J2EE Agent Architecture

Open Web Single Sign-On

Version 1.0

Please send comments to: dev@opensso.dev.java.net

Author **Hua Cui**
(hua.cui@sun.com)
Technical Lead

This document is subject to the following license:

Contents

1 Introduction

1.1 Document Status

Project Name	Open Web Single Sign-On
Document Title	J2EE Agent Architecture
Date of Issue	June 21, 2006
Current Version	1.0
Author	Hua Cui (hua.cui@sun.com)
Issuing Organization	Sun Microsystems, Inc.
Feedback E-mail	Opensso@sun.com

1.2 Revision History

Date	Version	Author	Comments
May 30, 2006	0.1	Hua Cui (hua.cui@sun.com)	Initial Revision
June 21, 2006	0.2	Hua Cui (hua.cui@sun.com)	First Revision
July 20, 2006	1.0	Hua Cui (hua.cui@sun.com)	Second Revision

1.3 Summary

J2EE agents enable deployment containers to enforce authentication and authorization using OpenSSO services. To ensure secure client access to hosted J2EE applications, J2EE agents enforce the following:

- Single sign-on (SSO)
- URL Policies (defined in OpenSSO)
- J2EE Declarative and Programmatic Security

J2EE agents can protect a variety of hosted J2EE applications, which can in turn require policy implementation that varies greatly from application to application. The security infrastructure of J2EE provides declarative as well as programmatic security that is platform-independent and is supported by all the J2EE-compliant deployment containers. For details on how to use the declarative and programmatic security of the J2EE platform, refer to J2EE documentation, available at http://java.sun.com/j2ee

J2EE agents help enable role-to-principal mapping for protected J2EE applications with OpenSSO principals. Thus at runtime, when a J2EE policy is evaluated, it is done against the information available in OpenSSO. Using this functionality, administrators can configure their hosted J2EE

applications to be protected by the agent, which provides real security services and also other key features such as single sign-on. Apart from enabling the J2EE security for hosted applications, the J2EE agents also provide complete support for OpenSSO-based URL policies for enforcing access control over web resources hosted in the deployment container.

The purpose of this document is to provide the architectural details for the implementation of the J2EE Agent in the Open Web Single Sign-On project (also referred to as OpenSSO) in satisfying the system requirements and the architectural concerns. The System Architecture Document [1] was used as a reference to write this document. The structure of this document is based on the recommendations provided by IEEE Standard 1471-2000 [1]. All the important terms, acronyms, or abbreviations used in this document are defined in the Glossary section.

1.4 Scope

This document describes the detailed architecture of the J2EE Agent in the OpenSSO project. It is not a requirement document although it does reflect the functional requirements addressed by this architecture.

The purpose of this document is to provide certain level of specifics about the J2EE Agent framework as well as to serve as an effective vehicle for facilitating any design and implementation practice pertaining to the J2EE Agent.

1.5 Context

J2EE Agent is installed on a J2EE Application Container. It uses OpenSSO's Authentication Service and Session Service for Authentication and uses Policy Service for Authorization. It also uses Logging Service to keep an audit trail of what resources are being accessed or denied.

Generally speaking, the J2EE Agent should perform some or all of the following actions:

- To control access to the whole web application or part of the web application
- Ability to do only authentication and not authorization
- Ability to do URL Policy based authorization
- Ability to do J2EE Policy based authorization
- Ability to do user attribute mapping between OpenSSO user attributes and container user attributes
- Ability to do configuration properties hot swapping
- Passing policy based personalization information to protected applications.
- Redirect to custom URL to handle error conditions
- To maintain an audit log of what all the resources were accessed and denied and maintain them in a centralized logging system
- Work in a Load Balancer environment

1.6 Glossary

Administrator	A privileged *user* who is responsible for configuring the *system* so that it can achieve *SSO*.
Authentication	The process by which the identity of a *user* or *administrator* is established within the *system*. This process may involve explicit user interaction with the system outside the scope of any of the *web applications* that participate in *SSO*.
Authentication Service	A *service* that facilitates the *authentication* of *users* and *administrators* within the *system*.
Authorization	The process of deciding whether and how a user can access a resource.
Authorization Service	A service that evaluates *policy*, deciding whether a given *user* is *authorized* to access a given resource.
Client	An entity that accesses a *service* within the *system*.
Cookie	A mechanism that allows a web server to store some data on the browser that accesses that server. RFC2965 [8]
Domain	A suffix used in fully qualified host names that allows the logical grouping of hosts.
Filter	A filter is a Java class that is invoked in response to a request for a resource in a Web Application.
Firewall	An entity that limits access to and from a network based on the configured security policies.
FQDN	Fully Qualified Domain Name
HTTP	Acronym for Hypertext Transfer Protocol. This is an open standards based protocol used for exchange of information between web browsers and web servers. RFC2616 [9]
J2EE	Java 2 Platform, Enterprise Edition
Logging Service	A service that performs remote logging.
OpenSSO	Abbreviation for the Open Web Single Sign-On project. This project is an open source initiative of Sun Microsystems Inc., that provides the foundation of identity services for the web platform.
Policy	Set of rules, subjects and constraints grouped together to define authorization permissions.

Realm	A *realm* is a collection of users that are controlled by the same J2EE authentication policy.
Service	An abstraction that represents functionality provided by a subsystem which can be accessed anywhere within the network using appropriate request and response message constructs.
Session Service	A *service* that provides the ability to associate a *user session* with a particular *user* once that user has successfully *authenticated*.
Session time-out	A preset interval of time after which the *user session* is considered invalidated. A session time-out can be a hard time-out or an idle time-out value depending upon the configuration of the *system*.
SSO	Abbreviation for Single Sign-On. SSO is defined as the ability of a *user* to *authenticate* once and gain access to a variety of *web application* resources that otherwise would have required individual authentication, with each authentication potentially requiring different set of credentials.
System	In the context of *OpenSSO,* the System represents a complete deployment where various *web applications* participate in an *SSO* environment using the identity services provided by OpenSSO.
System Stakeholder	A set of people who interact with the *system* at various stages and in different capacities. The system stakeholders could be individuals, teams, or organizations.
URL	Acronym for Uniform Resource Locator. A URL contains the necessary information regarding the address and access mechanism needed to access a resource available on the network. RFC1738 [10]
User	The user of the *system* is an end user who is interested in accessing one or more *web applications* that participate in *SSO*. This user has no administrative privileges and cannot change the behavior of the system for other users. This user may be able to change the behavior of system as experienced by self to the extent allowed by the *Administrator*.
User Session	An interval of time for which a *user* is considered *authenticated* and the associated identity information is available to all participating *web applications* in *SSO*. A user session begins with the successful authentication of the user and ends with the invalidation of session either by a direct action of the user such as an explicit logout, or by indirect means such as configured *session time-out* or being invalidated by an *administrator*.
Web Application	An application hosted on either a web server or an application server and is accessible via the web using a traditional browser.
XML	Acronym for Extensible Markup Language. XML is an open standards based data markup language used for representing structured data.

1.7 References

[1] IEEE Std. 1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE-SA Standards Board – September 2000

[2] Arvind Prabhakar, System Architecture – *Open Web Single Sign-On*, Version 1.0

[3] Dennis Seah, *Use Cases – Open Web Single Sign-On*, Version 1.0

[4] Alan Chu, Session Service Architecture – *Open Web Single Sign-On*, Version 1.0

[5] Mrudul Uchil, Authentication Service Architecture – *Open Web Single Sign-On*, Version 1.0

[6] Dilli Dorai, Policy Service Architecture – *Open Web Single Sign-On*, Version 1.0

[7] Madan Ranganath, Web Agent Architecture – *Open Web Single Sign-On*, Version 1.0

[8] HTTP State Management Mechanism http://www.ietf.org/rfc/rfc2965.txt

[9] Hypertext Transfer Protocol – HTTP/1.1 http://www.ietf.org/rfc/rfc2616.txt

[10] Uniform Resource Locators (URL) http://www.ietf.org/rfc/rfc1738.txt

2 Objectives

2.1 Mission

The mission of J2EE Agent is to protect a variety of hosted J2EE applications. In order to accomplish this the J2EE Agent communicates with the Authentication Service, Session Service, Policy Service and Logging Service of the OpenSSO via XML over HTTP. A J2EE Application Server might have a number of resources. The J2EE Agent should guarantee that only users with the right credentials can access the resources. The J2EE Agent has the responsibility of maintaining the SSO token till the user SSO token becomes invalid. The J2EE Agent also has the responsibility of dynamically changing its behavior to reflect the changes in the above mentioned services.

2.2 Stakeholders

System stakeholders are the people who have interests in, or concerns relative to, the system. These could be individual users, teams, and organizations that are charted with the development, adoption or execution of the system. The key stakeholders for the J2EE Agent are:

- *Developers*: Responsible for the overall development of the system. They may be involved in various development related activities associated with the J2EE Agent such as designing, developing, building, testing, documenting, and troubleshooting the functionalities provided by the J2EE Agent.
- *Administrators*: Privileged users who are chartered with deployment and configuration of the system in staging and production environments. Administrators can control the system behavior via the available configuration mechanisms at various levels and thus affect the way the system operates. They are expected to perform system checks to ensure its operations and take corrective actions where necessary if the system fails to perform satisfactorily.
- *Application Developers*: Developers who are responsible for the creation and deployment of web resources/applications. These developers may use the J2EE Agent to protect their web resources or J2EE applications.
- *Application Server Administrators*: Administrators are responsible for the maintenance of the Application server. These administrators will configure J2EE Agent as necessary in order to ensure that the web resources or any J2EE applications are protected.
- *System Integrators*: Developers who are chartered with the deployment of OpenSSO in a given environment to best utilize the J2EE Agent. They may be involved in building new J2EE Agents and/or deploying the J2EE Agent in a Load Balancer or cluster environment.

2.3 Architectural Concerns

There are quite a few architectural concerns which are identified and considered in formulating the architectural concept of the J2EE Agent from the perspectives of all relevant stakeholders. While the system wide concerns are already addressed in the OpenSSO System Architecture Document [2], this section intends to focus on the core concerns which are derived specifically from the system requirements for the J2EE Agent.

2.3.1 Resource Protection

When the J2EE Agent is installed and configured for a web application, it protects the whole web application. The Administrator might decide to have portion of the web application to be accessed without any authentication or authorization and the rest to be protected. The J2EE Agent should address this concern.

2.3.2 Ease of Installation and Configuration

The J2EE Agent should be easy to install. Making J2EE Agent configuration changes should also be simple.

One concern that Administrators have is to protect multiple web application server instances on the same system. Ability to install multiple versions of the J2EE Agent to protect multiple web application server instances on the same system should be addressed.

Another concern that should be addressed in the location where the application server is running (the J2EE Agent being installed here) and where the Authentication, Session, Policy and Logging services are running. One system can be within a Firewall and the other might be outside the Firewall or both the system can be inside the Firewall. The J2EE Agent should work in these kinds of installation.

2.3.3 Support Load Balancer

In a huge deployment scenario there might be a Load Balancer in front of multiple application servers. The Administrator should be able to install J2EE Agent in this kind of scenario to protect the web applications.

2.3.4 Notifications

The data on the Authentication, Session and Policy Service might change periodically or over a period of time. There should be some mechanism by which this change is informed to the J2EE Agent. J2EE Agent in turn should update its internal data structures to reflect this change.

2.3.5 Configuration Properties Hot Swapping

There are needs to change some of the J2EE Agent configuration properties and have the changes take

effect without having to restart the agent. J2EE Agent should support this configuration properties hot swapping.

2.3.6 Performance and Scalability

Given the limitation of the capabilities of the underlying hardware and software components, the J2EE Agent should perform and scale up to the necessary levels in order to accommodate more users.

2.3.7 Auditing

The J2EE Agent Administrator might be interested in knowing which resources are being accessed. The J2EE Agent should keep an audit trail of what resources are being accessed or denied.

2.3.8 Security

As the J2EE Agent interacts with a number of services (Authentication, Session, Policy and Logging), user's data should never be compromised. This might involve user's authentication data, session attributes values and policy data. Also malicious HTTP headers attach should also be prevented.

3 Architectural Views

3.1 J2EE Agent Client View

3.1.1 Viewpoint Specification

Name	J2EE Agent Client Viewpoint
Stakeholders	Developers
Concerns	Resource Protection, Ease of Installation and Configuration, Notifications, Auditing
Modeled As	Collaboration
Viewpoint Source	System Requirements

3.1.2 Details

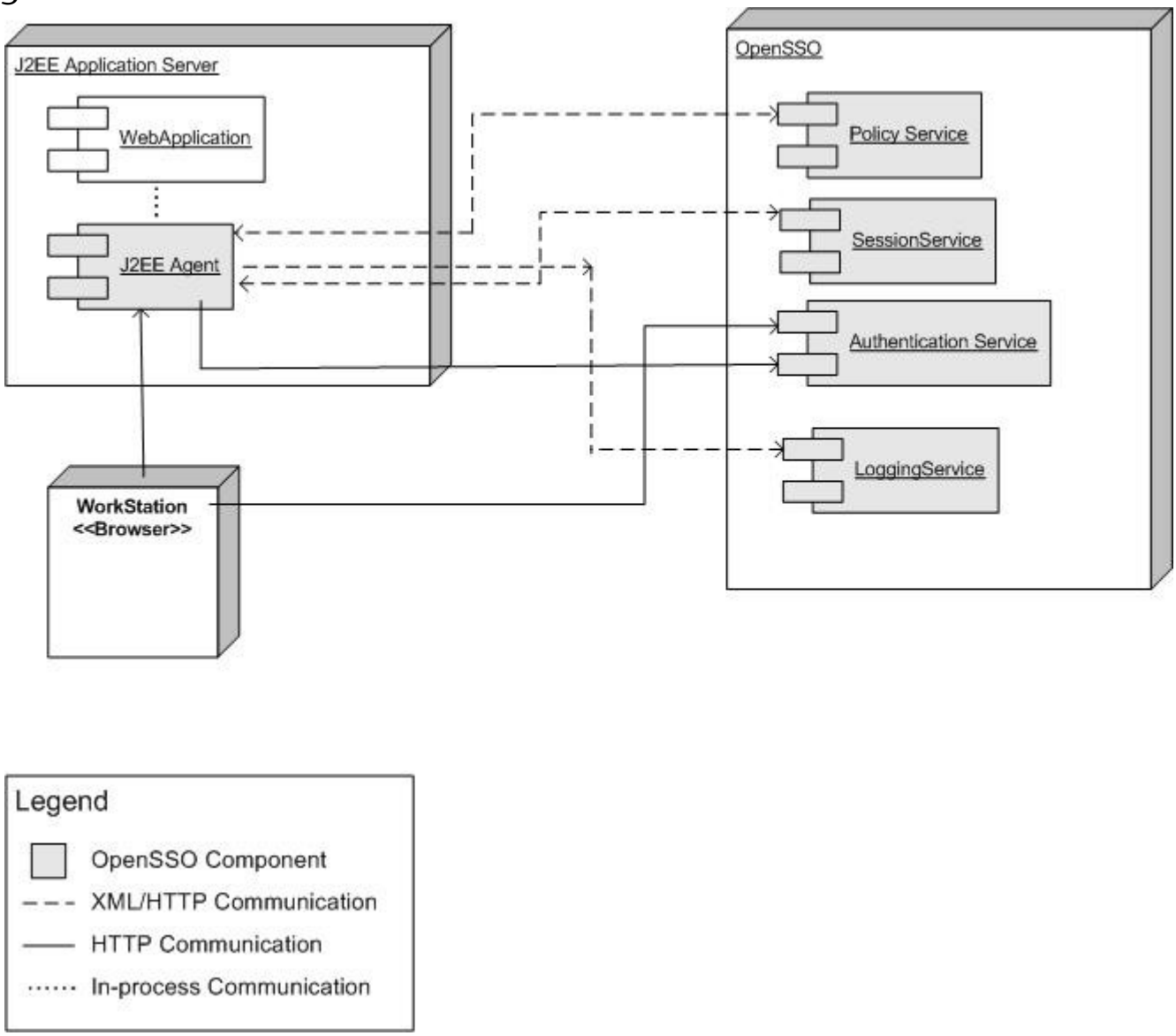

Figure 1: J2EE Agent Client View

3.1.3 Description

This view describes a simple deployment scenario for J2EE Agent. The OpenSSO components described in this view are the core identify services – Authentication, Session, Policy, Logging, along with the J2EE Agent. In this view the J2EE Agent is deployed in the J2EE Application Container and is protecting the web applications that are present on the container. The J2EE Agent is the first to intercept the

HTTP/HTTPS request. It validates the SSO Token with the Session Service. It redirects the request to the Authentication Service if the token is not found or invalid. It interacts with the Policy Service to obtain policy decisions to determine whether to grant or deny access. It also establishes the user principals that later being used to satisfy the J2EE Policy if any. Administrators can use the property file used by the J2EE Agent to configure what resources should be protected and what should not be.

3.2 J2EE Agent Transition View

3.2.1 Viewpoint Specification

Name	J2EE Agent Transition View
Stakeholders	Administrators
Concerns	Resource Protection, Ease of Installation and Configuration, Notifications, Auditing
Modeled As	Collaboration
Viewpoint Source	System Requirements

3.2.2 Detail

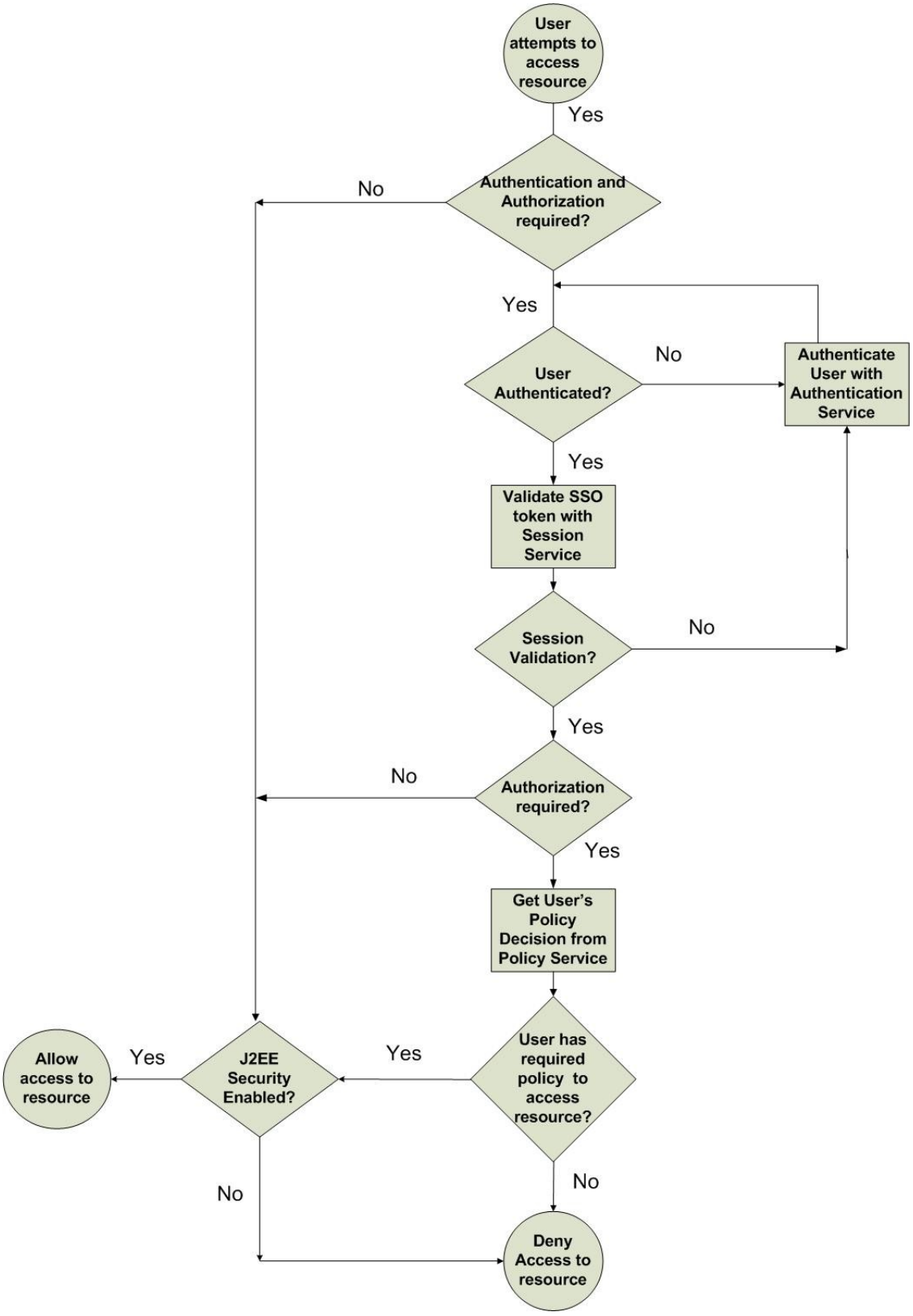

Figure 2: J2EE Agent Transition View

3.2.3 Description

This view depicts how the J2EE Agent intercepts the request and interacts with various Services before either allowing the user to view the resource or denying the user from viewing the resource.

3.3 Load Balancer Enabled Deployment View

3.3.1 Viewpoint Specification

Name	Load Balancer Enabled Deployment View
Stakeholders	System Integrators, Ease of installation and configuration
Concerns	Support Load Balancer and Proxy Server Environment, Security, Performance and Scalability
Modeled As	Deployment
Viewpoint Source	System Requirements

3.3.2 Detail

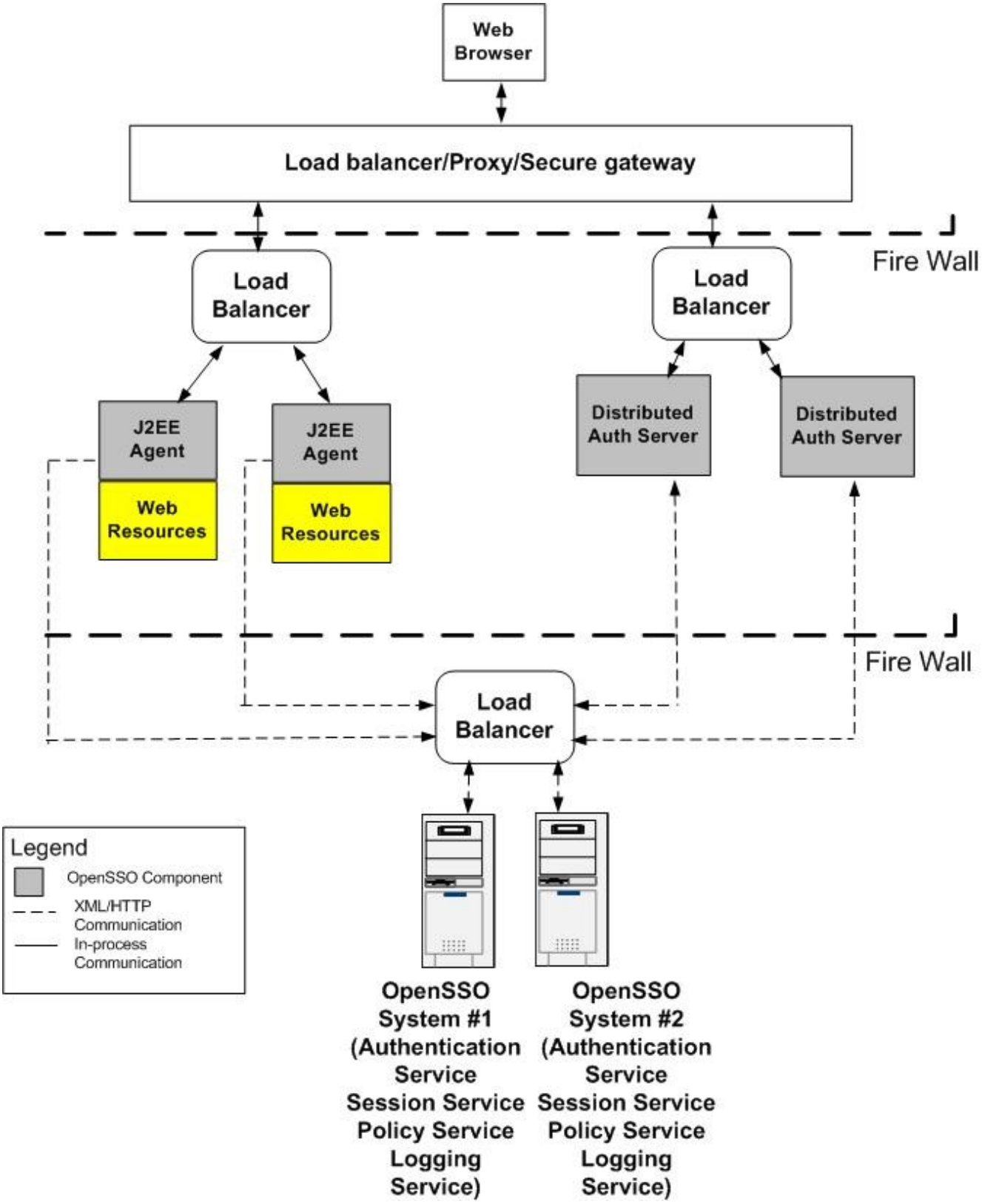

Figure 3: Load Balancer View

3.3.3 Description

The above view depicts one such scenario where multiple application servers are protected by J2EE Agent. The J2EE Agent should work fine in a Load Balancer, Proxy or a Firewall environment.

3.3.4 Extension

This view can be further extended to address the needs of the current deployment environment.

4 J2EE Agent Conceptual Implementation

The J2EE agent can be viewed as having two major components – an agent filter, and an agent realm; or more simply – a filter and a realm. The filter transparently intercepts a user request and allows the request to continue to its destination if no specific action is deemed necessary. The realm facilitates the fetching of userbase information from the IDRepo service and make it available for use within the agent implementation. These two components together form the basic skeleton for all agents. There are other components that exist within the agent implementation, but the most processing logic is actually handled by the filter and the realm components.

4.1 Agent Filter

The filter transparently intercepts a user request and allows the request to continue to its destination if no specific action is deemed necessary. Most regular agents require the configuration of a Servlet Filter which in turn delegates to this filter component. The agent filter has five Modes of Operation. It facilitates Single Sign-On, URL Policy, and J2EE Policy to protect the web applications residing on the same J2EE Application container.

4.1.1 Filter Modes of Operation

The following is a list of various modes of operation that the filter component can execute in:

- 1. NONE
- 2. SSO_ONLY
- 3. URL_POLICY
- 4. J2EE_POLICY
- 5. ALL

The NONE mode of operation implies that the filter does not do any processing of the request but one thing that the filter component always does, even in the NONE mode, is to generate an audit trail. In the NONE mode, the filter component does not know who the user is and thus the audit trail has no user information in it. However, it contains information about all the application resources that were intercepted by the filter component.

The SSO_ONLY mode implies that the filter component will ensure that the user has a valid AM session before allowing the request to proceed. If the protected applications on the application server have some form of J2EE security in place, it will require the user to authenticate regardless of whether the user has a valid AM session or not. In other words, this is a specific definition of SSO in which we mean that the user is authenticated with AM and has a valid session. This mode is very useful in cases where

the protected application does not have any security semantics . For example, this could be the case for a report application that provides reports which may be visible to all the authenticated users and not accessible to the anonymous users.

The URL_POLICY mode builds on top of the SSO_ONLY mode by adding the enforcement of URL policies to evaluate if the user can have access to the requested resource. This is very similar to how a web agent operates. See “Web Agent Architecture” (Reference [7]) for more information on how web agents operate.

The J2EE_POLICY mode builds on top of the SSO_ONLY mode by allowing local authentication and thus establishing the local identity. This, for a regular agent, implies that the user's principal is established and all the role-membership information is provided to the security subsystem of the application server by the agent.

The ALL mode of operation is the most restrictive mode of operation where the filter does all of what it does for the SSO_ONLY, URL_POLICY and J2EE_POLICY modes. In other words, this mode requires that the user has an authenticated session, that all applicable URL policies be evaluated and that the user's local identity be established before the request is allowed to proceed to its intended destination.

4.1.2 Task Handling Mechanism

The various modes of operations as described in the previous section perform independent units of work in different combinations to achieve different overall effects. At a high level, these independent units of work, or tasks, are:

- Session Validation
- URL Policy Enforcement, and
- Establishing Principal for the user.

Different combinations of these three tasks result in the various modes of operation as illustrated in the following table:

	Session Validation	*URL Policy Enforcement*	*Establishing Principal*
NONE	no	no	no
SSO_ONLY	yes	no	no
URL_POLICY	yes	yes	no
J2EE_POLICY	yes	no	yes
ALL	yes	yes	yes

This independent grouping of these tasks was the primary motivation of implementation of a task

handling mechanism in the filter component implementation. This implementation is based on the Chain of Responsibility design pattern. In a chain of responsibility design, independent units of work are modeled as tasks for which specialized task handlers are implemented. Each specialized task handler focuses on accomplishing its own task requirement with no regard to any other task related details.

When a request is received, these task handlers are invoked in a chain, with the provision that if any task handler requires special handling of the request, it would force the chain to break and the request to be fulfilled at that point of time in processing. In other words, the request passes from the first task handler to the next and so on, until the time when either all the task handlers have executed, or until some task handler breaks this chain by returning a result that fulfills the request.

As an example, in the case of filter component, there is one task handler for validating session, one for enforcing URL policies, and one for establishing the principal. The filter component itself maintains these three task handler instances and uses them to handle requests. When a request from an anonymous user is received, the filter component invokes the session validation task handler. Since the user does not hold a valid session, the session task handler breaks the request processing chain by fulfilling the request via a means of issuing an authentication redirect. The next time the request comes around, the user is authenticated and in this case the session validation task handler does not require any special handling of the request. In this case, the request trickles down to the next task handler – the URL policy enforcement task handler, and so on.

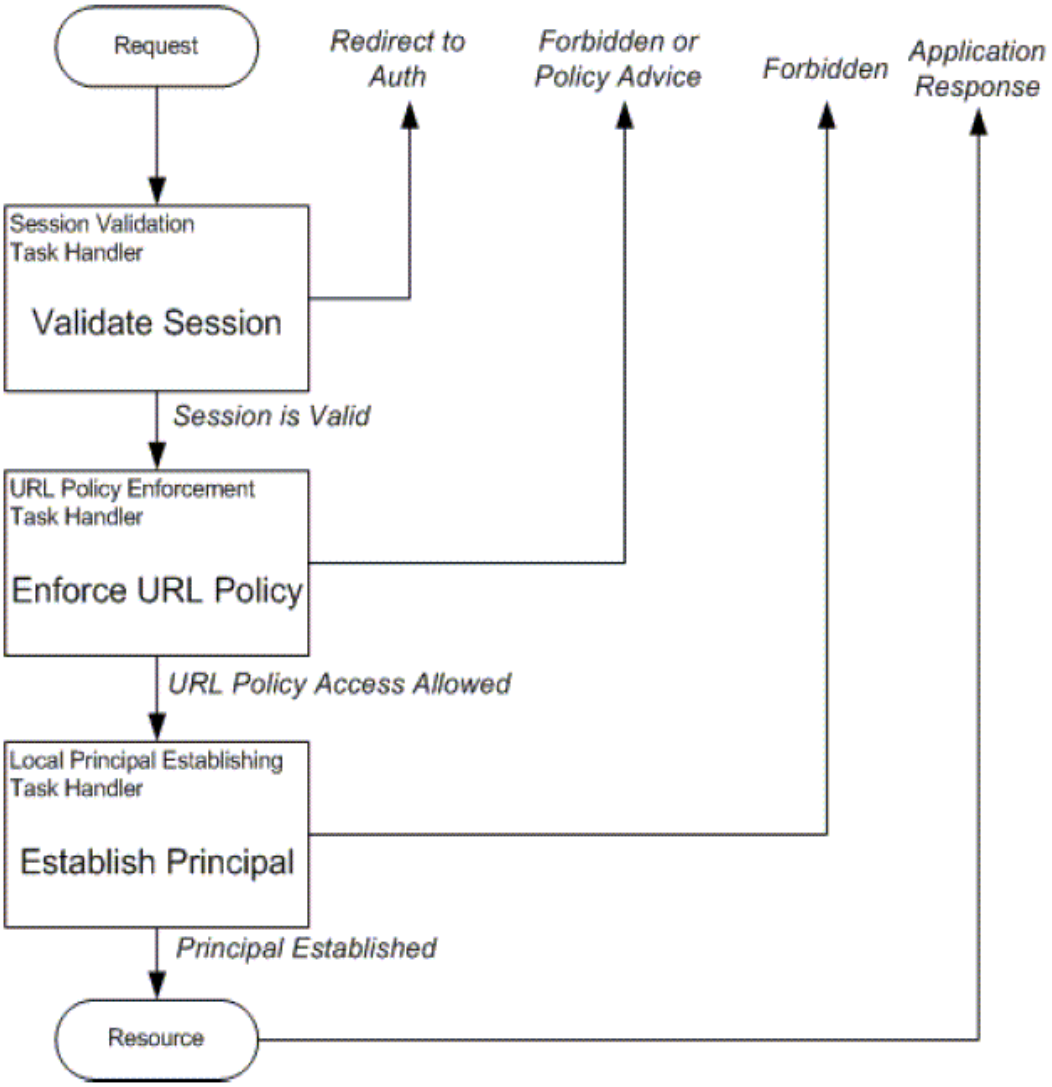

Figure 4: Sample Task Handler operation in Chain of Responsibility

These task handlers perform independent work that does not directly relate to each other in any way other than the order of their invocation. For example, except for the executing of session validation task handler before URL policy evaluation task handler, there is no relation between the two. The order of execution itself is also not that rigidly related since if URL policy task handler were to execute first, it would have ensured that the user gets authenticated, thereby implicitly making the session validation task handler redundant.

With the above separation of task handlers, all that is needed for the implementation of various task handlers is the ability to selectively include or exclude certain task handlers. In our hypothetical example of the filter consisting of three task handlers, when the mode of operation is SSO_ONLY, the filter needs to include only the Session Validation task handler in its request processing chain and ignore the other two. The actual implementation of task handlers within the filter component is not so coarse-grained as described in this example. The real task handlers that are used within the filter component are much more fine grained than these high level task handlers. The core idea here is – more fine grain the task handler is, the more flexible the filter behavior can be during runtime. Greater flexibility of the filter behavior results in its greater applicability to various diverse requirements thus making it an ideal component that can be tailored to any particular agent requirement. With so many task handlers, the filter component itself is not aware of the exact nature of these handlers, but instead relies on simple looping logic to process the request. The loop is executed over an ordered collection of task handlers that are identified and initialized during the initialization of the filter component. As the filter iterates over the ordered collection of task handlers, every iteration proceeds to the next task handler if and only if the previous task handler did not produce any result to fulfill the original request. When all the task handlers have completed execution, the request is allowed to proceed to the requested resource.

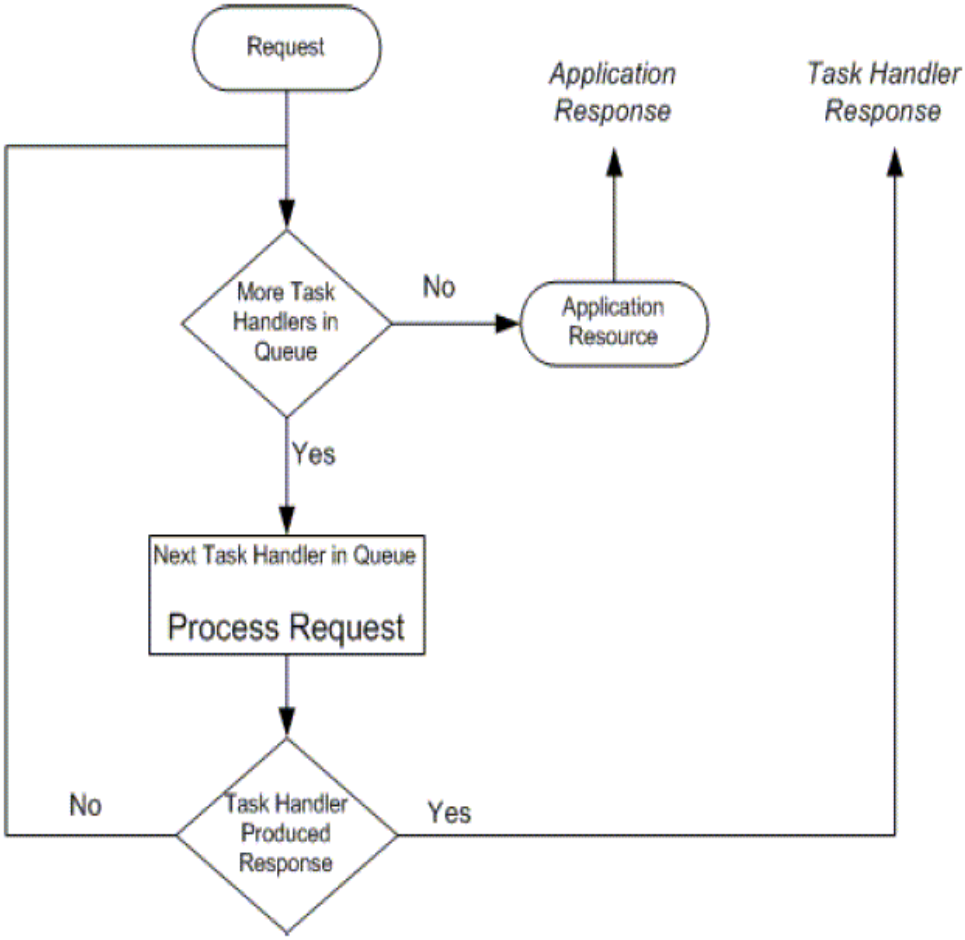

Figure 5: Task Handler looping logic in filter component

4.2 Agent Realm

The realm component provides the necessary means to supply userbase information to the agent runtime. This information is then used by the agent specific mechanism such as a custom realm implementation to supply this information to the underlying security subsystem of the application server. The realm component does the following three things:

- Given a user name and agent generated credential (also called the Transport Token), authenticate the user.
- If the user is a valid user, fetch the role-membership information as applicable to that user.
- Optionally, verify with an external store if the user exists in that store or not.

The authentication of the user is performed with a user name and an agent generated credential – the Transport Token. The transport token is nothing but a marshalled attribute pair representation that includes the user's SSO token, IP address, current application name and the user's mapped user-id among other things. The implementation of the transport token is such that it can accommodate arbitrary name value pairs. It offers the ability to generate plain-text and encrypted string representations that can be used as form values or cookie values where necessary. Given the user name and transport token, the realm authenticates the user by validating the user's session and if configured to do so, optionally, verify if the user does exist in the external store. Once the authentication and external verification is successful, the realm fetches all the applicable role memberships of the user and makes them available as a part of the authentication result. Note that the role membership of the user as determined by the realm is not limited the the actual role membership of the user, but can be extended via means of configuration to include other information such as group membership, filtered role membership, session properties etc.

4.3 Sequence of Operations

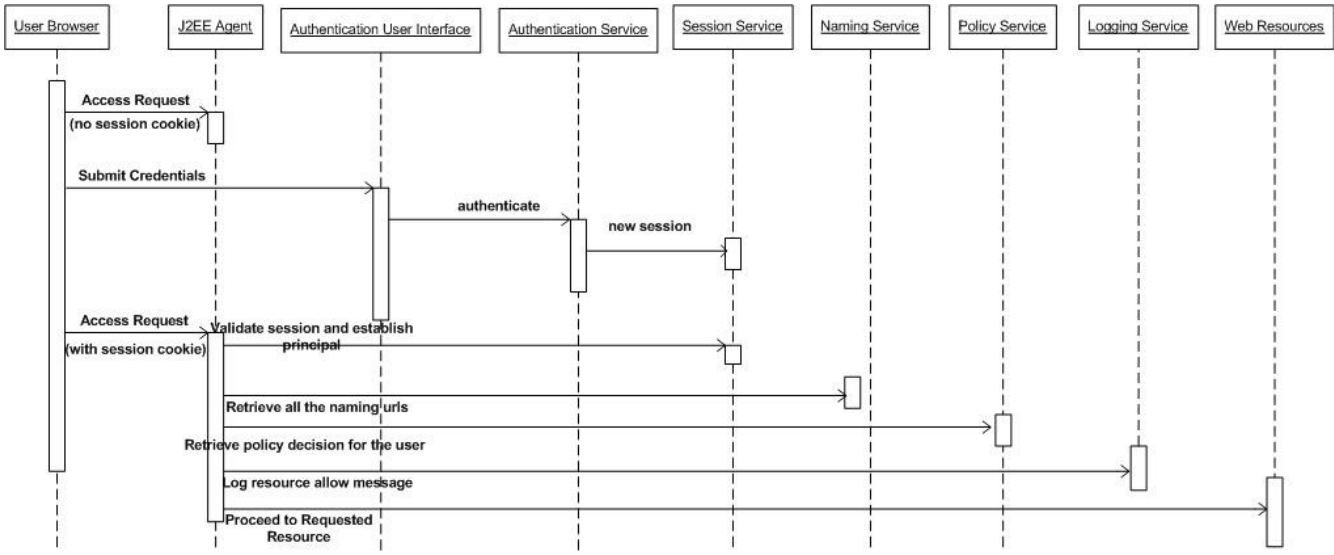

Figure 6: Sequence of Operations

The J2EE Agent does the 1 and 2 during its initialization.

1. The J2EE Agent authenticates itself to OpenSSO server as an application using a special user created in OpenSSO for this agent. The OpenSSO sends back an *appSSOToken* after the authentication succeeds.
2. *The J2EE Agent interacts with the Naming Service by sending the appSSOToken* and *retrieves all the Identity Service URLs (Policy Service, Logging Service)*

The following steps are repeated by J2EE Agent for every user request.

3. When a user tries to access the web resources, the J2EE Agent intercepts this request and checks whether the user has a valid SSO token aka session cookie.
4. If the session cookie is not found, the J2EE Agent will issue a client-side redirect to the OpenSSO Authentication Service with sufficient information in the query string to allow the Authentication Service to redirect the user back to the originally requested address once authentication is complete.
5. The Authentication Service will gather and verify the user's credentials. If the credentials are verified successfully, the Authentication service will issue a request to create a new user session, and set the user session handle as a domain cookie before redirecting the user back to the originally requested address.
6. Upon successful user authentication, the J2EE Agent establishes the principals for the user. The user principals are used later to check against the J2EE policies if any for the web application.
7. If a session cookie is found in user's request, the J2EE Agent will validate the cookie by sending

the session cookie to the Session Service. If the session is valid, the J2EE Agent will proceed to the next set of Services.

8. The J2EE Agent then tries to retrieve the user's policy decision for that particular resource from its policy cache.
9. If the policy decision is not found in the J2EE Agent policy cache, the J2EE Agent retrieves the user's policy decision by sending the user's SSO token, *appSSOToken,* and the resource that the user is trying to access to the OpenSSO Policy Service. The Policy Service will return with a *allow* or a *deny* policy response.
10. The J2EE Agent then does a remote log what resources were allowed or denied by using the Logging Service.
11. Now that the user has a valid SSO token, has the required policy defined to access the resource, the J2EE Agent allows the user to view the resource.

4.4 J2EE Agent Implementation

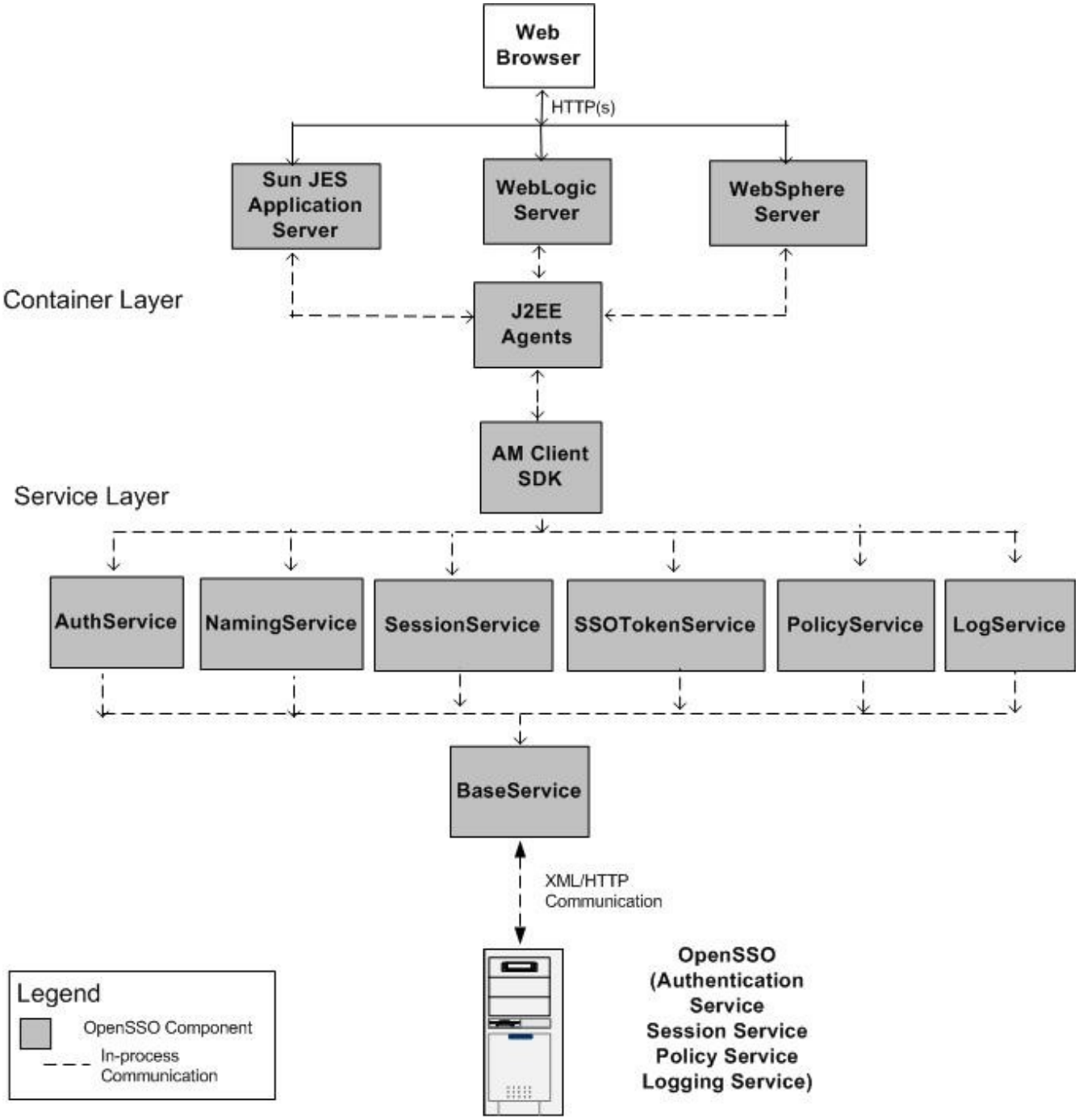

Figure 7: J2EE Agent Implementation

The implementation is divided into two layers:

- Web Container Layer
- Service Layer

4.4.1 Container Layer

In this layer, the individual J2EE Agent for a specific J2EE Application Container is implemented. Each agent implementation contains the two main components: Agent Filter and Agent Realm. These components have the logic to handle the major functionalities of the agent. i.e. achieving Single Sign-On, enforcing URL Policy, and establishing user principals for J2EE Policy.

4.4.2 Service Layer

The Service Layer layer, or the OpenSSO Client Layer, contains the client side APIs of the OpenSSO. The APIs provide the access to all different services that the OpenSSO facilitates on the server side. These services include Naming Service, Authentication Service, Session Service, Policy Service, Logging Service, and etc. These services provide the means for the client to create application and user SSO tokens, generate Naming Request, evaluate URL Policies, cache Policy Decisions, register and process notifications, and so on.

4.5 J2EE Agent Configuration Properties File

J2EE Agent uses the AMAgent.properties file for Initialization and Configuration purposes. The J2EE Agent AMAgent.properties configuration file contains the necessary configuration properties needed for the agent to function properly. It also contains the necessary information needed for the OpenSSO Client SDK to function properly in a client installation mode as used by the agent. During Application Server start-up the J2EE Agent reads the property file and populates its data structure with this information. The Configuration Property Hot Swapping feature of J2EE Agent provides runtime configuration update for certain configuration properties specified in the AMAgent.properties file so that the container doesn't have to be restarted for the configuration value change to take effect.

The following are some of the main configuration properties specified in AMAgent.properties:

- Agent Filter Operation Mode Property

com.sun.identity.agents.config.filter.mode (Hot Swap enabled: No)

This property specifies the mode of operation of the agent filter. The following are valid values:

1. NONE
2. SSO_ONLY
3. URL_POLICY
4. J2EE_POLICY
5. ALL

- Agent Username and Password Properties

  com.sun.identity.agents.app.username (Hot Swap enabled: No)

  This property specifies the user name used by the agent to identify and authenticate itself to OpenSSO before requesting any services to the OpenSSO.

  com.iplanet.am.service.secret (Hot Swap enabled: No)

  This property specifies the password used by the agent to identify and authenticate itself to OpenSSO before requesting any services to the OpenSSO.

- User Mapping Properties

  com.sun.identity.agents.config.user.mapping.mode (Hot Swap enabled: No)

  This property specifies the mechanism by which the user ID used on the protected server for the authenticated user is determined by the J2EE agent. The following are valid values for this property:

1. USER_ID
2. PROFILE_ATTRIBUTE
3. HTTP_HEADER
4. SESSION_PROPERTY

  com.sun.identity.agents.config.user.attribute.name (Hot Swap enabled: No)

  This property specifies the name of the profile attribute, HTTP header, or session property that

  contains the user ID used on the protected server for the authenticated user. This property is not used when property com.sun.identity.agents.config.user.mapping.mode = USER_ID

  com.sun.identity.agents.config.user.principal (Hot Swap enabled: No)

  This property is a flag that indicates how the user is authenticated on the protected server.

When this property is set to true, the principal of the authenticated user, not simply the user ID, is used for authentication purposes. This property is only used when property com.sun.identity.agents.config.user.mapping.mode = USER_ID

com.sun.identity.agents.config.user.token (Hot Swap enabled: No)

This property specifies a session property name which contains the user ID of the authenticated user in session.

This property is only used when the following properties are set as shown:

com.sun.identity.agents.config.user.mapping.mode = USER_ID

com.sun.identity.agents.config.user.principal = false

- Configuration Reload Interval Property (Hot Swap enabled: Yes)

  com.sun.identity.agents.config.load.interval

  This property specifies the interval in seconds between configuration reloads. When this property is set to 0, the hot-swap mechanism is disabled.

- Audit Log Properties

  com.sun.identity.agents.config.audit.accesstype (Hot Swap enabled: No)

  This property specifies the access type logged by the agent. The following are valid values for this property:

1. LOG_NONE (No access logging)
2. LOG_ALLOW (Log allowed access)
3. LOG_DENY (Log denied access)
4. LOG_BOTH (Log both allowed and denied accesses)

com.sun.identity.agents.config.log.disposition (Hot Swap enabled: Yes)

This property specifies the audit log mode that the agent uses when writing audit log messages. The following are valid values for this property:

1. LOCAL (Log file resides on the machine where the agent is installed)
2. REMOTE (Log file resides on the machine where the OpenSSO is installed)
3. ALL (Log files reside on both agent and OpenSSO machines)

com.sun.identity.agents.config.remote.logfile (Hot Swap enabled: Yes)

This property specifies the log file name on the remote server.

com.sun.identity.agents.config.local.logfile (Hot Swap enabled: Yes)

This property specifies the log file name on the local server.

com.sun.identity.agents.config.local.log.rotate (Hot Swap enabled: Yes)

This property is a flag that indicates whether the log rotation is enabled.

com.sun.identity.agents.config.local.log.size (Hot Swap enabled: Yes)

This property specifies the size of the local log file, beyond which the agent rotates the log file.

- Access Denied URI Property

  com.sun.identity.agents.config.access.denied.uri (Hot-swap enabled: Yes)

  This property specifies the URI used by the agent to block unauthorized access requests. If you will not employ this property, leave it blank.

- Login URL Property

  com.sun.identity.agents.config.login.url[] (Hot-swap enabled: Yes)

  This property is a list construct for listing the login URL (one or more) to be used by the agent to redirect incoming users without sufficient credentials to the OpenSSO authentication service.

- Cookie Reset Processing Properties

  com.sun.identity.agents.config.cookie.reset.enable (Hot-swap enabled: Yes)

  This property is a flag that specifies whether cookie reset processing is enabled.

  com.sun.identity.agents.config.cookie.reset.name[] (Hot-swap enabled: Yes)

  This property is a list construct for listing cookie names that are reset by the agent.

  com.sun.identity.agents.config.cookie.reset.domain[] (Hot-swap enabled: Yes)

  This property is a map construct. The key for this map construct is a cookie name and the value for this map construct is the domain of that cookie.

  com.sun.identity.agents.config.cookie.reset.path[] (Hot-swap enabled: Yes)

This property is a map construct. The key for this map construct is a cookie name and the value for this map construct is the path of that cookie.

- Not-Enforced URI Processing Properties

  com.sun.identity.agents.config.notenforced.uri[] (Hot-swap enabled: Yes)

  This property is a list construct for listing URI for which protection is not enforced by the agent.

  com.sun.identity.agents.config.notenforced.uri.invert (Hot-swap enabled: Yes)

  This property is a flag that specifies whether to invert the list of URI on the not-enforced list. A value of true directs the agent to deny access (enforce protection) to URI on the list and to allow access (not enforce protection) to URI that are not on the list. Entries on this list can contain the wild card character "*". This property enforces URI on the not-enforced list, which is the list assigned to the following property:

  com.sun.identity.agents.config.notenforced.uri[]

  com.sun.identity.agents.config.notenforced.uri.cache.enable (Hot-swap enabled: Yes)

  This property is a flag that specifies whether the caching of the not-enforced URI list evaluation results is enabled or disabled.

  com.sun.identity.agents.config.notenforced.uri.cache.size (Hot-swap enabled: Yes)

  This property specifies the size of the cache to be used if caching of not-enforced URI list evaluation results is enabled. This property is only used when the following property is set as shown:

  com.sun.identity.agents.config.notenforced.uri.cache.enable = true

- Not-Enforced Client IP Processing Properties

  com.sun.identity.agents.config.notenforced.ip[] (Hot-swap enabled: Yes)

  This property is a list construct for listing client IP addresses for which protection is not enforced by the agent.

  com.sun.identity.agents.config.notenforced.ip.invert (Hot-swap enabled: Yes)

  This property is a flag that specifies whether to invert the not-enforced client IP address list. A value of true directs the agent to deny access (enforce protection) to client IP addresses on the list and to allow access (not enforce protection) for all other client IP addresses. Entries on this list can contain the wild card character "*". This property enforces URI on the not-enforced IP

list, which is the list assigned to the following property:

com.sun.identity.agents.config.notenforced.ip[]

com.sun.identity.agents.config.notenforced.ip.cache.enable (Hot-swap enabled: Yes)

A flag that specifies whether the caching of not-enforced IP list evaluation results is enabled.

com.sun.identity.agents.config.notenforced.ip.cache.size (Hot-swap enabled: Yes)

This property specifies the size of the cache to be used if caching of not-enforced IP list evaluation results is enabled. This property is only used when the following property is set as shown:

com.sun.identity.agents.config.notenforced.ip.cache.enable = true

- Profile Attribute Processing Properties

  com.sun.identity.agents.config.profile.attribute.fetch.mode (Hot-swap enabled: Yes)

  This property specifies the mode used to fetch profile attributes. The following are valid values for this property:

1. NONE
2. HTTP_HEADER
3. REQUEST_ATTRIBUTE
4. HTTP_COOKIE

com.sun.identity.agents.config.profile.attribute.mapping[] (Hot-swap enabled: Yes)

This property is a map construct that specifies the profile attributes populated under specific names for the currently authenticated user. The key for this map construct is the profile attribute name and the value is the name under which that attribute is made available.

- Session Attribute Processing Properties

  com.sun.identity.agents.config.session.attribute.fetch.mode (Hot-swap enabled: Yes)

  This property specifies the mode used to fetch session attributes. The following are

  valid values for this property:

1. NONE
2. HTTP_HEADER
3. REQUEST_ATTRIBUTE
4. HTTP_COOKIE

com.sun.identity.agents.config.session.attribute.mapping[] (Hot-swap enabled: Yes)

This property is a map construct that specifies the session attributes populated under specific names for the currently authenticated user. The key for this map construct is the session attribute name and the value is the name under which that attribute is made available.

- Response Attribute Processing Properties

  com.sun.identity.agents.config.response.attribute.fetch.mode (Hot-swap enabled: Yes)

  This property specifies the mode used to fetch policy response attributes. The following are valid values for this property:

1. NONE
2. HTTP_HEADER
3. REQUEST_ATTRIBUTE
4. HTTP_COOKIE

  com.sun.identity.agents.config.response.attribute.mapping[] (Hot-swap enabled: Yes)

  This property is a map construct that specifies the policy response attributes to be populated under specific names for the currently authenticated user. The key for this map construct is the policy response attribute name and the value is the name under which that attribute is made available.

- Privileged Attribute Processing Properties

  com.sun.identity.agents.config.default.privileged.attribute[] (Hot-swap enabled: No)

  This property is a list construct for listing privileged attributes to be granted to all users who have a valid OpenSSO session.

  com.sun.identity.agents.config.privileged.attribute.type[] (Hot-swap enabled: No)

  This property is a list construct for listing privileged attribute types to be fetched for each user.

  com.sun.identity.agents.config.privileged.attribute.tolowercase[] (Hot-swap enabled: No)

  This property is a map construct that specifies whether the privileged attribute types are converted to lowercase. This property converts the attribute types assigned to the following property to lower case:

  com.sun.identity.agents.config.privileged.attribute.type[]

com.sun.identity.agents.config.privileged.session.attribute[] (Hot-swap enabled: No)

This property is a list construct for listing session property names that hold privileged attributes for the authenticated user.

- Debug Service Property

  com.iplanet.services.debug.level (Hot-swap enabled: No)

  This property specifies the debug level to be used. The following are valid values for this property:

1. off
2. error
3. warning
4. message

com.iplanet.services.debug.directory (Hot-swap enabled: No)

This property specifies the complete path to the directory where debug files are to be stored by the agent.

- Authentication Service Properties

  com.iplanet.am.server.protocol (Hot-swap enabled: No)

  This property specifies the protocol to be used by Authentication Service.

  com.iplanet.am.server.host (Hot-swap enabled: No)

  This property specifies the host to be used by Authentication Service.

  com.iplanet.am.server.port (Hot-swap enabled: No)

  This property specifies the port to be used by Authentication Service.

- Policy Client Properties

  com.sun.identity.agents.server.log.file.name (Hot-swap enabled: No)

  This property specifies the name of the log file for logging messages to OpenSSO.

  com.sun.identity.agents.logging.level (Hot-swap enabled: No)

This property specifies the level of remote policy logging. The following are valid values for this property:

1. ALLOW
2. DENY
3. BOTH
4. NONE

com.sun.identity.agents.notification.enabled (Hot-swap enabled: No)

This property is a flag that specifies whether notifications are enabled or disabled for the remote policy client.

com.sun.identity.agents.notification.url (Hot-swap enabled: No)

This property specifies the notification URL for the remote policy client. This property is used if notification is enabled for a remote policy client property, which occurs when the following property is set as shown:

com.sun.identity.agents.notification.enabled = true

com.sun.identity.agents.polling.interval (Hot-swap enabled: No)

This property specifies the duration in minutes after which the cached entries are refreshed by the remote policy client.

com.sun.identity.policy.client.cacheMode (Hot-swap enabled: No)

This property specifies the mode of caching to be used by the remote policy client. The following are valid values for this property:

1. subtree
2. self

The subtree value is preferable for a small number of policy rules. In all other cases, the self value is preferable.

com.sun.identity.policy.client.booleanActionValues (Hot-swap enabled: No)

This property specifies boolean action values for policy action names. Assign values to this property using the following format:

serviceName|actionName|trueValue|falseValue

com.sun.identity.policy.client.resourceComparators (Hot-swap enabled: No)

This property specifies resource comparators to be used for different service names.

com.sun.identity.policy.client.clockSkew (Hot-swap enabled: No)

This property specifies the time in seconds which is allowed to accommodate the time difference between the OpenSSO machine and the remote policy client machine.

5 Conclusion

This document described the OpenSSO J2EE agent architecture in some detail. Readers planning to implement a J2EE agent should join the OpenSSO project (https://opensso.dev.java.net/), subscribe to the dev@opensso.dev.java.net mailing list and examine the existing OpenSSO J2EE agent source code.