed. Whether it succeeds or fails, authentication still continues to proceed down the module instance list.

For more information, see XXXXX "Authentication Chaining" in *Sun Java System Access Manager 7.1 Administration Guide*. For information on authentication module instances, see "Authentication Modules" on page 100.

Fully Qualified Domain Name Mapping

Fully Qualified Domain Name (FQDN) mapping enables the Authentication Service to take corrective action in the case where a user may have typed in an incorrect URL. This is necessary, for example, when a user specifies a partial host name or IP address to access protected resources. This feature can also be used to allow access to one instance of Federated Access Manager using many different aliases. For example, you might configure one instance of Federated Access Manager as `intranet.example.com` for employees and `extranet.example.com` for partners. For more information, see XXXXX "Fully Qualified Domain Name Mapping" in *Sun Java System Access Manager 7.1 Administration Guide*.

Persistent Cookies

A *persistent cookie* is an information packet that is written to the user's hard drive and, therefore, continues to exist after the web browser is closed. The persistent cookie enables a user to log into a new browser session without having to reauthenticate. The Authentication Service

can be enabled to write persistent cookies rather than cookies that are written to a web browser's memory. For more information, see XXXXX "Persistent Cookie" in *Sun Java System Access Manager 7.1 Administration Guide*.

Session Upgrade

The Authentication Service allows for the upgrade of a valid session based on a second, successful authentication performed by the same user. If a user with a valid session attempts to authenticate to a second resource secured under the realm to which he is currently authenticated, and this second authentication request is successful, the Authentication Service updates the session with the new properties based on the new authentication. If the authentication fails, the current user session is returned without an upgrade. If the user with a valid session attempts to authenticate to a resource secured in a different realm, the user will receive a message asking whether the user would like to authenticate to the new realm. The user can choose to maintain the current session, or can attempt to authenticate to the new realm. Successful authentication will result in the old session being destroyed and a new one being created. For more information, see XXXXX "Session Upgrade" in *Sun Java System Access Manager 7.1 Administration Guide*.

Validation Plug-in Interface

DEPRECATED: An administrator can write username or password validation logic appropriate for a particular realm, and plug the logic into the Authentication Service. Before authenticating a user or changing the user password, Federated Access Manager will invoke this plug-in. If the validation is successful, authentication continues. If validation fails, an authentication failed page will be thrown. The plug-in extends the `com.iplanet.am.sdk.AMUserPasswordValidation` class which is part of the Service Configuration package. For more information, see "Validation Plug-in Interface" in *Sun Java System Access Manager 7.1 Administration Guide*.

Note – The Validation Plug-In Interface is only supported by the LDAP and Membership authentication module types.

Authentication Modules

An *authentication module* is a plug-in that collects user information such as a user ID and password, and compares the information against entries in a database. If a user provides information that meets the authentication criteria, the user is validated and, assuming the appropriate policy configuration, granted access to the requested resource. If the user provides

information that does not meet the authentication criteria, the user is not validated and denied access to the requested resource. Federated Access Manager is deployed with a number of authentication modules. Table 6–1 provides a brief description of each.

TABLE 6–1 Authentication Service Modules

Authentication Module Name	Description
Active Directory	Uses an Active Directory operation to associate a user identifier and password with a particular Active Directory entry. You can define multiple Active Directory authentication configurations for a realm. Allows both LDAP and Active Directory to coexist under the same realm.
Anonymous	Enables a user to log in without specifying credentials. You can create an Anonymous user so that anyone can log in as Anonymous without having to provide a password. Anonymous connections are usually customized by the Federated Access Manager administrator so that Anonymous users have limited access to the server.
Certificate	Enables a user to log in through a personal digital certificate (PDC). The user is granted or denied access to a resource based on whether or not the certificate is valid. The module can optionally require the use of the Online Certificate Status Protocol (OCSP) to determine the state of a certificate.
Data Store	Enables authentication against one or more configuration data stores within a realm.
HTTP Basic	Enables authentication to occur with no data encryption. Credentials are validated internally using the LDAP authentication module.
Java Database Connectivity (JDBC)	Enables authentication through any Structured Query Language (SQL) databases that provide JDBC-enabled drivers. The SQL database connects either directly through a JDBC driver or through a JNDI connection pool.
LDAP	Enables authentication using LDAP bind, a directory server operation which associates a user identifier and password with a particular LDAP entry. You can define multiple LDAP authentication configurations for a realm.
Membership	Enables user to self-register a user entry. The user creates an account, personalizes it, and accesses it as a registered user without the help of an administrator. Implemented similarly to personalized sites such as `my.site.com`or `mysun.sun.com`.

TABLE 6–1 Authentication Service Modules *(Continued)*

Authentication Module Name	Description
MSISDN	The Mobile Station Integrated Services Digital Network (MSISDN) authentication module enables authentication using a mobile subscriber ISDN associated with a device such as a cellular telephone. It is a non-interactive module. The module retrieves the subscriber ISDN and validates it against the user repository to find a user that matches the number.
RADIUS	Uses an external Remote Authentication Dial-In User Service (RADIUS) server to verify identities.
Security Assertion Markup Language (SAML)	Receives and validates SAML assertions on a target server by using either a web artifact or a POST response.
SafeWord®	Uses Secure Computing's SafeWord PremierAccess™ server software and SafeWord tokens to verify identities.
SecurID™	Uses RSA ACE/Server software and RSA SecurID authenticators to verify identities.
UNIX®	Solaris and Linux modules use a user's UNIX identification and password to verify identities.
Windows Desktop Single Sign-On (SSO)	Allows a user who has already authenticated with a key distribution center to be authenticated by Federated Access Manager without having to provide the login information again. Leverages Kerberos authentication and is specific to the Windows operating system.
Windows NT	Uses a Microsoft Windows NT™ server to verify identities.

You can use the Federated Access Manager console to enable and configure the authentication modules that are installed by default. You can also create and configure multiple instances of a particular authentication module. (An *authentication module instance* is a child entity that extends the schema of a parent authentication module and adds its own subschema.) Finally, you can write your own custom authentication module (or plug-in) to connect to the Federated Access Manager authentication framework. See XXXXXX Chapter 4, "Managing Authentication," in *Sun Java System Access Manager 7.1 Administration Guide* for detailed information about enabling and configuring default authentication modules and authentication module instances. See Chapter 2, "Using the Authentication Interfaces," in *Sun Java System Federated Access Manager 8.0 Developer's Guide* for more information about writing custom authentication modules.

Configuring for Authentication

The authentication framework includes the following pluggable and customizable services:

GO OVER THIS SECTION WITH NEW CONSOLE

Realm Authentication Configuration

The General Authentication Service is used for server-related attribute configuration. (Some of the attributes described in this service are default attributes for all Federated Access Manager authentication modules.) Configuration is available in each configured realm. The General Authentication Service enables the administrator to define default values for a realm's authentication parameters. These values can be used if no overriding value is defined in the specified authentication module. The default values for the General Authentication Service are defined in the `amAuth.xml` file and stored in the configuration data store. For more information, see XXXXX.

Authentication Configuration Service

The Authentication Configuration Service describes all the dynamic attributes for service-based authentication. This service is used for configuring roles. When you assign a service to a role, you can also assign other attributes such as a success URL or an authentication post-processing class to the role. For more information, see XXXXX "Role-based Authentication" in *Sun Java System Access Manager 7.1 Administration Guide*.

Authentication Service User Interface

The Authentication Service implements a user interface that is separate from the Federated Access Manager administration console. The Authentication Service user interface provides a dynamic and customizable means for gathering authentication credentials. When a user requests access to a protected resource, the Authentication Service presents a web-based login page and prompts the user for user name and password. Once the credentials have been passed back to Federated Access Manager and authentication is successful, the user can gain access based on the user's specific privileges:

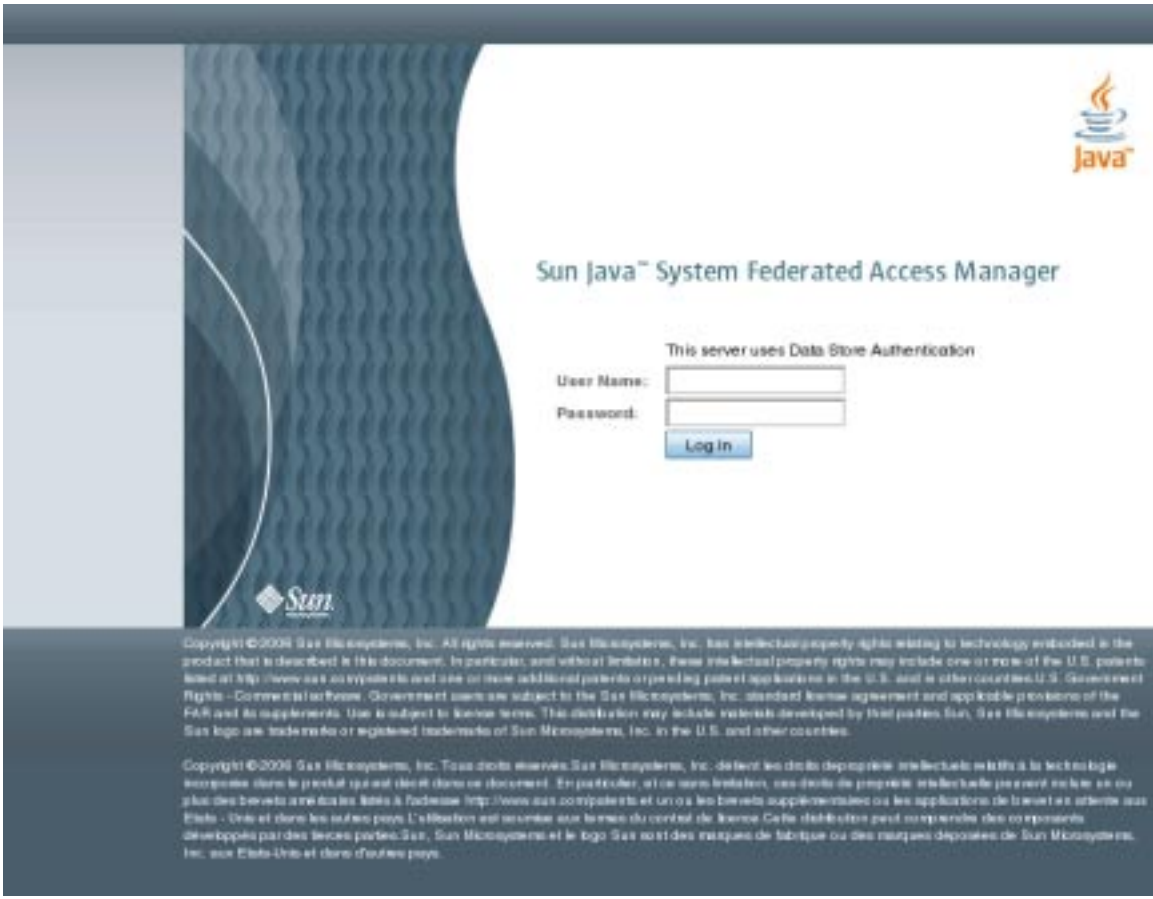


FIGURE 6–1 Authentication Service User Interface

The Authentication Service user interface can be used for the following:

- Administrators can access the administration portion of the Federated Access Manager console to manage their realm's identity data.
- Users can access their own profiles to modify personal data.
- A user can access a resource defined as a redirection URL parameter appended to the login URL.
- A user can access the resource protected by a policy agent.

Federated Access Manager provides customization support for the Authentication Service user interface. You can customize JavaServer Pages™ (JSP™) and the file directory level for

`/org/service/locale/client_type`. See Chapter 16, "Customizing the Authentication User Interface," in *Sun Java System Federated Access Manager 8.0 Developer's Guide* for more information.

Distributed Authentication User Interface

Federated Access Manager provides a remote authentication user interface component to enable secure, distributed authentication across two firewalls. A web browser communicates an HTTP request to the remote authentication user interface which, in turn, presents a login page to the user. The web browser then sends the user login information through a firewall to the remote authentication user interface which, in turn, communicates through the second firewall with Federated Access Manager. The Distributed Authentication User Interface enables a policy agent or an application that is deployed in a non-secured area to communicate with the Federated Access Manager Authentication Service installed in a secured area of the deployment. Figure 6–2 illustrates this scenario.

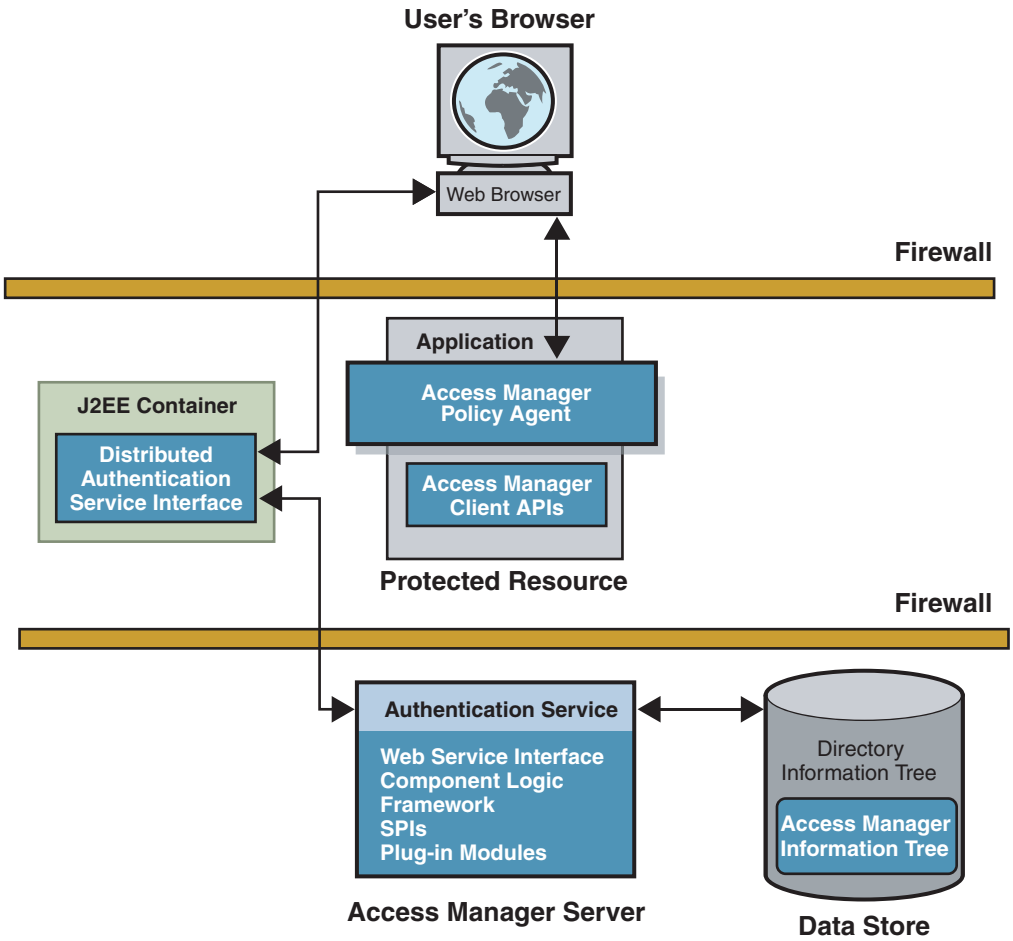


FIGURE 6–2 Distributed Authentication Process

The Distributed Authentication User Interface uses a JATO presentation framework and is customizable. (See screen capture in "Authentication Service User Interface" on page 103.) You can install the Distributed Authentication User Interface on any servlet-compliant web container within the non-secure layer of a Federated Access Manager deployment. The remote component then works with the Authentication client APIs and authentication utility classes to authenticate web users. For a more detailed process, see "User Authentication" on page 83. For detailed installation and configuration instructions, see XXXXXX Chapter 11, "Deploying a Distributed Authentication UI Server," in *Sun Java System Access Manager 7.1 Postinstallation Guide*.

Inside the Core Authentication Component

The core Authentication component is where default configurations are stored and authentication processes are invoked. This section contains information on the following components.

Client Detection Service

Because the Authentication Service is client-type aware, the initial step in the authentication process is to identify the type of client making the HTTP(s) request. This feature is known as *client detection*. The URL information in the HTTP(s) request is used to retrieve the client's characteristics. Based on these characteristics, the appropriate authentication pages are returned. For example, when a web browser is used for requesting access, an HTML login page will be displayed. Once the user is authenticated, the client type is added to the session token for future use. For more information, see Chapter 11, "Identifying the Client Type," in *Sun Java System Federated Access Manager 8.0 Developer's Guide*.

Note – Federated Access Manager supports all configured client types such as cookie-less and cookie-enabled client types.

Authentication Type Configurations

After granting or denying access to a resource, Federated Access Manager checks for information about where to redirect the user. A specific order of precedence is used when checking for this information. The order is based on whether the user was granted or denied access to the protected resource, and on the type of authentication specified. When you install Federated Access Manager, a number of authentication types are automatically configured.

Realm-based Authentication.	User authenticates to a configured realm or sub-realm.
Role-based Authentication.	User authenticates to a configured role within a realm or sub-realm. The user must possess the role. A *static role* is possessed when an attribute is assigned to a specific user or container. A *filtered role* is dynamically generated based on an attribute contained in the user's or container's entry. For example, all users that

	contain a value for the `employee` attribute can be included in a role named *employees* when the filtered role is created.
Service-based Authentication.	User authenticates to a specific service or application registered to a realm or sub-realm.
User-based Authentication.	User authenticates using an authentication process configured specifically for him or her.
Authentication Level-based Authentication	An administrator specifies the security level of the authentication modules by defining each with an *authentication level*. Successful authentication to a higher authentication level defines a higher level of trust for the user. If a user attempts to access a service, the service can determine if the user is allowed access by checking the authentication level in the user's session data. If the authentication level is not high enough, the service redirects the user to go through an authentication process with a set authentication level.
Module-based Authentication.	Allows a user to specify the module to which they will authenticate.
Organization-based Authentication.	User authenticates to an organization or sub-organization.

Note – This authentication type only applies to Federated Access Manager when installed in Legacy mode.

For each of these methods, the user can either pass or fail the authentication. Once the determination has been made, each method follows this procedure. Step 1 through Step 3 follows a successful authentication; Step 4 follows both successful and failed authentication. 1. Access Manager confirms whether the authenticated user(s) is defined in the Directory Server data store and whether the profile is active. The User Profile attribute in the Core Authentication module can be defined as Required, Dynamic, Dynamic with User Alias, or Ignored. Following a successful authentication, Access Manager confirms whether the authenticated user(s) is defined in the Directory Server data store and, if the User Profile value is Required, confirms that the profile is active. (This is the default case.) If the User Profile is Dynamically Configured, the Authentication Service will create the user profile in the Directory Server data store. If the User Profile is set to Ignore, the user validation will not be done. 2. Execution of the Authentication Post Processing SPI is accomplished. The Core Authentication

module contains an Authentication Post Processing Class attribute which may contain the authentication post-processing class name as its value. AMPostAuthProcessInterface is the post-processing interface. It can be executed on either successful or failed authentication or on logout. 3. The following properties are added to, or updated in, the session token and the user?s session is activated. realm. This is the DN of the realm to which the user belongs. Principal. This is the DN of the user. Principals. This is a list of names to which the user has authenticated. (This property may have more then one value defined as a pipe separated list.) UserId. This is the user?s DN as returned by the module, or in the case of modules other than LDAP or Membership, the user name. (All Principals must map to the same user. The UserID is the user DN to which they map.) Note ? This property may be a non-DN value. UserToken. This is a user name. (All Principals must map to the same user. The UserToken is the user name to which they map.) Host. This is the host name or IP address for the client. authLevel. This is the highest level to which the user has authenticated. AuthType. This is a pipe separated list of authentication modules to which the user has authenticated (for example, module1|module2|module3). clientType. This is the device type of the client browser. Locale. This is the locale of the client. CharSet. This is the determined character set for the client. Role. Applicable for role-based authentication only, this is the role to which the user belongs. Service. Applicable for service-based authentication only, this is the service to which the user belongs. 4. Access Manager looks for information on where to redirect the user after either a successful or failed authentication. URL redirection can be to either an Access Manager page or a URL. The redirection is based on an order of precedence in which Access Manager looks for redirection based on the authentication method and whether the authentication has been successful or has failed. This order is detailed in the URL redirection portions of the following authentication methods sections.

For more information, see XXXXXX "Authentication Types" in *Sun Java System Access Manager 7.1 Administration Guide*.

Login URLs and Redirection URLs

In the last phase of the authentication process, Federated Access Manager either grants or denies access to the user. If access is granted, Federated Access Manager uses a login URL to display a page in the browser. If access is denied, Federated Access Manager uses a redirection URL to display an alternate page in the browser. A typical alternate page contains a brief message indicating the user has been denied access.

Each authentication type (as discussed in "Authentication Type Configurations" on page 107) uses a login URL or redirection URL based on a specific order of precedence, and on whether the authentication succeeded or failed. For a detailed description of how Federated Access Manager proceeds through the order of precedence, see XXXXX "Authentication Types" in *Sun Java System Access Manager 7.1 Administration Guide*.

JAAS Shared State

The JAAS shared state enables sharing of both a user identifier and a password between authentication module instances. Options are defined for each authentication module type by realm, user, service and role. If an authentication fails with the credentials from the shared state, the authentication module restarts the authentication process by prompting for its required credentials. If it fails again, the module is marked failed. After a commit, an abort, or a logout, the shared state will be cleared. For more information, see "JAAS Shared State" in *Sun Java System Access Manager 7.1 Administration Guide*.

Authentication Programming Interfaces

Federated Access Manager provides both Java APIs and C APIs for writing authentication clients that remote applications can use to gain access to the Authenticate Service. Communication between the APIs and the Authentication Service occurs by sending XML messages over HTTP(S). The Java and C APIs support all authentication types supported by the browser-based user interface. Clients other than Java and C clients can use the XML/HTTP interface directly to initiate an authentication request.

Additionally, you can add custom authentication modules to Federated Access Manager by using the service provider interface (SPI) package, `com.iplanet.authentication.spi`. This SPI implements the JAAS `LoginModule`, and provides additional methods to access the Authentication Service and module configuration properties files. Because of this architecture, any custom JAAS authentication module will work within the Authentication Service.

For more information, see Chapter 2, "Using Authentication APIs and SPIs," in *Sun Java System Access Manager 7.1 Developer's Guide*.

◆ ◆ ◆ CHAPTER 7

Authorization and the Policy Service

The Sun Federated Access Manager Policy Service determines if a user has been given permission by a recognized authority to access a protected resource. The process is referred to as user authorization. This chapter describes how the various parts of the Policy Service work together to perform authorization. Topics covered include:

Authorization Overview

A *policy* is a rule that defines who is authorized to access a resource. A single policy can define either binary or non-binary decisions. A binary decision is yes/no, true/false or allow/ deny. A non-binary decision represents the value of an attribute. For example, a mail service might include a `mailboxQuota` attribute with a maximum storage value set for each user. In general, a policy is configured to define what a subject can do to which resource and under what conditions.

The Access Manager Policy Service allows administrators to define, modify, and delete policies for protected resources within the Access Manager deployment. Configured policies are grouped into *realms* and stored in the Access Manager information tree. The Policy Service relies on the following:

- A Policy Enforcement Point (PEP) protects an enterprise's resources by enforcing access control. The policy agent is the PEP.
- A Policy Decision Point (PDP) is where the policy is evaluated and a determination is made. The Policy Service is the PDP.

- A data store in which configured policies are stored and from which they are retrieved. The proprietary Access Manager information tree is the data store.

The Access Manager Policy Service uses configured policies to determine if a user has been given permission by a recognized authority to access a protected resource. When a user attempts to access a resource protected by a PEP, the PEP contacts the PDP to get a policy decision. The Policy Service evaluates the policies that protect the resource and are applicable to the requesting user. This results in a policy decision indicating whether the user is allowed to access the resource. Upon receiving the decision, the PEP allows or denies access accordingly. This whole process is referred to as *authorization*.

Access Control and Realms

When a user logs into an application, Access Manager plug-ins retrieve all user information, authentication properties, and authorization policies that the Access Manager framework needs to form a temporary, virtual user identity. The Authentication Service and the Policy Service use this virtual user identity to authenticate the user and enforce the authorization policies, respectively. All user information, authentication properties, and authorization policies is contained in realms. You can create a realm when you want to apply policies to a group of related subjects, services or servers. For example, you can create a realm that groups all servers and services that are accessed regularly by employees in one region. And, within that regional grouping realm, you can group all servers and services accessed regularly by employees in a specific division such as Human Resources. A configured policy might state that all Human Resources administrators can access the URL `http://HR.example.com/HRadmins/index.html.`. You might also add constraints to this policy: it is applicable only Monday through Friday from 9:00 a.m. through 5:00 p.m. Realms facilitate the delegation of policy management privileges.

Note – Access control realms can be configured to use any user database.

Policy Types

The Policy Service authorizes access to a user based on the policies stored in the Access Manager information tree. The following sections contain information on the two types of policies you can create using Access Manager:

- "Normal Policy" on page 113
- "Referral Policy" on page 115

Normal Policy

A *normal policy* specifies a protected resource and who is allowed to access the resource. The protected resource can be anything hosted on a protected server. Examples of protected resources are applications, document files, images, or the server itself. Only a Top-Level Realm or Policy Administrator can create or manage polices that apply to a resource. A normal policy consists of rules, subjects, conditions, and response providers. The following sections contain information regarding these elements.

Rules

A *rule* defines the policy itself by specifying a resource, one or more sets of an action, and values for each action.

- A *resource* defines the specific object that is being protected. Examples of protected objects are an HTML page on a web site, or a user's salary information accessed using a human resources service.
- An *action* is the name of an operation that can be performed on the resource. Examples of web page actions are POST and GET. An allowable action for a human resources service might be `canChangeHomeTelephone`.
- A *value* defines the permission for the action. Examples are `allow` and`deny`.

Subjects

A *subject* specifies, by implication, the user or collection of users that the policy affects.

You can implement custom subjects by using the Policy APIs. You can assign the following subjects to policies:

Access Manger Roles	The roles you create and manage under the Realms Subject tab can be added as a value of the subject.
Access Manager Identity	The identities you create and manage under the Realms Subject tab can be added as a value of the subject.
Authenticated Users	Any user with a valid SSOToken is a member of this subject. All authenticated users would be member of this Subject, even if they have authenticated to a realm that is different from the realm in which the policy is defined.
LDAP Groups	Any member of an LDAP group can be added as a value of this subject.

LDAP Roles	Any LDAP role can be added as a value of this subject. An LDAP Role is any role definition that uses the Sun Java System Directory Server role capability. These roles have object classes mandated by Directory Server role definition. The LDAP Role Search filter can be modified in the Policy Configuration Service to narrow the scope and improve performance.
LDAP Users	Any LDAP user can be added as a value of this subject.
Organization	Any realm can be added as a value of this subject
Web Services Clients	Valid values are the DNs of trusted certificates in the local JKS keystore, which corresponds to the certificates of trusted web service clients (WSCs). A WSC identified by the SSOToken is a member of this subject, if the DN of any principal contained in the SSOToken matches any selected value of this subject. This subject has dependency on the Access Manager implementation of the Liberty Alliance Project Identity Web Services Framework and should be used only by web service providers to authorize WSCs.

Conditions

A *condition* specifies additional constraints that must be satisfied for a policy be applicable. For example, you can define a condition to limit a user's network access to a specific time period. The condition might state that the subject can access the network only between 7:00 in the morning and 10:00 at night.

You can implement custom conditions by using the Policy APIs. Access Manager provides the following conditions:

Active Session Time	Sets a condition based on constraints configured for user session time such as maximum session time.
Authentication Chain	The policy is applicable if the user has successfully authenticated to the authentication chain in the specified realm. If the realm is not specified, authentication to any realm at the authentication chain will satisfy the condition.
Authentication Level	The Authentication Level attribute indicates the level of trust for authentication. The policy is applicable if the user's authentication level is greater than or equal to the Authentication Level set in the condition, or if the user's authentication level is less than or equal to the Authentication Level set in the condition, depending on the configuration.

Authentication Module Instance	The policy applies if the user has successfully authenticated to the authentication module in the specified realm. If the realm is not specified, authentication to any realm at the authentication module will satisfy the condition.
IP Address/DNS Names	Sets a condition based on a range of IP Addresses, or a DNS name.
Current Session Properties	Decides whether a policy is applicable to the request based on values set in the user's Access Manager session.
LDAP Filter Condition	The policy is applicable when the defined LDAP filter locates the user entry in the LDAP directory that was specified in the Policy Configuration service.
Realm Authentication	The policy applies if the user has authenticated to the specified realm.
Time	Sets the condition based on time constraints (time, day, date, time zone).

Response Providers

Response providers are plug-ins that provide policy response attributes. Policy response attributes typically provide values for attributes in the user profile. The attributes are sent with policy decisions to the PEP which, in turn, passes them in headers to an application. The application typically uses these attributes for customizing pages such as a portal page. Access Manager includes one implementation, the `IDResponseProvider`. You can implement custom response providers by using the Policy APIs.

Referral Policy

A Realm Administrator or Policy Administrator at the root or top level of the Access Manager information tree can create policy for any resource. A *referral policy* enables a Realm Administrator or a Policy Administrator to delegate policy configuration tasks. A referral policy delegates both policy creation and policy evaluation, and consists of one or more rules and one or more referrals.

- A *rule* defines the resource whose policy creation or evaluation is being referred.
- A *referral* defines the identity object to which the policy creation or evaluation is being referred.

Referral policies delegate policy management privileges to another entity such as a peer realm, a subrealm, or even a third-party product. (You can implement custom referrals by using the

Policy APIs.) For example, a top-level realm exists named `ISP`. It contains two subrealms: `company1` and `company2`. The Top-Level Administrator for `ISP` can delegate policy management privileges so that a Realm Administrator in `company1` can create and manage policies only within the`company1` realm, and a Realm Administrator in `company2` can create and manage policies only within the `company2` realm. To do this, the Top-Level Administrator creates two referral policies, defining the appropriate realm in the rule and the appropriate administrator in the referral.

Note – An administrator or Policy Administrator for realms configured below the root level of the Access Manager information tree have permission to create policies only for resources delegated to that realm.

Policy Framework

The Policy framework in Access Manager are the services where policy management and administration are implemented. The Policy framework includes the following:

- "Policy Service" on page 116
- "Policy Configuration Service" on page 116

Policy Service

The Policy Service is defined using the `amPolicy.xml`. It performs the following functions:

- Provides a means for defining and managing access policies.
- Provides a means for defining custom policy plug-ins by providing names and class locations.
- Evaluates access policies.
- Acts as a PDP to deliver the result of a policy evaluation.

In order to configure for custom policy plug-ins, modify `amPolicy.xml` and use `amadmin` to reload it. See "Developing Custom Subjects, Conditions, Referrals, and Response Providers" in *Sun Java System Access Manager 7.1 Developer's Guide*.

Policy Configuration Service

The Policy Configuration Service provides a means to specify how policies are defined and evaluated. The Policy Configuration Service enables you to specify, for example:

- Which directory to use for subject lookup

- The directory password
- Which search filters to use
- Which subjects, conditions, and response providers to use

This configuration can be done within a realm or a subrealm and is accessible through the Access Manager console.

Policy SPIs and Plug-Ins Layer

Access Manager includes SPIs that work with the Policy framework to create and manage policies. You can develop customized plug-ins for creating custom policy subjects, referrals, conditions, and response providers. For information on creating custom policy plug-ins, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

The following table summarizes the Policy service provider interfaces (SPIs), and lists the specialized Policy plug-ins that come bundled with Access Manager.

TABLE 7–1 Policy Service Provider Interfaces

Interface	Description
Subject	Defines a set of authenticated users for whom the policy applies. The following Subject plug-ins come bundled with Access Manager: Access Manager Identity Subject, Access Manager Roles, Authenticated Users, LDAP Groups, LDAP Roles, LDAP Users, Organization Web, and Services Clients.
Referral	Delegates management of policy definitions to another access control realm.
Condition	Specifies applicability of policy based on conditions such as IP address, time of day, authentication level. The following Condition plug-ins come bundled with Access Manager: Authentication Level, Authentication Scheme, IP Address, LE Authentication Level, Session, SessionProperty, and Time.
Resource Name	Allows a pluggable resource.
Response Provider	Gets attributes that are sent along with policy decision to the policy agent, and used by the policy agent to customize the client applications. Custom implementations of this interface are now supported in Access Manager 7.1.

Policy Client APIs

Access Manager provides client APIs that implement policy evaluation logic on a remote web server or application server. For policy client API information, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

XACML

Extensible Access Control Markup Language (XACML) is an OASIS standard that describes both a policy language and an access control decision request/response language (both encoded in XML). The policy language is used to describe general access control requirements, and has standard extension points for defining new functions, data types, and combining logic. The request/response language lets you form a query, to ask whether or not a given action should be allowed, and interpret the result. The response always includes one of four response values: Permit, Deny, Indeterminate or Not Applicable. In Federated Access Manager the XACML implementation provides a more standard/interoperable way of retrieving policy evaluation decisions. This would facilitate implementation of policy evaluation points (PEP) that use XACML defined schema. Customers, third parties, or other Sun groups may use XACML policy evaluation requests to get back an XACML response using the SAML v2 protocol. SAML v2, in turn, would use a SOAP binding. Policies would still be configured using Federated Access Manager as previously but the response sent back would be encoded using XACML.

◆ ◆ ◆ CHAPTER 8

Delivering Identity Web Services

The goal of Identity Web Services in Federated Access Manager would be to deliver simple web services that expose the following security-related functions:

- authenticate
- authorize (an authenticated identity's access to a resource)
- attributes (of an authenticated identity)
- log by providing web services in both the SOAP/WSDL style (which the SOA/BI community prefers) and the REST style (which the Web 2.0 community prefers)

- "About Identity Web Services" on page 119

About Identity Web Services

Keeping these web services very simple (such that the WSDL completely defines the input arguments and return values) allows an application developer to consume them simply by pointing an IDE to the service and allowing the IDE to generate the stub code that wraps a call to a web service. (Prototype currently supports Eclipse, NetBeans, and MS Visual Studio.) Client libraries for REST-style web services are a convenience in building requests and parsing responses. These web services are exposed via a well-defined URI.

Federated Access Manager provides client interfaces for authentication, authorization, session, identity management and auditing in Java, in C (C++) and in HTTP(S)/XML. These interfaces are used by web and Java EE policy agents as well as custom applications developed externally. With recent advancements and adoption of Web Services, Service Oriented Architecture (SOA) and Representational State Transfer (REST), developer communities like Web 2.0 and few customers are expecting FAM to deliver simple web services that expose the functions of Sun's Identity Suite. Although standards based on web services have been specified for authentication and single-sign-on (Liberty and SAML), authorization (XACML) and identity management (SPML), customer expectations are for a very simple interfaces that can be used for rapid development. Secondly, the standards specifications are considered to be heavy weight and to

be used for interoperability with other partners and vendors. Hence the goal of Identity Web Services in FAM/OpenSSO would be to deliver simple web services that expose the following security-related functions: ? authenticate ? authorize (an authenticated identity's access to a resource) ? attributes (of an authenticated identity) ? log by providing web services in both the SOAP/WSDL style (which the SOA/BI community prefers) and the REST style (which the Web 2.0 community prefers).

PART III

Federation Using Federated Access Manager

This third part of the Sun Federated Access Manager Technical Overview contains information on the Federation features of Federated Access Manager. It contains the following chapters:

- Chapter 9, "Implementing Federation"

◆ ◆ ◆ CHAPTER 9

Implementing Federation

Sun Java™ System Federated Access Manager provides a framework for implementing a federated identity infrastructure, enabling single sign-on, provisioning users dynamically, and sharing identity attributes across security domains. Forming trust relationships across security domains allows an organization to integrate applications offered by different departments or divisions within the enterprise as well as engage in relationships with cooperating business partners that offer complementary services. Towards this end, multiple industry standards, such as those developed by the Organization for the Advancement of Structured Information Standards (OASIS) and the Liberty Alliance Project, are supported. This chapter contains an overview of the federation framework and federation options.

Federating Identities

Identity federation allows users to link the many personal online identities they have created with 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-establish identity. The following sections contain information about the underlying concepts regarding a framework for federating identities.

The Concept of Identity

In one dictionary, *identity* is defined as "a set of information by which one person is definitively distinguished." This information undoubtedly begins with the document that corroborates a person's name: a birth certificate. Over time, additional information further defines different aspects of an individual's identity:

- An address
- A telephone number
- One or more diplomas
- A driver's license
- A passport
- Financial institution accounts
- Medical records
- Insurance statements
- Employment records
- Magazine subscriptions
- Utility bills

Each of these represents data that designates a piece of a person's identity as it relates to the enterprise for which the data was defined. The composite of this data constitutes an overall identity with each specific piece providing a distinguishing characteristic.

Because the Internet is one of the primary vehicles for the types of interactions represented by this identity-defining information, people are now creating online identities specific to the businesses with which they are interacting. By creating a user account with an identifier and password, an email address, personal preferences (such as style of music, or opt-in/opt-out marketing decisions) and other information specific to the particular business (a bank account number or ship-to address), a user is able to distinguish their account from others who also use the enterprise's services. This distinguishing information is referred to as a *local identity* because it is specific to the service provider for which it has been defined.

Considering the number of service providers for which a user can define a local identity, accessing each one can be a time-consuming and frustrating experiencing. In addition, although most local identities are configured independently (and fragmented across the Internet), it might be useful to connect the information. For example, a user's local identity with a bank could be securely connected to the same user's local identity with a utility company for easy, online payments. Federation addresses this idea of linking disparate service provider user accounts.

The Concept of Federation

Federation establishes a standards-based method of sharing and managing identity data and establishing single sign-on across security domains and organizations. As a concept, federation encompasses both *identity federation* and *provider federation*.

Identity Federation

Identity federation, as it has evolved with regard to online activity, begins with the notion of local identity. (See "The Concept of Identity" on page 124.) Sending and receiving email, checking bank balances, finalizing travel arrangements, accessing utility accounts, and shopping are just a few online services for which a user might define a local identity. If a user accesses all of these services, many different local identities have been configured. This virtual phenomenon offers an opportunity to fashion a system for users to *federate* these local identities.

Identity federation allows the user to link, connect, or bind the local identities that have been created for each *service provider* (a networked entity that provides one or more services to other entities). The linked local identities, referred to as a *federated identity*, allow the user to log in to one service provider site and click through to an affiliated service provider without having to reauthenticate or reestablish identity; in effect, single sign-on.

Provider Federation

Provider federation begins with a circle of trust. A *circle of trust* is a group of service providers who contractually agree to exchange authentication information. Each circle of trust must include at least one *identity provider*, a service provider that maintains and manages identity data, and provides authentication services. After the contracts and policies defining a circle of trust are in place, the specific protocols, profiles, endpoints, and security mechanisms being used in the deployment are collected into a metadata document that is exchanged among the members of the circle. Federated Access Manager provides the tools necessary to integrate the metadata and enable a circle of trust technologically. Authentication within this federation is honored by all membered providers.

Note – The establishment of contractual trust agreements between providers is beyond the scope of this guide. See "The Concept of Trust" on page 125 for an overview.

The Concept of Trust

Federating identities assumes existing trust relationships between participants. This trust is usually defined through business arrangements or contracts that describe the technical, operational, and legal responsibilities of each party and the consequences for not completing them. When defined, a trust relationship allows one organization to trust the user authentication and authorization decisions of another organization. This trust then enables a user to log in to one site and, if desired, access a trusted site without reauthentication.

Ensure that trust agreements are in force before configuring circles of trust with Federated Access Manager and going live. The Liberty Alliance Project has created a support document for helping to establish these trust arrangements. The *Liberty Trust Model Guidelines* document is located on the Support Documents and Utility Schema Files page of the Liberty Alliance Project web site.

How Federation Works

The goal of federation is to enable individuals and service providers to easily conduct network transactions across secure domains while protecting identity data. When organizations form a trust agreement, they agree to exchange user authentication information using specific web technologies. The trust agreement would be among multiple service providers that offer web-based services to users and, at least, one *identity provider* (a service provider that maintains and manages identity information). Once metadata is exchanged and the trust is established, single sign-on can be enabled between all the included providers, and users (in some federation protocol) may opt to federate their multiple identities. In Federated Access Manager, the trust agreement is virtually configured as a *circle of trust* using the console or command line interface. A circle of trust contains *entity providers* (service providers or identity providers) that are grouped together for the purpose of offering identity federation. *Identity federation* occurs when a user chooses to unite distinct service provider and identity provider accounts while retaining the individual account information with each provider. The user establishes a link that allows the exchange of authentication information between provider accounts. Users can choose to federate any or all identities they might have. After identity federation, when a user successfully authenticates to one of the service providers, access to any of the federated accounts within the circle of trust is allowed *without having to reauthenticate*. The following figure shows the subjects involved in federation.

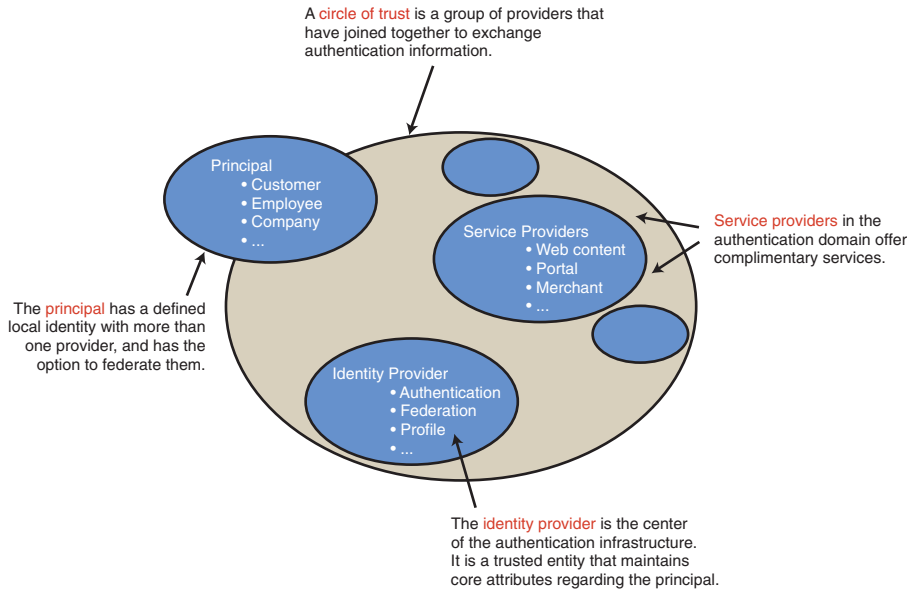


FIGURE 9–1 Subjects Involved in an Identity Federation

- A *principal* can have a defined local identity with more than one provider, and it has the option to federate the local identities. The principal might be an individual user, a group of individuals, a corporation, or a component of the Liberty architecture.
- A *service provider* is a commercial or not-for-profit organization that offers a web-based service such as a news portal, a financial repository, or retail outlet.
- An *identity provider* is a service provider that stores identity profiles and offers incentives to other service providers for the prerogative of federating their user identities. Identity providers might also offer services above and beyond those related to identity profile storage.
- To support identity federation, all service providers and identity providers must join together into a *circle of trust*. A circle of trust must contain at least one identity provider and at least two service providers. (One organization may be both an identity provider and a service provider.) Providers in a circle of trust must first write trust agreements to define their relationships. A *trust agreement* is a contract between organizations that defines how the circle will work. For more information, see "The Concept of Trust" on page 125.

A travel portal is a good example of a circle of trust. Typically, a travel portal is a web site designed to help you access various travel-related services from one location. The travel portal forms a partnership with each service provider displayed on its web site. (This might include hotels, airlines, and car rental agencies.) The user registers with the travel portal which, in effect, is the identity provider for the circle of trust. After logging in, the user might click through to an airline service provider to look for a flight. After booking a flight, the user might click through

to an accommodations service provider to look for a hotel. Because of the trust agreements previously established, the travel portal shares authentication information with the airline service provider, and the airline service provider with the accommodations service provider. The user moves from the hotel reservations web site to the airline reservations web site without having to reauthenticate. All of this is transparent to the user who must, depending on the underlying federation protocol, choose to federate any or all local identities. The following figure illustrates the travel portal example.

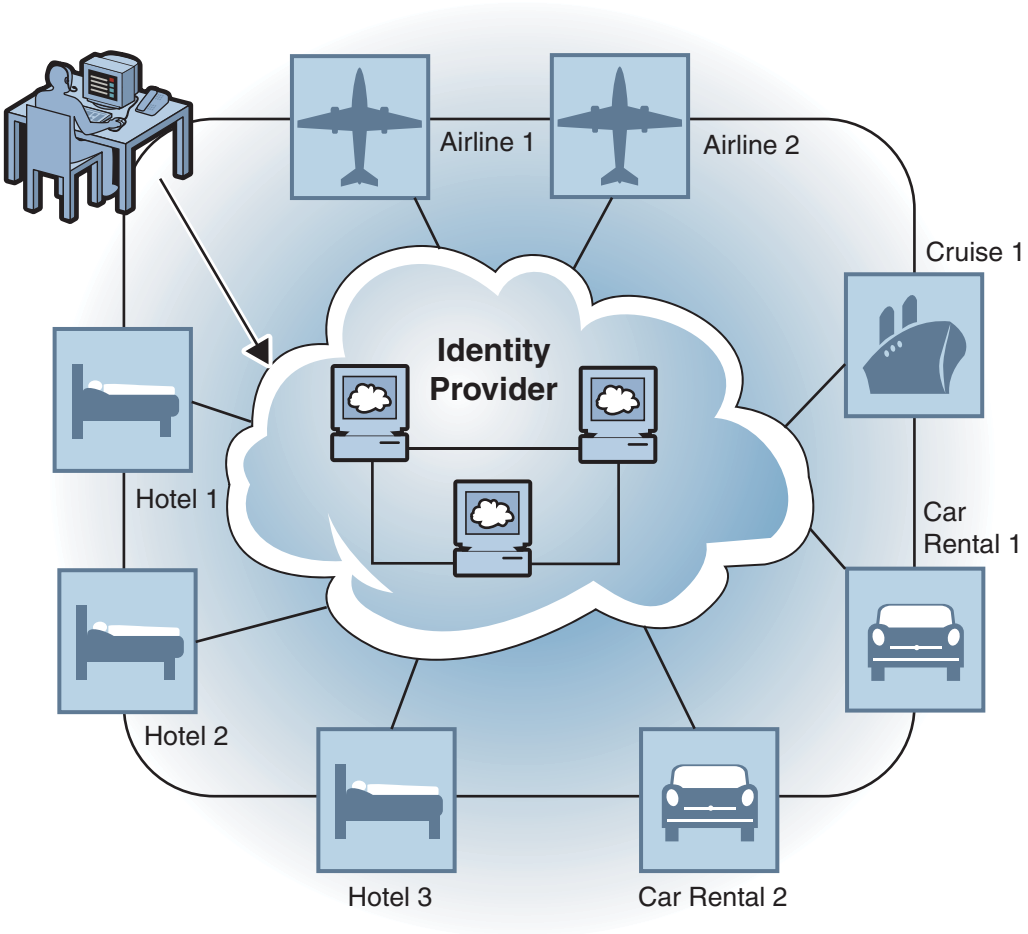


FIGURE 9–2 Federation Within a Travel Portal

Choosing a Federation Option

Federation is used to solve the problem of cooperation across heterogeneous, autonomous environments. In the beginning, federation meant using the Liberty Alliance Project Identity Federation Framework (Liberty ID-FF). Since then, other specifications have been developed with which federation can be accomplished including the Security Assertion Markup Language (SAML) and WS-Federation. To get started, use SAML v2 whenever possible as it supersedes both the Liberty ID-FF and SAML v1.x specifications. If you are integrating Federated Access Manager with Microsoft Active Directory with Federation Services, you **must** use WS-Federation. The Liberty ID-FF and SAML v1.x should only be used when integrating with a partner that is not able to use SAML v2.

Note – Federated Access Manager has appropriated the terms from the Liberty ID-FF for all federation protocol implementations in the Federated Access Manager console.

More information on these options can be found in the following sections:

- "Using SAML" on page 129
- "Using the Liberty ID-FF" on page 138
- "Using WS-Federation" on page 147

Using SAML

SAML defines an XML-based framework for exchanging identity information across security domains for purposes of authentication, authorization and single sign-on. It was designed to be used within other specifications (the Liberty Alliance Project, the Shibboleth project, and the Organization for the Advancement of Structured Information Standards have all adopted aspects of SAML) although the latest release (SAML v2) has incorporated elements from the specifications developed by those very same organizations. The SAML specifications consist of a number of components, illustrated by the following figure.

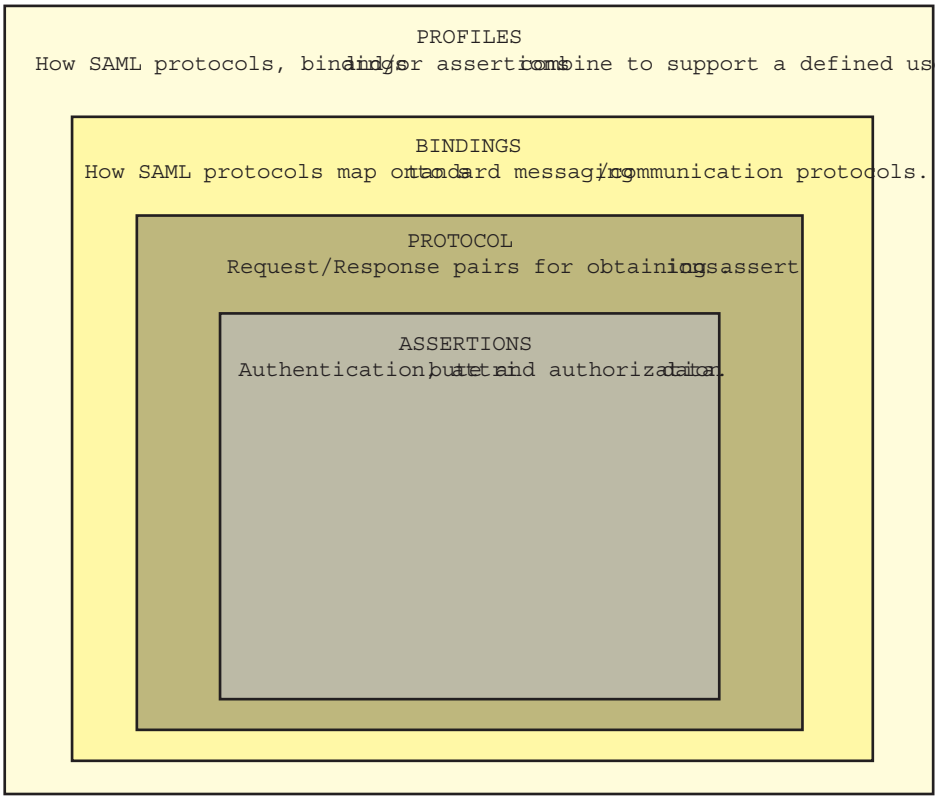


FIGURE 9–3 Components of the SAML Specifications

The SAML specification defines the *assertion* security token format as well as *profiles* that standardize the HTTP exchanges required to transfer XML requests and responses between an asserting authority and a trusted partner. An *assertion* is a package of verified security information that supplies one or more statements concerning a principal's authentication status, access authorization decisions, or identity attributes. (A person identified by an email address is a principal as might be a printer.) Assertions are issued by an asserting authority (a platform or application that declares whether a subject has been authenticated into its system), and received by relying parties (partner sites defined by the authority as *trusted*). Asserting authorities use different sources to configure assertion information, including external data stores or assertions that have already been received and verified.

The most recent SAML v2 specifications are defined more broadly than those developed for SAML v1.x — with particular attention paid to functionality dealing with federation. Before SAML v2 was introduced, SAML v1.x was simply a way to exchange identity data. In fact, up to version 1.1, the Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) was developed using the SAML 1.0 specification. Liberty ID-FF version 1.2 development was continued using the SAML v1.1 specification. But, following the release of version 1.2, the

Liberty ID-FF was incorporated into the SAML v2 specification. Additionally, SAML v2 adds components of the Shibboleth initiative. Thus, SAML v2 is a major revision, providing significant additional functionality and making the previous versions of SAML incompatible with it. Going forward, SAML v2 will be the basis on which Federated Access Manager implements federation. The following diagram illustrates the convergence.

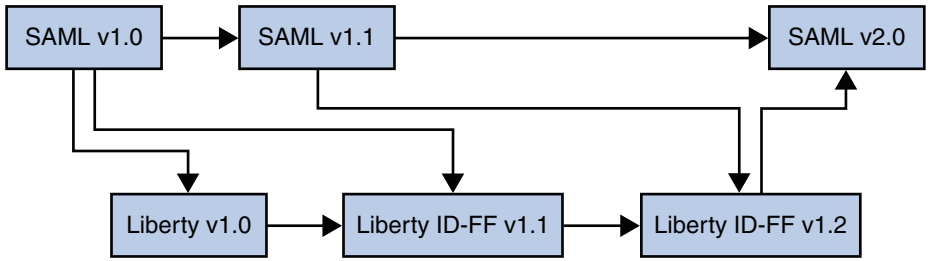


FIGURE 9–4 Liberty ID-FF and SAML Convergence

Note – For more information on this convergence (including how the Shibboleth Project was also integrated), see the Federation section of Strategic Initiatives on the Liberty Alliance Project web site.

In Federated Access Manager, you can federate identities using the Liberty ID-FF (as discussed in "Using the Liberty ID-FF" on page 138) or SAML v1.x or SAML v2 (as discussed in the following sections).

- "About SAML v2" on page 131
- "About SAML v1.x" on page 135

Caution – SAML v1.x and SAML v2 assertions and protocol messages are incompatible.

About SAML v2

Federated Access Manager delivers a solution that allows businesses to establish a framework for sharing trusted information across a distributed network of partners using the standards-based SAML v2. Towards this end, HTTP(S)-based service endpoints and SOAP service endpoints are supplied as well as assertion and protocol object manipulating classes. A web browser can access all HTTP(S)-based service endpoints and an application can make use of the SOAP endpoints and API as long as metadata for each participating business on BOTH sides of the SAML v2 interaction is exchanged beforehand. The key features of SAML v2 in Federated Access Manager include:

- Single sign-on using the POST profile, the Artifact binding (also referred to as HTTP redirect), and unsolicited responses (initiated by the identity provider)
- Single logout using HTTP redirect and SOAP binding
- Federation termination using HTTP redirect and SOAP binding
- Auto-federation (automatic linking of service provider and identity provider user accounts based on a common attribute)
- Bulk federation
- Dynamic creation of user accounts
- One time federation (transient NameID format for SSO)
- Basic Authentication, SSL and SSL with client authentication for SOAP binding security
- SAML v2 authentication
- Identity provider discovery
- XML verification, signing, encryption and decryption
- Profile initiation and processing using included JavaServer Pages™ (JSP™)
- Load balancing support
- Protocol coexistence with the SAML v1.x and the Liberty ID-FF

The following figure illustrates the SAML v2 framework which consists of web-based services [using SOAP, XML over HTTP(S) or HTML over HTTP(S)], and Java™-based application provider interfaces (API) and service provider interfaces (SPI). Additionally, the figure shows an agent embedded into a web container in which a service provider application is deployed. This agent enables the service provider to participate in the SAML v1.x or Liberty ID-FF protocols.

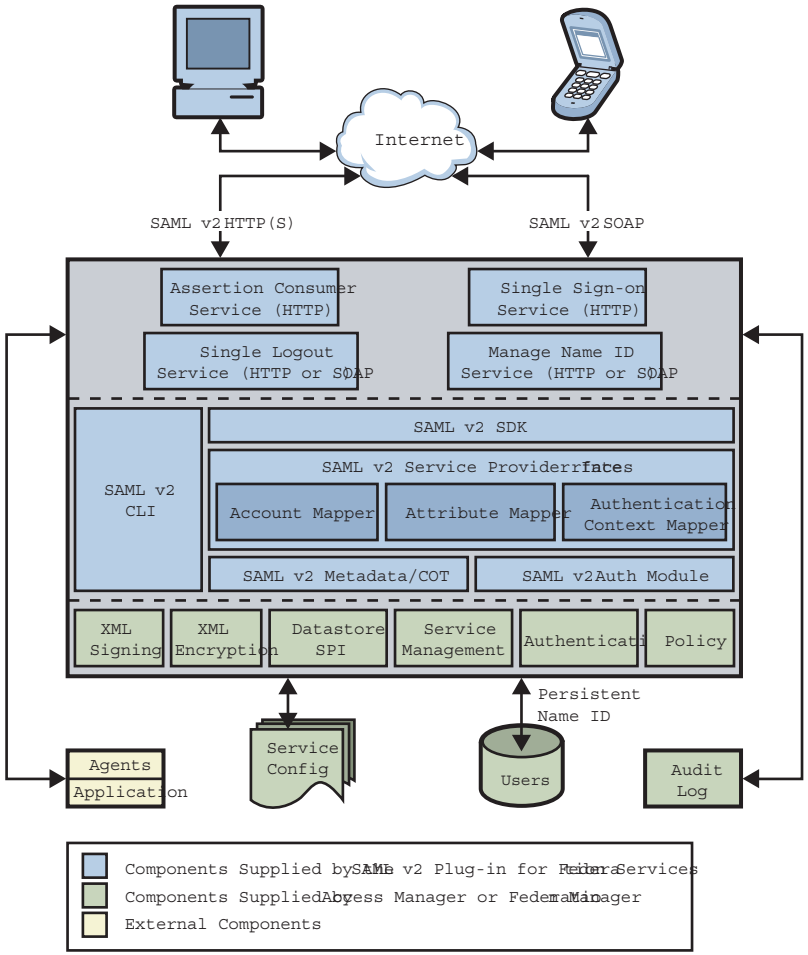


FIGURE 9–5 SAML v2 Architecture

More information about the SAML v2 framework can be found in the following sections:

- "Administration" on page 134
- "Application Programming Interfaces" on page 134
- "Service Provider Interfaces" on page 134
- "JavaServer Pages" on page 135

Administration

In order to communicate using the SAML v2 profiles you need, at least, two instances of Federated Access Manager. One instance will act for the identity provider and the other will act for the service provider. Name identifiers are used to communicate with each other regarding a user.

Note – SAML v2 single sign-on interactions support both *persistent* and *transient* identifiers. A persistent identifier is saved to a particular user entry as the value of two attributes. A transient identifier is temporary and no data will be written to the user's data store entry.

To prepare your instances for SAML v2 interactions, you need to exchange configuration information or *metadata* between all participating identity and service providers, and assemble the providers into a *circle of trust*. Utility APIs can then be used to communicate with the data store, reading, writing, and managing the relevant properties and property values. For more information see the XXXXXX.

Application Programming Interfaces

The SAML v2 framework contains API that can be used to construct and process assertions, requests, and responses. The SAML v2 Java API packages include:

- The `com.sun.identity.saml2.assertion` package provides interfaces to construct and process SAML v2 assertions. It also contains the `AssertionFactory`, a factory class used to obtain instances of the objects defined in the assertion schema.
- The `com.sun.identity.saml2.common` package provides interfaces and classes used to define common SAML v2 utilities and constants.
- The `com.sun.identity.saml2.protocol` package provides interfaces used to construct and process the SAML v2 requests and responses. It also contains the `ProtocolFactory`, a factory class used to obtain object instances for concrete elements in the protocol schema.

More information can be found in "Using the SAML v2 SDK" in *Sun Java System Federated Access Manager 8.0 Developer's Guide* and the *Federated Access Manager 8.0 Java API Reference*.

Service Provider Interfaces

The `com.sun.identity.saml2.plugins` package provides pluggable interfaces to implement SAML v2 functionality into your application. Default implementations are provided, but a customized implementation can be plugged in by modifying the corresponding attribute in the entity provider's extended metadata configuration file. The interfaces include mappers for:

- Account mapping (map between the account referred to in the incoming request and the local user account)

- Attribute mapping (specifies which set of user attributes in an identity provider user account needs to be included in an assertion, and maps the included attributes to attributes in the user account defined by the service provider)
- Authentication context mapping (map between Authentication Contexts defined in the SAML v2 specifications and authentication framework schemes defined in Federated Access Manager (user/module/service/role/level based authentication)

More information can be found in "Service Provider Interfaces" in *Sun Java System Federated Access Manager 8.0 Developer's Guide* and the *Federated Access Manager 8.0 Java API Reference*.

JavaServer Pages

The SAML v2 framework provides JSP that can be used to initiate single sign-on, single logout and termination requests from either the identity provider or the service provider using a web browser. The JSP accept query parameters to allow flexibility in constructing SAML v2 requests; they can be modified for your deployment. More information can be found in "JavaServer Pages" in *Sun Java System Federated Access Manager 8.0 Developer's Guide*.

About SAML v1.x

Federated Access Manager can be configured to use SAML v1.x to achieve interoperability between vendor platforms that provide SAML v1.x assertions. The SAML v1.x protocol defines the *assertion* security token format as well as *profiles* that standardize the HTTP exchanges required to transfer XML requests and responses between a SAML v1.x authority and a trusted partner site. An *assertion* is a package of verified security information that supplies one or more statements concerning a subject's authentication status, access authorization decisions, or identity attributes. (A person identified by an email address is a subject as might be a printer.) Assertions are issued by a SAML v1.x asserting authority (a platform or application that declares whether a subject has been authenticated into its system), and received by relying parties (partner sites defined by the authority as *trusted*). SAML v1.x authorities use different sources to configure the assertion information, including external data stores or assertions that have already been received and verified. SAML v1.x can be used to allow Federated Access Manager to:

- Authenticate users and access trusted partner sites without having to reauthenticate; in effect, single sign-on. This single sign-on process independent of the process enabled by Access Manager user session management.
- Act 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. For example, employees of an organization can be allowed to order office supplies from suppliers if they are authorized to do so.
- Act as both an attribute authority that allows trusted partner sites to query a subject's attributes, and an authentication authority that allows trusted partner sites to query a subject's authentication information.

- Validate parties in different security domains for the purpose of performing business transactions.
- Build Authentication, Authorization Decision, and Attribute Assertions using the SAML v1.x API.
- Permit an XML-based digital signature signing and verifying functionality to be plugged in.

Note – Although Liberty ID-FF (as described in About the Identity Federation Framework) integrates aspects of the SAML v1.x specifications, its usage of SAML v1.x is independent of the SAML v1.x framework as described in this section.

The following figure illustrates how SAML v1.x interacts with the other components in Federated Access Manager.

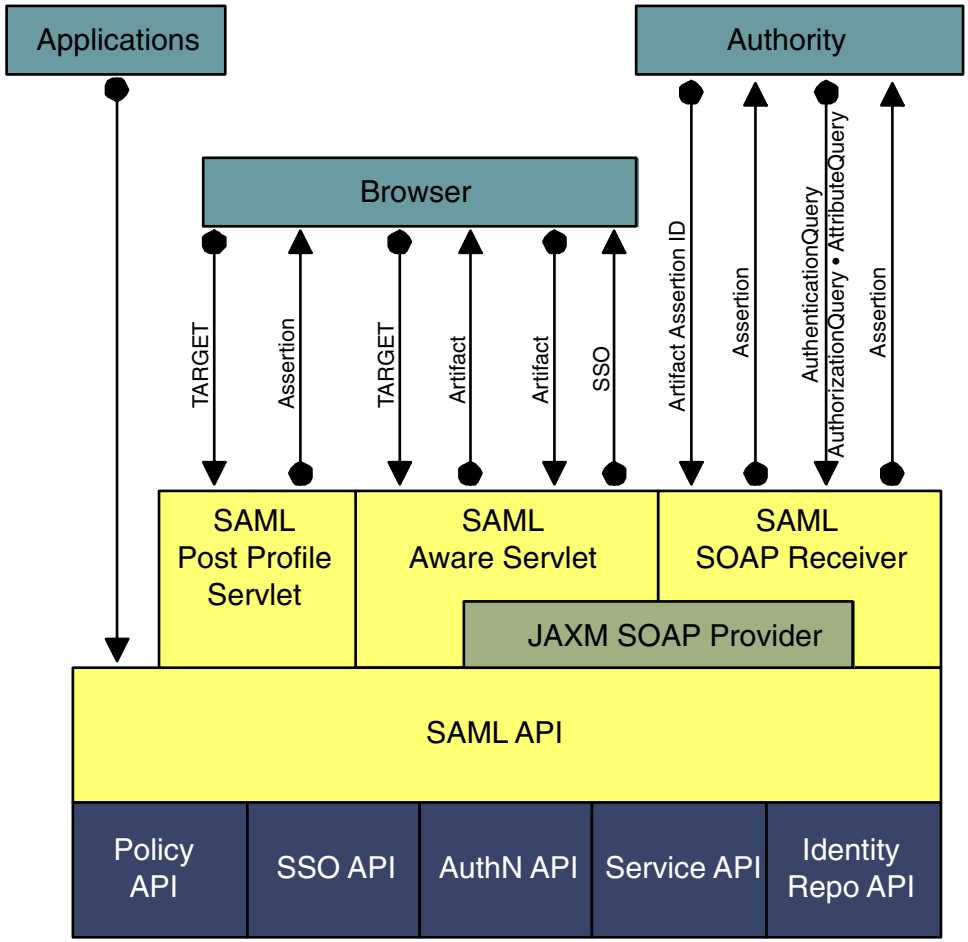


FIGURE 9–6 SAML v1.x Interaction in Federated Access Manager

Comparison of SAML v1.x and the Liberty Alliance Project Specifications

SAML v1.x was designed to address the issue of cross-domain single sign-on. The Liberty Alliance Project was formed to develop technical specifications that would solve business process problems. These issues include single sign-on, but also incorporate protocols for account linking and consent, among others. SAML v1.x, on the other hand, does not solve issues such as privacy, single logout, and federation termination.

The SAML v1.x specifications and the Liberty Alliance Project specifications do not compete with one another. They are complementary. In fact, the Liberty Alliance Project specifications

leverage profiles from the SAML specifications. The decision of whether to use SAML v1.x or the Liberty specifications depends on your goal. In general, SAML v1.x should suffice for single sign-on basics. The Liberty Alliance Project specifications can be used for more sophisticated functions and capabilities, such as global sign-out, attribute sharing, web services. The following table compares the benefits of the two.

TABLE 9–1 Comparison of the SAML v1.x and Liberty Alliance Project Specifications

SAML v1.x Uses	Liberty Alliance Project Uses
Cross-domain single sign-on	Single sign-on *only* after user federation
No user federation	User federation
No privacy control, best for use within one company	Built on top of SAML
User identifier is sent in plain text	User identifier is sent as a unique handle

Using the Liberty ID-FF

The Liberty Alliance Project was formed to develop technical specifications that would solve business process issues including single sign-on, federation and consent, among others. The Liberty Alliance Project Identity Federation Framework (Liberty ID-FF) uses a name identifier to pass identity data between identity providers and service providers. The *name identifier* is a randomly generated character string that is assigned to a principal and used to federate the principal's accounts at the identity provider and service provider sites. This pseudonym allows all providers to identify a principal without knowing the user's actual identity. The name identifier has meaning only in the context of the relationship between providers. SAML v1.x is used for provider interaction.

Note – Liberty ID-FF was initially defined as an extension of SAML 1.0 (and later SAML 1.1). The extensions have now been contributed back into SAML v2 which, going forward, will be the basis on which the Liberty Alliance Project builds additional federated identity applications. See "Using SAML" on page 129 for more information on this convergence.

The following sections contain information about the Liberty ID-FF and the features implemented in Federated Access Manager.

About the Liberty ID-FF Process

The Liberty ID-FF is designed to work with heterogeneous platforms, various networking devices (including personal computers, mobile phones, and personal digital assistants), and emerging technologies. The process of Liberty ID-FF federation begins with authentication. A user attempting to access a resource protected by Federated Access Manager are redirected to the proprietary Authentication Service via an Federated Access Manager login page. After the user provides credentials, the Authentication Service allows or denies access to the resource based on the outcome.

Note – For more information about the proprietary Authentication Service, see the Chapter 6, "Understanding the Authentication Service."

When the user attempts access to a resource that is a trusted member provider of a configured circle of trust using the Liberty ID-FF, the process of user authentication begins with the search for a valid Federated Access Manager session token from the proprietary Authentication Service. The process can go in one of two directions based on whether a session token is found.

- If no session token is found, the principal is redirected to a location defined by the pre-login URL to establish a valid session. See "Pre-login Process" on page 141 for details.
- If a session token is found, the principal is granted (or denied) access to the requested page. Assuming access is granted, the requested page contains a link so the principal may federate the Federated Access Manager identity with the identity local to the requested site. If the principal clicks this link, federation begins. See "Liberty ID-FF Federation and Single Sign-On" on page 141 for details.

The following figure illustrates these divergent paths. The process shown is the default process when no application has been deployed. When an application is deployed and using Federated Access Manager, the process will change based on the query parameters and preferences passed to Federated Access Manager from the participating application. For more information, see XXXXXX The Pre-login URL.

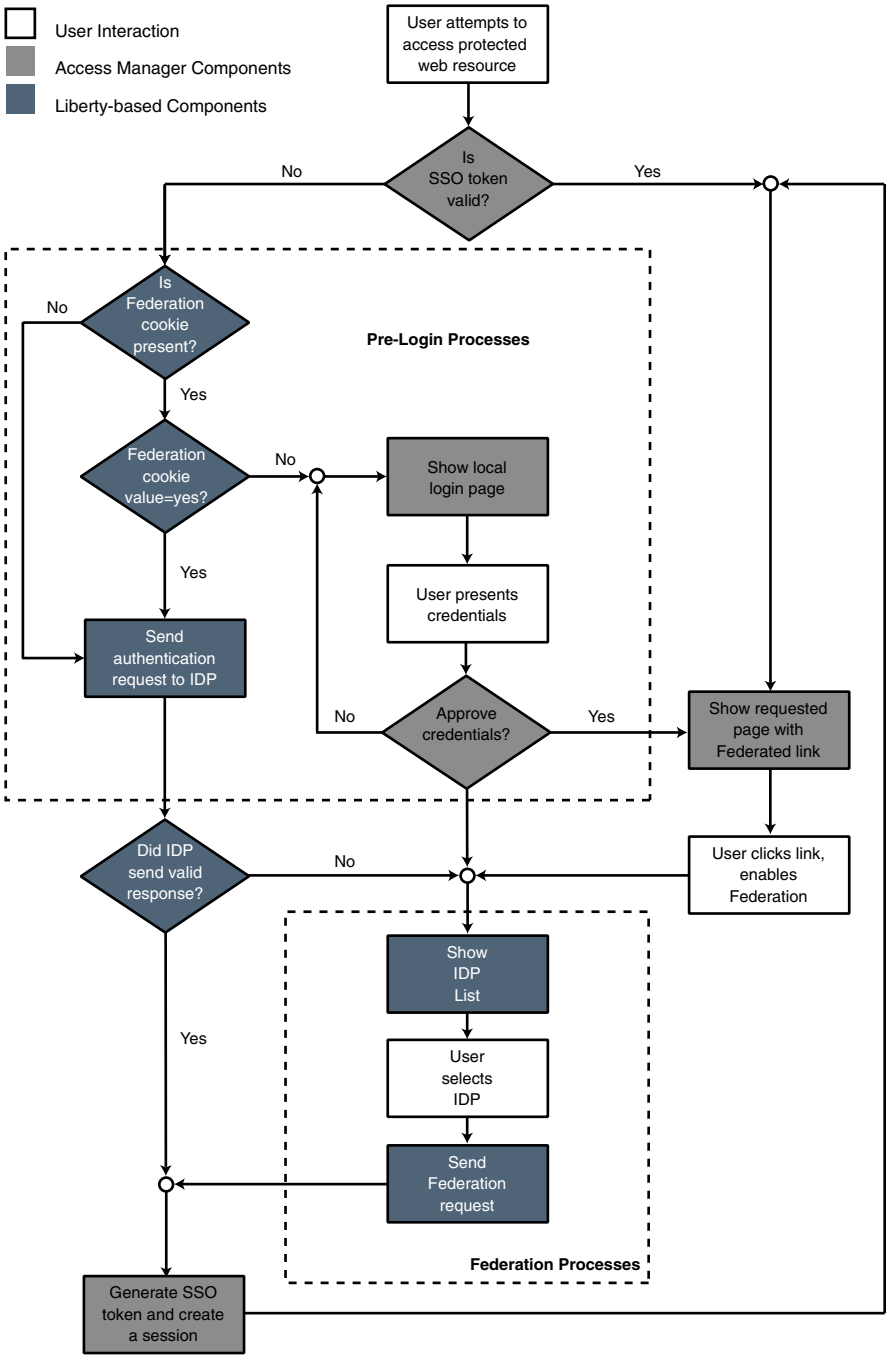


FIGURE 9–7 Default Process of Federation

Pre-login Process

The pre-login process establishes a valid Federated Access Manager session. When a principal attempts to access a service provider site and no Federated Access Manager session token is found, Federated Access Manager searches for a federation cookie. A *federation cookie* is implemented by Federated Access Manager and is called `fedCookie`. It can have a value of either `yes` or `no`, based on the principal's federation status.

Note – A federation cookie is *not* defined in the Liberty Alliance Project specifications.

At this point, the pre-login process may take one of the following paths:

- If a federation cookie is found and its value is `no`, a Federated Access Manager login page is displayed and the principal submits credentials to the proprietary Authentication Service. When authenticated by Federated Access Manager, the principal is redirected to the requested page, which might contain a link to allow for identity federation. If the principal clicks this link, federation begins. See "Liberty ID-FF Federation and Single Sign-On" on page 141 for details.
- If a federation cookie is found and its value is `yes`, the principal has already federated an identity but has not been authenticated by an identity provider within the circle of trust for this Federated Access Manager session. Authentication to Federated Access Manager is achieved on the back end by sending a request to the principal's identity provider. After authentication, the principal is directed back to the requested page.
- If no federation cookie is found, a *passive* authentication request (one that does not allow identity provider interaction with the principal) is sent to the principal's identity provider. If an affirmative authentication is received back from the identity provider, the principal is directed to the Federated Access Manager Authentication Service, where a session token is granted. The principal is then redirected to the requested page. If the response from the identity provider is negative (for example, if the session has timed out), the principal is sent to a common login page to complete either a local login or Liberty ID-FF federation. See "Liberty ID-FF Federation and Single Sign-On" on page 141 for details.

Note – This pre-login process is the default behavior of Federated Access Manager. This process might change based on parameters passed to Federated Access Manager from the participating application. For more details, see the section on XXXXXX The Pre-login URL.

Liberty ID-FF Federation and Single Sign-On

When a principal logs in to access a protected resource or service, Federated Access Manager sends a request to the appropriate identity provider for authentication confirmation. If the identity provider sends a positive response, the principal gains access to all provider sites within the circle of trust. If the identity provider sends a negative response, the principal is directed to authenticate again using the Liberty ID-FF process.

In the Liberty ID-FF process, a principal selects an identity provider and sends credentials for authentication. After authentication is complete and access is granted, the principal is issued a session token from the Federated Access Manager Authentication Service and redirected to the requested page. As long as the session token remains valid, the principal can access other service providers in the authentication domain without having to authenticate again.

Note – The Common Domain for Identity Provider Discovery is used by a service provider to determine the identity provider used by a principal in a circle of trust that contains multiple identity providers. See "Creating a Common Domain for Identity Provider Discovery" on page 149 for details.

Liberty ID-FF Features

The following sections contain information about the Liberty ID-FF features implemented in Federated Access Manager.

Identity Federation and Single Sign-On

Let's assume that a principal has separate user accounts with a service provider and an identity provider in the same circle of trust. In order to gain access to these individual accounts, the principal would authenticate with each provider separately. If federating with the Liberty ID-FF though, after authenticating with the service provider, the principal may be given the option to federate the service provider account with the identity provider account. Consenting to the federation of these accounts links them for SSO, the means of passing a user's credentials between applications without the user having to reauthenticate. SSO and federated SSO have different processes. With Federated Access Manager, you can achieve SSO in the following ways:

- Install a policy agent in a web container to protect the application and pass the `HTTP_HEADER` and `REMOTE_USER` variables to the application to capture the user credentials. You may or may not need a custom authentication module.
- Customize the application's authentication module to create an `SSOToken` from the request object or from the SSO cookie. Afterwards, retrieve the user credentials using the SSO API and create a session data structure using the application's API.

To set up federated SSO, you must first establish SSO. Following that, enable federation in the metadata for the service provider entity and the identity provider entity using Federated Access Manager. Liberty ID-FF providers differentiate between federated users by defining a unique

identifier for each account. (They are not required to use the principal's actual provider account identifier.) Providers can also choose to create multiple identifiers for a particular principal. However, identity providers must create one handle per user for service providers that have multiple web sites so that the handle can be resolved across all of them.

Note – Because both the identity provider entity and the service provider entity in a federation need to remember the principal's identifier, they create entries that note the value in their respective user repositories. In most scenarios, the identity provider's identifier is conveyed to a service provider and not vica versa. For example, if a service provider does not maintain its own user repository, the identity provider's identifier is used.

Federated Access Manager can accommodate the following SSO and federation-related functions:

- Providers of either type notify the principal upon identity federation or defederation.
- Providers of either type notify each other regarding a principal's defederation.
- Identity providers notify the appropriate service providers regarding a principal's account termination.
- Providers of either type display a list of federated identities to the principal.
- Users can terminate federations or defederate identities.

Additionally, Federated Access Manager can accommodate the features explained in the following sections.

- "Auto-Federation" on page 143
- "Bulk Federation" on page 143

Auto-Federation

Auto federation will automatically federate a user's disparate provider accounts based on a common attribute. During SSO, if it is deemed a user at provider A and a user at provider B have the same value for the defined common attribute (for example, an email address), the two accounts will be federated without consent or interaction from the principal. For more information, see XXXXXXX Auto-Federation.

Bulk Federation

Federating one user's service provider account with their identity provider account generally requires the principal to visit both providers and link them. An organization though needs the ability to federate user accounts behind the scenes. Federated Access Manager provides a script for federating user accounts in bulk. The script allows the administrator to federate many (or all) of a principal's provider accounts based on metadata passed to the script. Bulk federation is useful when adding a new service provider to an enterprise so you can federate a group of existing employees to the new service. For more information, see XXXXXX Bulk Federation.

Authentication and Authentication Context

SSO is the means by which a provider of either type can convey to another provider that a principal has been authenticated. Authentication is the process of validating user credentials; for example, a user identifier accompanied by an associated password. You can authenticate users with Federated Access Manager in the following ways:

- Use a policy agent to insert HTTP header variables into the request object. This functions for web applications only.
- Use the authentication API to validate and retrieve user identity data. This will work with either web or non-web applications.

Identity providers use local (to the identity provider) session information mapped to a user agent as the basis for issuing Security Assertion Markup Language (SAML) authentication assertions to service providers. Thus, when the principal uses a user agent to interact with a service provider, the service provider requests authentication information from the identity provider based on the user agent's session information. If this information indicates that the user agent's session is presently active, the identity provider will return a positive authentication response to the service provider. Federated Access Manager allows providers to exchange the following minimum set of authentication information with regard to a principal.

- Authentication status (active or not)
- Instant (time authenticated)
- Authentication method
- Pseudonym (temporary or persistent)

SAML v1.x is used for provider interaction during authentication but not all SAML assertions are equal. Different authorities issue SAML assertions of different quality. Therefore, the Liberty ID-FF defines how the consumer of a SAML assertion can determine the amount of assurance to place in the assertion. This is referred to as the *authentication context*, information added to the SAML assertion that gives the assertion consumer the details they need to make an informed entitlement decision. For example, a principal uses a simple identifier and a self-chosen password to authenticate to a service provider. The identity provider sends an assertion to a second service provider that states how the principal was authenticated to the first service provider. By including the authentication context, the second service provider can place an appropriate level of assurance on the associated assertion. If the service provider were a bank, they might require stronger authentication than that which has been used and respond to the identity provider with a request to authenticate the user again using a more stringent context. The authentication context information sent in the assertion might include:

- The initial user identification mechanism (for example, face-to-face, online, or shared secret).
- The mechanisms for minimizing compromise of credentials (for example, private key in hardware, credential renewal frequency, or client-side key generation).
- The mechanisms for storing and protecting credentials (for example, smart card, or password rules).

- The authentication mechanisms (for example, password or smart card with PIN).

The Liberty ID-FF specifications define authentication context *classes* against which an identity provider can claim conformance. The Liberty ID-FF authentication contexts are listed and described in the following table.

TABLE 9–2 Authentication Context Classes

Class	Description
MobileContract	Identified when a mobile principal has an identity for which the identity provider has vouched.
MobileDigitalID	Identified by detailed and verified registration procedures, a user's consent to sign and authorize transactions, and DigitalID-based authentication.
MobileUnregistered	Identified when the real identity of a mobile principal has not been strongly verified.
Password	Identified when a principal authenticates to an identity provider by using a password over an unprotected HTTP session.
Password-ProtectedTransport	Identified when a principal authenticates to an identity provider by using a password over an SSL-protected session.
Previous-Session	Identified when an identity provider must authenticate a principal for a current authentication event and the principal has previously authenticated to the identity provider. This affirms to the service provider a time lapse from the principal's current resource access request. **Note –** The context for the previously authenticated session is not included in this class because the user has not authenticated during this session. Thus, the mechanism that the user employed to authenticate in a previous session should not be used as part of a decision on whether to now allow access to a resource.
Smartcard	Identified when a principal uses a smart card to authenticate to an identity provider.
Smartcard-PKI	Identified when a principal uses a smart card with an enclosed private key and a PIN to authenticate to an identity provider.
Software-PKI	Identified when a principal uses an X.509 certificate stored in software to authenticate to the identity provider over an SSL-protected session.

TABLE 9–2 Authentication Context Classes *(Continued)*

Class	Description
Time-Sync-Token	Identified when a principal authenticates through a time synchronization token.

For more information, see the *Liberty ID-FF Authentication Context Specification*. Additionally, there is an XML schema defined which the identity provider authority can use to incorporate the context of the authentication in the SAML assertions it issues.

Identifiers and Name Registration

Federated Access Manager supports name identifiers that are unique across all providers in a circle of trust. This identifier can be used to obtain information for or about the principal without requiring the user to consent to a long-term relationship with the service provider. When beginning federation, the identity provider generates an opaque value that serves as the initial name identifier that both the service provider and the identity provider use to refer to the principal when communicating with each other. After federation, the identity provider or the service provider may register a different opaque value. If a service provider registers a different opaque value for the principal, the identity provider must use the new identifier when communicating with the service provider about the principal. The reasons for changing an identifier would be implementation-specific. The initial name identifier defined by the identity provider is always used to refer to the principal unless a new name identifier is registered.

Global Logout

A principal may establish authenticated sessions with both an identity provider and individual service providers, based on authentication assertions supplied by the identity provider. When the principal logs out of a service provider session, the service provider sends a logout message to the identity provider that provided the authentication for that session. When this happen, or the principal manually logs out of a session at an identity provider, the identity provider sends a logout message to each service provider to which it provided authentication assertions under the relevant session. The one exception is the service provider that sent the logout request to the identity provider.

Dynamic Identity Provider Proxying

An identity provider can choose to proxy an authentication request to an identity provider in another authentication domain if it knows that the principal has been authenticated with this identity provider. The proxy behavior is defined by the local policy of the proxying identity provider. However, a service provider can override this behavior and choose not to proxy. This function can be implemented as a form of authentication when, for instance, a roaming mobile user accesses a service provider that is not part of the mobile home network. For more information see XXXXXX Dynamic Identity Provider Proxying.

Using WS-Federation

WS-Federation is part of the larger Web Services Security (WS-Security) framework which provides a means for applying security to web services through the use of security tokens. WS-Security describes how to attach signature and encryption headers as well as security tokens (including binary security tokens such as X.509 certificates and Kerberos tickets) to SOAP messages. WS-Trust, another specification in the WS-Security framework, provides for federation by defining a Security Token Service (STS) and a protocol for requesting and issuing the security tokens. WS-Federation, as implemented in Federated Access Manager, uses the Federated Access Manager Security Token Service (modelled on the WS-Trust specification) to allow providers in different security realms to broker trust using information on identities, identity attributes and authentication, and provider federation. A principal requests a token from the Security Token Services. This token, which may represent the principal's primary identity, a pseudonym, or the appropriate attributes, is presented to the service provider for authentication and authorization. WS-Federation uses several security tokens as well as the mechanism for associating them with messages. This release of Federated Access Manager has implemented the following features of the WS-Federation specification.

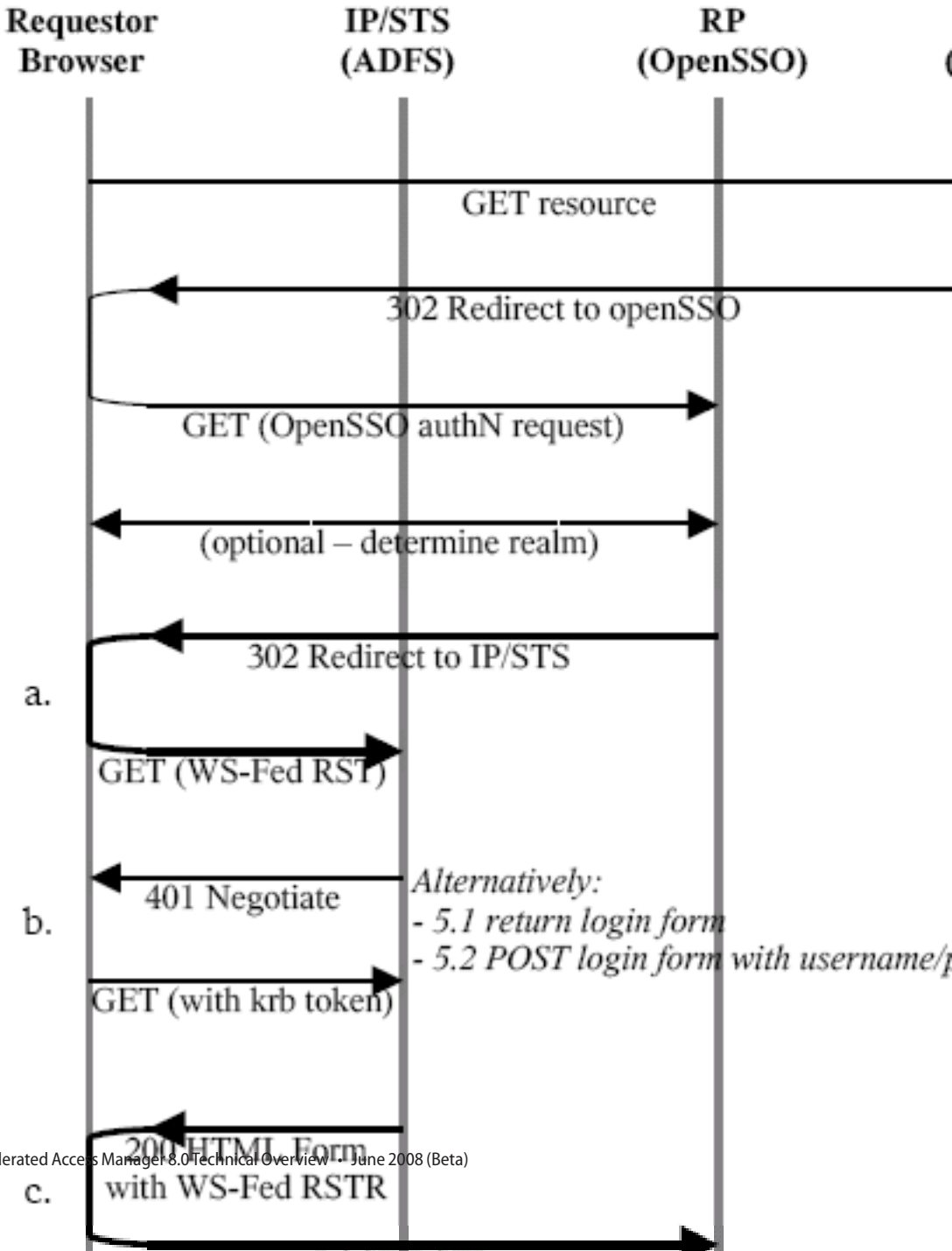
- The *Web (Passive) Profile* defines single sign-on, single logout, attribute and pseudonym token exchanges for passive requestors; for example, a web browser that supports HTTP. For the passive mechanisms to provide a single or reduced sign-on, there needs to be a service that will verify that the claimed requestor is really the requestor. Initial verification MUST occur in a secure environment; for example, using SSL/TLS or HTTP/S. The token is abstract and the token exchange is based on the Security Token Service model of WS-Trust.
- Tokens based on the *Web Services-Interoperability Basic Security Profile* (WS-I BSP) define security that is implemented inside a SOAP message; and security implemented at the transport layer, using HTTPS. The protocol covers how you generate or handle security tokens.

The WS-Federation implementation in Federated Access Manager is based on the application's SAML v2 code and uses WS-Federation 1.1 metadata. Authentication request parameters are represented directly as GET parameters, and the authentication response is a WS-Trust `RequestSecurityTokenResponse` element.

Note – There is no authentication context mapping, persistent or transient NameID identifiers or auto-federation in the Federated Access Manager implementation of WS-Federation.

The entry points for all WS-Federation functionality will be implemented as servlets. JavaServer Pages (JSP) are used only for HTML content (for example, the HTML form used to send the WS-Federation single response from the identity provider to the service provider). The following figure illustrates the flow of messages between Federated Access Manager (acting as the service provider) and the Active Directory (acting as the identity provider).

Single Sign-On – ADFS as IP/STS, OpenSSO as RP:



In a WS-Federation interaction, an unauthenticated user attempting to access a protected web site. The site redirects the user to the Active Directory for Federation Services (ADFS) identity provider. After the user is authenticated (either by a back-end single sign-on or by entering credentials), ADFS posts a form containing a signed SAML assertion to the service provider. The service provider validates the assertion, copies the attributes into the user's session, and gives the appropriate access.

Note – Microsoft Active Directory Federation Services supports single sign-on via WS-Federation.

Creating a Common Domain for Identity Provider Discovery

Service providers need a way to determine which identity provider is used by a principal requesting authentication. Because circles of trust are configured without regard to their location, this function must work across DNS-defined domains. A common domain is configured, and a common domain cookie written, for this purpose.

Let's suppose a circle of trust contains more than one identity provider. In this case, a service provider trusts more than one identity provider so, when a principal needs authentication, the service provider with which the principal is communicating must have the means to determine the correct identity provider. To ascertain a principal's identity provider, the service provider invokes a protocol exchange to retrieve the *common domain cookie*, a cookie written for the purpose of *introducing* the identity provider to the service provider. If no common domain cookie is found, the service provider will present a list of trusted identity providers from which the principal can choose. After successful authentication, the identity provider writes (using the configured Writer Service URL) a common domain cookie and, the next time the principal attempts to access a service, the service provider finds and reads the common domain cookie (using the configured Reader Service URL), to determine the identity provider. More information on the Common Domain for Identity Provider Discovery is available in the following sections:

- "The Common Domain" on page 149
- "The Common Domain Cookie" on page 150
- "The Writer Service and the Reader Service" on page 150

The Common Domain

The *common domain* is established for use only within the scope of identity provider discovery in a defined circle of trust. In Federated Access Manager deployments, the identity provider discovery WAR is deployed in a web container installed in a predetermined and pre-configured *common* domain so that the common domain cookie is accessible to all providers in the circle of trust. For example, if an identity provider is available at `http://www.Bank.com`, a service

provider is available at `http://www.Store.com`, and the defined common domain is `RetailGroup.com`, the addresses will be `Bank.RetailGroup.com` and `Store.RetailGroup.com`, respectively. If the HTTP server in the common domain is operated by the service provider, the service provider will redirect the user agent to the appropriate identity provider.

The Common Domain Cookie

After an identity provider authenticates a principal, the identity provider sets a URL-encoded cookie defined in a predetermined domain common to all identity providers and service providers in the circle of trust. The *common domain cookie* is named `_liberty_idp` for Liberty ID-FF and `_saml_idp` for SAML v2. After successful authentication, a principal's identity provider appends their particular encoded identifier to a list in the cookie. If their identifier is already present in the list, the identity provider may remove the initial appearance and append it again. The intent is that the service provider reads the last identifier on the cookie's list to find the principal's most recently established identity provider.

Note – The identifiers in the common domain cookie are a list of `SuccinctID` elements encoded in the Base64 format. One element maps to each identity provider in the circle of trust. Service providers then use this `SuccinctID` element to find the user's preferred identity provider.

The Writer Service and the Reader Service

After a principal authenticates with a particular identity provider, the identity provider redirects the principal's browser to the configured Writer Service URL using a parameter that indicates they are the identity provider for this principal. The Writer Service then writes a cookie using the parameter. Thereafter, all providers configured in this common domain will be able to tell which identity provider is used by this principal. Thus, the next time the principal attempts to access a service hosted by a service provider in the same common domain, the service provider retrieves and reads the common domain cookie, using the configured Reader Service URL, to determine the identity provider.

The Writer Service URL and the Reader Service URL can be defined for use with the Liberty ID-FF or the SAML v2 federation protocol. The URLs are defined when you create a circle of trust for federation. The Common Domain for Identity Provider Discovery for Liberty ID-FF is based on the Identity Provider Introduction Profile detailed in the *Liberty ID-FF Bindings and Profiles Specifications*. The Common Domain for Identity Provider Discovery for SAML v2 is an implementation of the Identity Provider Discovery Profile as described in the Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0 specification.

PART IV

Web Services and Web Services Security in Federated Access Manager

This fourth part of the Sun Federated Access Manager Technical Overview contains information on implemented web services and web services security. It contains the following chapters:

- Chapter 10, "Accessing and Securing Web Services"

◆ ◆ ◆ CHAPTER 10

Accessing and Securing Web Services

In Federated Access Manager, the Federation framework enables the secure exchange of authentication and authorization information by providing an interface for creating, modifying, and deleting circles of trust and configuring service providers and identity providers (both remote and hosted types) as entity providers. Additionally, implemented web services define a stack that supports the Federation framework. This chapter contains the following sections:

Web Services Architecture

The following figure illustrates the architecture of the web services stack and how a web service client (WSC) communicates with the web service provider (WSP) which, in this case, is Federated Access Manager.

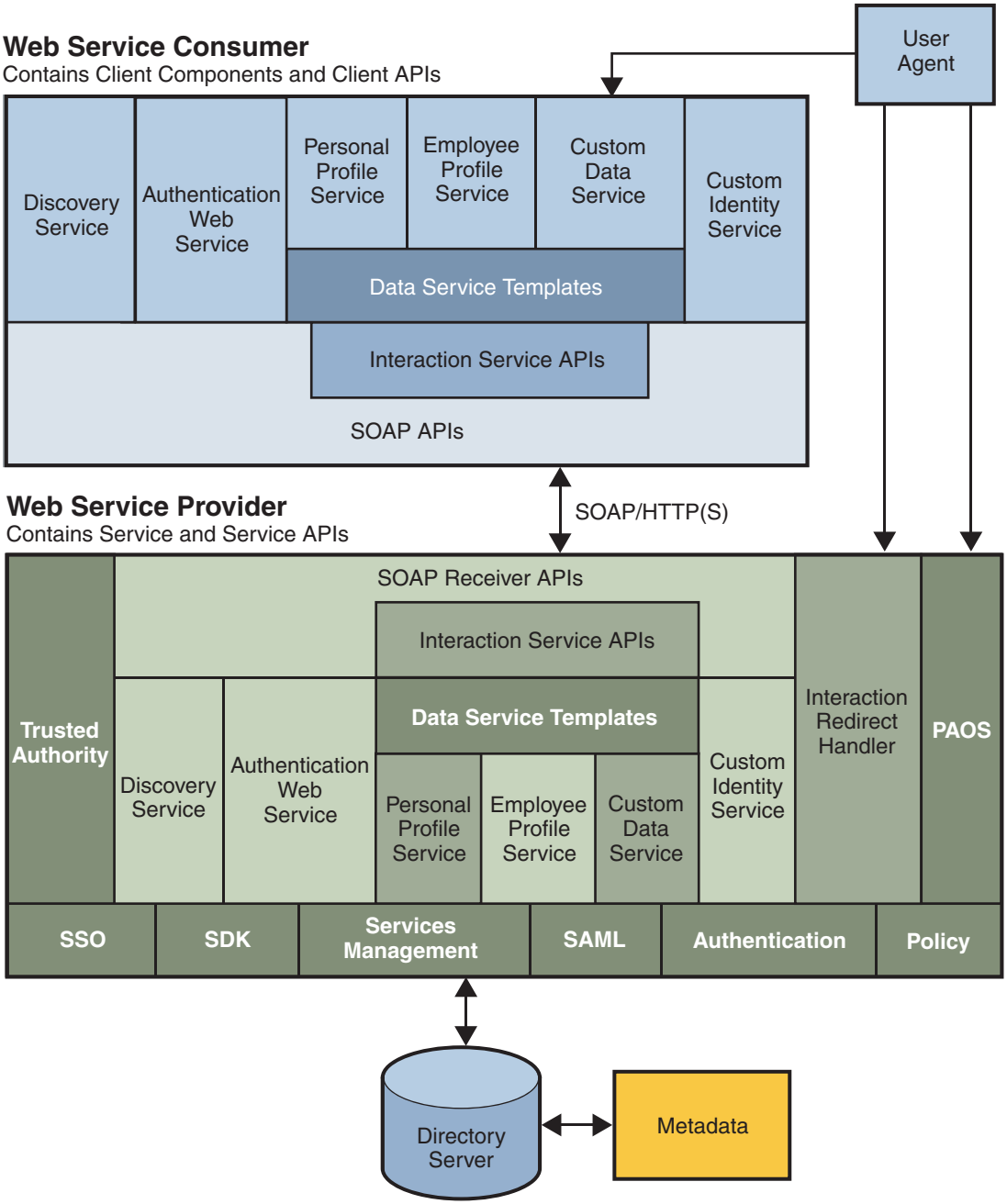


FIGURE 10–1 Web Services Architecture

Implemented Services

Federated Access Manager includes the following web services:

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.

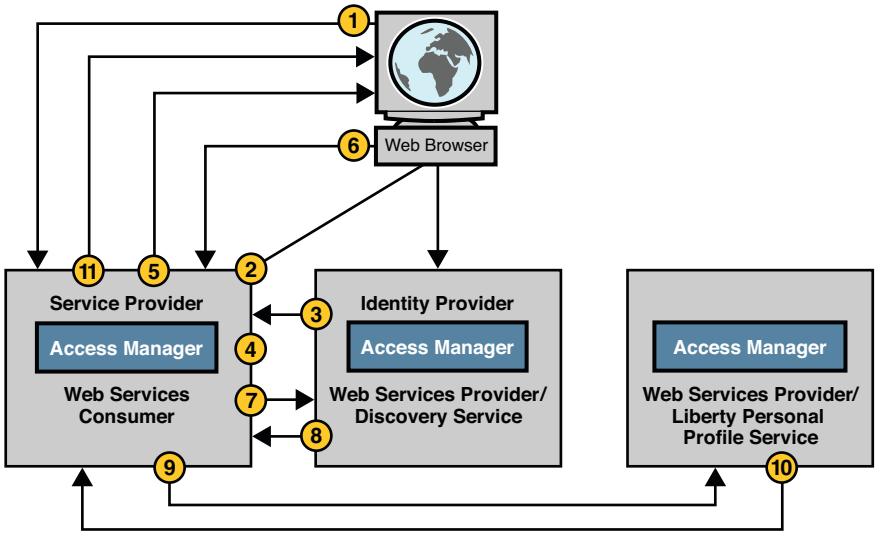


FIGURE 10–2 Web Services Stack Process

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.
7. 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.
8. 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.
9. 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.

10. 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.

11. 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 XXXXXX Sun Java System Access Manager 7.1 Federation and SAML Administration Guide.

PART V

Additional Features

This final section of the Sun Federated Access Manager Technical Overview contains information on the Logging Service and third-party product information. It contains the following chapters:

- Chapter 11, "Logging and the Java Enterprise System Monitoring Framework"
- Chapter 12, "Third-Party Product Integration"

◆ ◆ ◆ CHAPTER 11

Logging and the Java Enterprise System Monitoring Framework

Sun Federated Access Manager provides its own logging feature that records information such as user login, user logout, session creation, and policy evaluation. This chapter describes how Access Manager logging works, and provides some information about the Java Enterprise System Monitoring Framework. It contains the following sections:

Caution – See common criteria logging paper by Burt Fujii

Logging Overview

The Logging Service enables Access Manager services to record information such as access denials, access approvals, authentication events, and authorization violations. Administrators can use the logs to track user actions, analyze traffic patterns, audit system usage, review authorization violations, and troubleshoot. The logged information from all Access Manager services is recorded in one centralized directory. The default location for all Access Manager log files is `/var/opt/SUNWam/logs`. Logging client APIs enable external applications to access the Logging framework. This section contains the following:

Logging Service

The Logging Service stores the attributes and values for the logging function. A global service configuration file named `amLogging.xml` defines the Logging attributes. Examples of Logging Service attributes are maximum log size, log location, and log format (flat file or relational database). The attribute values are applied across the Access Manager deployment and inherited by every configured realm. By default, `amLogging.xml` is located in the directory `/etc/opt/SUNWam/config/xml` when Access Manager is installed in a Solaris environment. (When installed on Windows, the directory is *jes-install-dir*`\identity\config\xml`; on HP-UX the directory is `/etc/opt/sun/identity/config/xml`.) The structure of `amLogging.xml` is defined by file `sms.dtd`.

Logging Configuration

When Access Manager starts or when any logging configuration data is changed using the Access Manager console, the logging configuration data is loaded (or reloaded) into the Logging Service. This data includes the log message format, log file name, maximum log size, and the number of history files. Applications can use the client APIs to access the Logging features from a local or remote server. The client APIs use an XML-over-HTTP layer to send logging requests to the Logging component on the server where Access Manager is installed.

Recorded Events

The client passes the Logging Service logs information to the `com.sun.identity.log.LogRecord` class. The following table summarizes the items logged by default in the `LogRecord`.

TABLE 11–1 Events Recorded in LogRecord

Event	Description
Time	The date (`YYYY-MM-DD`) and time (`HH:MM:SS`) at which the log message was recorded. This field is not configurable.
Data	Variable data pertaining to the log records's `MESSAGE ID`. This field is not configurable.
Module Name	Name of the Access Manager service or application being logged. Additional information on the value of this field can be found in “Adding Log Data” on page 88.
Domain	Access Manager domain to which the user belongs.
Log Level	The Java 2 Platform, Standard Edition (J2SE) version 1.4 log level of the log record.

TABLE 11–1 Events Recorded in LogRecord *(Continued)*

Event	Description
Login ID	ID of the user as the subject of the log record. The user ID is taken from the session token.
IP Address	IP address from which the operation was performed.
Logged By	User who writes the log record. The information is taken from the session token passed during `logger.log(logRecord, ssoToken)`.
Host Name	Host name associated with the IP Address above.
MessageID	Non-internationalized message identifier for this log record's message.
ContextID	Identifier associated with a particular login session.

Log Files

The following sections contain information about Access Manager log files:

Log File Formats

Access Manager can record events in either of the following formats:

Flat File Format

The default flat file format is the W3C Extended Log Format (ELF). Access Manager uses this format to record the default fields in each log record. See "Recorded Events" on page 162 for a list of default fields and their descriptions. The following example illustrates an authentication log record formatted for a flat file. The fields are in this order: `Time`, `Data`, `ModuleName`, `MessageID`, `Domain`, `ContextID`, `LogLevel`, `LoginID`, `IPAddr`, `LoggedBy`, and `HostName`.

EXAMPLE 11–1 Flat File Record From `amAuthentication.access`

```
"2005-08-01 16:20:28"   "Login Success" LDAP    AUTHENTICATION-100
   dc=example,dc=com       e7aac4e717dda1bd01      INFO
uid=amAdmin,ou=People,dc=example,dc=com 192.18.187.152
"cn=exampleuser,ou=Example Users,dc=example,dc=com" exampleHost
```

Relational Database Format

When Access Manager uses a relational database to log messages, the messages are stored in a database table. Access Manager uses Java Database Connectivity (JDBC) to access the database table. JDBC provides connectivity to a wide range of SQL databases. JDBC also provides access to other tabular data sources such as spreadsheets or flat files. Oracle® and MySQL databases are currently supported.

For log records generated by Access Manager, the `Data` and `MessageID` fields are used slightly differently than in previous versions of Access Manager. Starting with this version of Access Manager, the `MessageID` field is introduced as a template for types of log messages. For example, in previous versions, Access Manager would generate the following message in the `Data` field:

```
Data: "Created group
cn=agroupSubscription1,ou=Groups,dc=iplanet,dc=com"
```

In this version of Access Manager, two log records are recorded for the one event:

```
Data:      agroupSubscription1|group|/
MessageID:    CONSOLE-1
```

and

```
Data:      agroupSubscription1|group|/
MessageID:    CONSOLE-2
```

These log records reflect the use of identities and realms. In this example, `CONSOLE-1` indicates an attempt to create an identity object, and `CONSOLE-2` indicates the attempt to create an identity object was successful. The root organization notation (`dc=iplanet,dc=com`) is replaced with a forward slash (`/`). The variable parts of the messages (*agroupSubscription1*, *group*, and */*) are separated by a pipe character (|), and continue to go into the `Data` field of each log record. The `MessagID` string is not internationalized in order to facilitate machine-readable analysis of the log records in any locale.

The following table summarizes the schema for a relational database.

TABLE 11–2 Relational Database Log Format

Column Name	Data Type	Description
`TIME`	VARCHAR(30)	Date of the log in the format `YYYY-MM-DD HH:MM:SS`.
`DATA`	VARCHAR(1024)	The variable data part of the log record pertaining to the MESSAGE ID. For MySQL, the Data Type is VARCHAR(255).
`MODULENAME`	VARCHAR(255)	Name of the Access Manager component invoking the log record.

TABLE 11–2 Relational Database Log Format *(Continued)*

Column Name	Data Type	Description
DOMAIN	VARCHAR(255)	Access Manager domain of the user.
LOGLEVEL	VARCHAR(255)	JDK 1.4 log level of the log record.
LOGINID	VARCHAR(255)	Login ID of the user who performed the logged operation.
IPADDR	VARCHAR(255)	IP Address of the machine from which the logged operation was performed.
LOGGEDBY	VARCHAR(255)	Login ID of the user who writes the log record.
HOSTNAME	VARCHAR(255)	Host name of machine from which the logged operation was performed.
MESSAGE ID	VARCHAR(255)	Non-internationalized message identifier for this log record's message.
CONTEXT ID	VARCHAR(255)	Identifier associated with a particular login session.

Error and Access Logs

There are two types of Access Manager log files:

- Access log files
- Error log files

Access log files record general auditing information concerning the Access Manager deployment. An access log may contain a single record for an event such as a successful authentication, or multiple records for the same event. For example, when an administrator uses the console to change an attribute value, the Logging Service logs the attempt to change in one record but, the Logging Service also logs the results of the execution of the change in a second record. *Error log files* record errors that occur within the application. While an operation error is recorded in the error log, the operation attempt is recorded in the access log file.

Flat log files are appended with the `.error` or `.access` extension. Database column names end with `_ERROR` or `_ACCESS`. For example, a flat file logging console events is named `amConsole.access` while a database column logging the same events is named `AMCONSOLE_ACCESS` or `amConsole_access`.

Note – The period (.) separator in a log filename is converted to an underscore (_) in database formats. Also in databases, table names may be converted to all upper case. For example, `amConsole.access` may be converted to `AMCONSOLE_ACCESS`, or it may be converted to `amConsole_access`.

Access Manager Component Logs

The log files record a number of events for each of the Access Manager components using the Logging Service. Administrators typically review these log files on a regular basis. The default location for all Access Manager log files is `/var/opt/SUNWam/logs` when Access Manager is installed in a Solaris environment. (When installed on Windows, the directory is *jes-install-dir*`\identity\logs`; on HP-UX the directory is `/var/opt/sun/identity/logs`.) The following table provides a brief description of the log files produced by each Access Manager component.

TABLE 11–3 Access Manager Component Logs

Component	Log Filename	Information Logged
Session	■ `amSSO.access`	Session management attributes values such as login time, logout time, and time out limits. Also session creations and terminations.
Administration Console	■ `amConsole.access` ■ `amConsole.error`	User actions performed through the administration console such as creation, deletion and modification of identity-related objects, realms, and policies. `amConsole.access` logs successful console events while `amConsole.error` logs error events.
Authentication	■ `amAuthentication.access` ■ `amAuthentication.error`	User logins and log outs, both successful and failed.
Federation	■ `amFederation.access` ■ `amFederation.error` ■ `amLiberty.access` ■ `amLiberty.error`	Federation-related events such as the creation of an authentication domain or the creation of a hosted provider entity.
Authorization (Policy)	■ `amPolicy.access` ■ `amPolicy.error` ■ `amAuthLog`	Policy-related events such as policy creation, deletion, or modification, and policy evaluation. `amPolicy.access` logs policy allows, `amPolicy.error` logs policy error events, and `amAuthLog` logs policy denies.
Policy Agent	`amAgent`	Exceptions regarding resources that were either accessed by a user or denied access to a user. `amAgent` logs reside on the server where the policy agent is installed. Agent events are logged on the Access Manager machine in the Authentication logs.
SAML	■ `amSAML.access` ■ `amSAML.error`	SAML-related events such as assertion and artifact creation or removal, response and request details, and SOAP errors.

TABLE 11–3 Access Manager Component Logs *(Continued)*

Component	Log Filename	Information Logged
Command-line	■ `amAdmin.access` ■ `amAdmin.error`	Event successes and errors that occur during operations using the command line tools. Examples are: loading a service schema, creating policy, and deleting users.
Password Reset	■ `amPasswordReset.access`	Password reset events.

For detailed reference information about events recorded in each type of Access Manager log, see the *Sun Java System Access Manager 7.1 Administration Guide*.

Additional Logging Features

You can enable a number of logging features for added functionality. The additional features include:

- "Secure Logging" on page 167
- "Remote Logging" on page 167
- "Log Reading" on page 168

Secure Logging

This feature adds an extra measure of security to the Logging Service. When secure logging is enabled, the Logging component can detect unauthorized changes to the security logs. No special coding is required to leverage this feature. However, secure logging uses a certificate that you must create and install in the container that runs Access Manager. When secure logging is enabled, a Manifest Analysis and Certification (MAC) is generated and stored for every log record, and a special signature record is periodically inserted in the log. The signature record represents the signature for the contents of the log written up to that point. The combination of the certificate and the signature record ensures that the logs have not been tampered. For detailed information about enabling secure logging, see the *Sun Java System Access Manager 7.1 Administration Guide*.

Remote Logging

Remote logging allows a client using the Client APIs to create log records on an instance of Access Manager deployed on a remote machine. Remote logging is useful in the following situations:

- When the login URL in the Naming Service of an Access Manager instance points to a remote Access Manager instance, and a trust relationship between the two instances has been configured.

- When the Access Manager APIs are installed in a remote Access Manager instance, and a client application or a simple Java class running on the Access Manager server uses the logging APIs.
- When logging APIs are used by Access Manager agents.

Log Reading

Access Manager provides Logging APIs for writing your own custom log reading program. You can set up queries to retrieve specific records from the log file or database. This is useful for auditing purposes. For more information, see the *Sun Java System Access Manager 7.1 Developer's Guide*.

Java Enterprise System Monitoring Framework

Access Manager 7.1 integrates with the Java Enterprise System (JES) monitoring framework through Java Management Extensions (JMX). JMX technology provides the tools for building distributed, web-based, modular, and dynamic solutions for managing and monitoring devices, applications, and service-driven networks. Typical uses of the JMX technology include: consulting and changing application configuration, accumulating statistics about application behavior, notification of state changes and erroneous behaviors. Data is delivered to centralized monitoring console. Access Manager 7.1 uses the Java ES Monitoring Framework to capture statistics and service-related data such as:

- Number of attempted, successful, and failed authentications
- Number of active sessions, statistics from session failover DB
- Session failover database statistics
- Policy caching statistics
- Policy evaluation transaction times
- Number of assertions for a given provider in a SAML/Federation deployment

For comprehensive information about how the JES monitoring framework works and how you can use the monitoring framework with Access Manager, see the *Sun Java Enterprise System 5 Update 1 Monitoring Guide*.

◆ ◆ ◆ CHAPTER 12

Third-Party Product Integration

Intro text

ID Manager

(Rahul G)

Site Minder

(Malla)

Oracle Access Manager

(Malla)

Index

R

S

T

U

V

W

X