Session Service Architecture

Open Web Single Sign-On

Version 1.0

Please send comments to: opensso@sun.com

Author **Alan Chu**
(alan.chu@sun.com)

This document is subject to the following license:

Contents

1 Introduction

1.1 Document Status

Project Name	Open Web Single Sign-On
Document Title	Session Service Architecture
Date of Issue	October 31, 2005
Current Version	1.0
Author	Alan Chu (alan.chu@sun.com)
Issuing Organization	Sun Microsystems, Inc.
Feedback E-mail	opensso@sun.com

1.2 Revision History

Date	Version	Author	Comments
October 31, 2005	1.0	Alan Chu (alan.chu@sun.com)	Initial Revision

1.3 Summary

Session Service is one of the core components in the Single Sign-On (SSO) infrastructure. Its function is to provide facilities for keeping information about authenticated user sessions across a number of applications participating in the same SSO environment. The Session Service also tracks a user's interactions with web applications and provides the fundamental administration and monitoring capabilities for managing user sessions.

This purpose of this document is to provide the architectural details for the implementation of the Session Service 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 essentially describes the detailed architecture of the Session Service framework in the OpenSSO project. It is not a requirements 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 Session Service framework as well as to serve as an effective vehicle for facilitating any design and implementation practice pertaining to the Session Service framework.

1.5 Context

Session Service 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 the system, and enforces time-out limits when necessary.

Generally speaking, the Session Service should perform some or all of the following actions:

- Generates session identifiers.
- Maintains session state information.
- Implements time-dependent behavior of sessions.
- Implements session life cycle events (e.g. session creation, session destruction, etc.).
- Generates session life cycle event or other state change notifications so that all participants in the same SSO environment can get notified.
- Enables Single Sign-On (SSO) among web applications.
- Allows participating clients to share information across the deployments.
- Provides session management interfaces.
- Implements high availability facilities.

A user session is the interval between the moment a user logs in to the OpenSSO system, and the moment the user logs out of the system. Using a concrete use case example, in a typical user session an employee attempts to access the corporate benefits administration application. The application is protected by an SSO Agent, and the OpenSSO system prompts the user for a user name and a password. First, the Authentication Service verifies that the user is a valid user. The OpenSSO system then allows the user access to the application.

In the same user session (without logging out of the corporate benefits application), the same employee attempts to access the corporate expense reporting application. The expense reporting application is also protected by an SSO Agent. In this second transaction, the Session Service provides continued proof of the user's authentication, and the employee is automatically allowed to access the expense reporting application. The employee has accessed more than one service in a single user session without having to re-authenticate. This functionality is called Single Sign-On (SSO).

SSO is always preceded by a basic user session in which a session is created and its session token is validated, and in which the user is authenticated. SSO begins to occur when the authenticated user requests a protected resource on a second server in the same domain. Session Service maintains user session information with input from all applications participating in the single sign-on.

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*.

Client — An entity that accesses a *service* within the *system*.

Client Library — A specialized component that provides programmatic access to a set of *services* within the *system* by acting as a *client* on behalf of the subsystem that uses it. A client library abstracts the underlying communication and other implementation details necessary to efficiently access the service from within the system.

Cluster — A *system* where more than one installation of OpenSSO *services* are available which operate together as a single logical installation.

Cookie — A mechanism that allows a web server to store some data on the browser that accesses that server.

Domain	A suffix used in fully qualified host names that allows the logical grouping of hosts.
Firewall	An entity that limits access to and from a network based on the configured security policies.
HTTP	Acronym for Hypertext Transfer Protocol. This is an open standards based protocol used for exchange of information between web browsers and web servers.
OpenSSO	Alias 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.
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 Client	A *client library* for *session service*.
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.
SSO Agent	A minimally intrusive transparent software component that can be added to the access path of a *web application* to allow it to participate in *SSO*.
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.

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

2 Objectives

2.1 Mission

The mission of Session Service is to provide the functionality to maintain information about an authenticated user's session across all web applications participating in a Single Sign-On environment. The focus in the Session Service is to establish the fundamental SSO infrastructure by serving a number of critical functions which enable users to authenticate once yet access multiple resources such that successive attempts by a user to access protected resources will not require the user to provide authentication credentials for each attempt.

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 Session Service are:

- ***Developers***: Responsible for the overall development of the system. They may be involved in various development related activities associated with the Session Service such as designing, developing, building, testing, documenting, and troubleshooting the functionality provided by the Session Service.

- ***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.

- ***Web Application Developers***: Developers who are responsible for the creation and deployment of web applications on the network. These developers may use the Session Client and other public interfaces in order to make their applications participate in the SSO environment and thus enhance its overall functionality.

- ***Web Application Administrators***: Administrators who are chartered with the deployment of web applications. These administrators will configure Session Client or OpenSSO agents as necessary in order to ensure that the hosted web applications can take advantage of the available session services.

- ***System Integrators***: Developers who are chartered with the deployment of OpenSSO in a

given environment to best utilize the Session Service. They may be involved in building new SSO agents and using the exposed public SSO interfaces or Session Client where necessary.

2.3 Architectural Concerns

There are quite a few architectural concerns which are identified and considered in formulating the architectural concept of the Session Service from the perspectives of all relevant stakeholders. While the system wide concerns are already addressed in the OpenSSO System Architecture Document [1], this section intends to focus on the core concerns which are derived specifically from the system requirements for the Session Service.

2.3.1 Pluggability of SSO Providers

The Session Service infrastructure should allow the creation and integration of customized SSO provider implementations which can be plugged into the framework to enhance the SSO functionality. These SSO providers can also offer the specialized services tailored to meet different system requirements but have to be able to integrate with other services in the OpenSSO system.

2.3.2 Session Life Cycle Management

The Session Service must implement the mechanisms to manage the session state changes which can be triggered by either user's actions (e.g. login, logout) or the system events such as cleanup for expired sessions. These session state changes usually depend on the time-dependent behaviors, which may include creation, activation, timing out, and destruction of user sessions.

2.3.3 Remote Access to Session Service

The Session Service should offer the client libraries by which the user sessions can be validated, updated, and destroyed remotely. These libraries or interfaces provided by the Session Service should also take care of all the complexities of any underlying transport mechanism, which should be transparent to the Clients.

2.3.4 Generation of Session Event Notification

The Session Service framework should provide the notification service which allows for session event notifications to be sent to Session Clients participating in the same SSO environment. These session events may include the session state changes (e.g. session creation, session timeout, and session termination) as well as the occurrences of session property modifications.

2.3.5 Session Administration and Monitoring

The Session Service should provide the mechanisms to manage the user sessions. These

administration capabilities may include the retrieval of some or all of the user sessions currently hosted by the system, the ability to terminate the user sessions on demand, and the enforcement to limit the maximum number of current sessions for a user. Besides, the Session Service should also provide the facility to monitor the session related resource consumption.

2.3.6 High Availability and Service Continuation

The Session Service should provide a function of fault tolerance, which means the ability to retain the authenticated session state in case of a hardware or software failure of a single server. It should provide both the infrastructure redundancy (no single point of failure) as well as the state redundancy (protect against the loss of information for existing sessions after failure).

2.3.7 Security

The information stored in the user session should be protected by the Session Service in a secure fashion. Only the people who obtain the correct session identifier or the privileged administrators can have the access to the respective user session. The Session Service should also guarantee that no user session information is disclosed in the communication between the Session Service and other external components (e.g. persistent data store).

2.3.8 Performance and Scalability

Given the limitation of the capabilities of the underlying hardware and software components, the Session Service should perform and scale up to the necessary levels in order to accommodate more user sessions.

3 Architectural Views

3.1 SSO Interface View

3.1.1 Viewpoint Specification

Name	SSO Interface Viewpoint
Stakeholders	Developers, Web Application Developers, System Integrators
Concerns	Pluggability of SSO Providers
Modeled As	Interface
Viewpoint Source	System Requirements

3.1.2 Detail

Interface Summary
Retrieve a session
Validate a session
Refresh a session state
Update a session
Register a session listener
Destroy a session
Retrieve all valid sessions from a OpenSSO system

3.1.3 Description

In the Single Sign-On environment, all of the clients participating in the SSO require a handle to a valid session to be able to perform actions and interact with other components within the system. A set of interfaces should be defined to allow these clients to manipulate the sessions and their resulting behaviors. This view describes these interfaces by which external applications can use to participate in the SSO environment with the given session handle and to retrieve, validate, refresh, modify, or destroy a user session.

3.1.4 Extension

1. This view can be extended to elaborate the process of how a valid session was originally created via the the authentication process.

3.2 Session State Transition View

3.2.1 Viewpoint Specification

Name	Session State Transition Viewpoint
Stakeholders	All
Concerns	Session Life Cycle Management, Remote Access to Session Service, Generation of Session Event Notification
Modeled As	Collaboration
Viewpoint Source	System Requirements

3.2.2 Detail

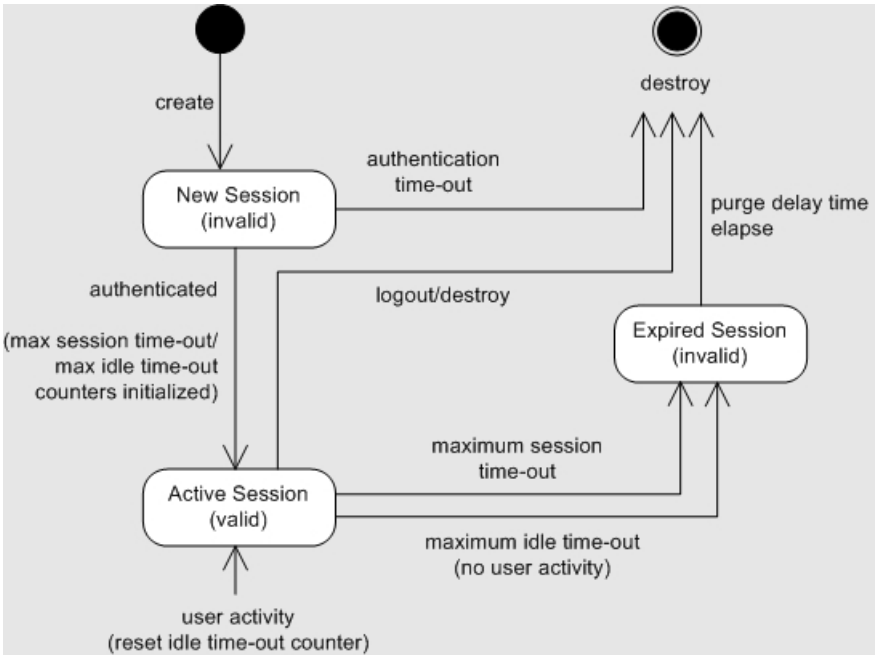

Figure 1: Transitions of Session States

3.2.3 Description

The transitions of session states are controlled by the Session Service based on the user actions or the time-dependent behaviors which are enforced by the Session Service. When a request to create a new user session is triggered, the Session Service generates a new session ID and a new instance of session object. The session object is initially created in the INVALID state with default values of attributes and empty property list. Newly created sessions do not carry any identity. Once the session is authenticated, session state is updated with all the identity attributes and properties.

3.2.4 Extension

1. This view does not describe certain conditions where a session may not be activated (changing the state from INVALID to VALID) even though the authentication process is complete. These special situations occur when the Session Service enforces the checking of either user-based or system-based constraints such as limitation on the maximum number of concurrent sessions allowed for a user or for the entire system.

3.3 Session Client View

3.3.1 Viewpoint Specification

Name	Session Client Viewpoint
Stakeholders	Developers, Web Application Developers, System Integrators
Concerns	Session Life Cycle Management, Remote Access to Session Service, Generation of Session Event Notification
Modeled As	Collaboration
Viewpoint Source	System Requirements

3.3.2 Detail

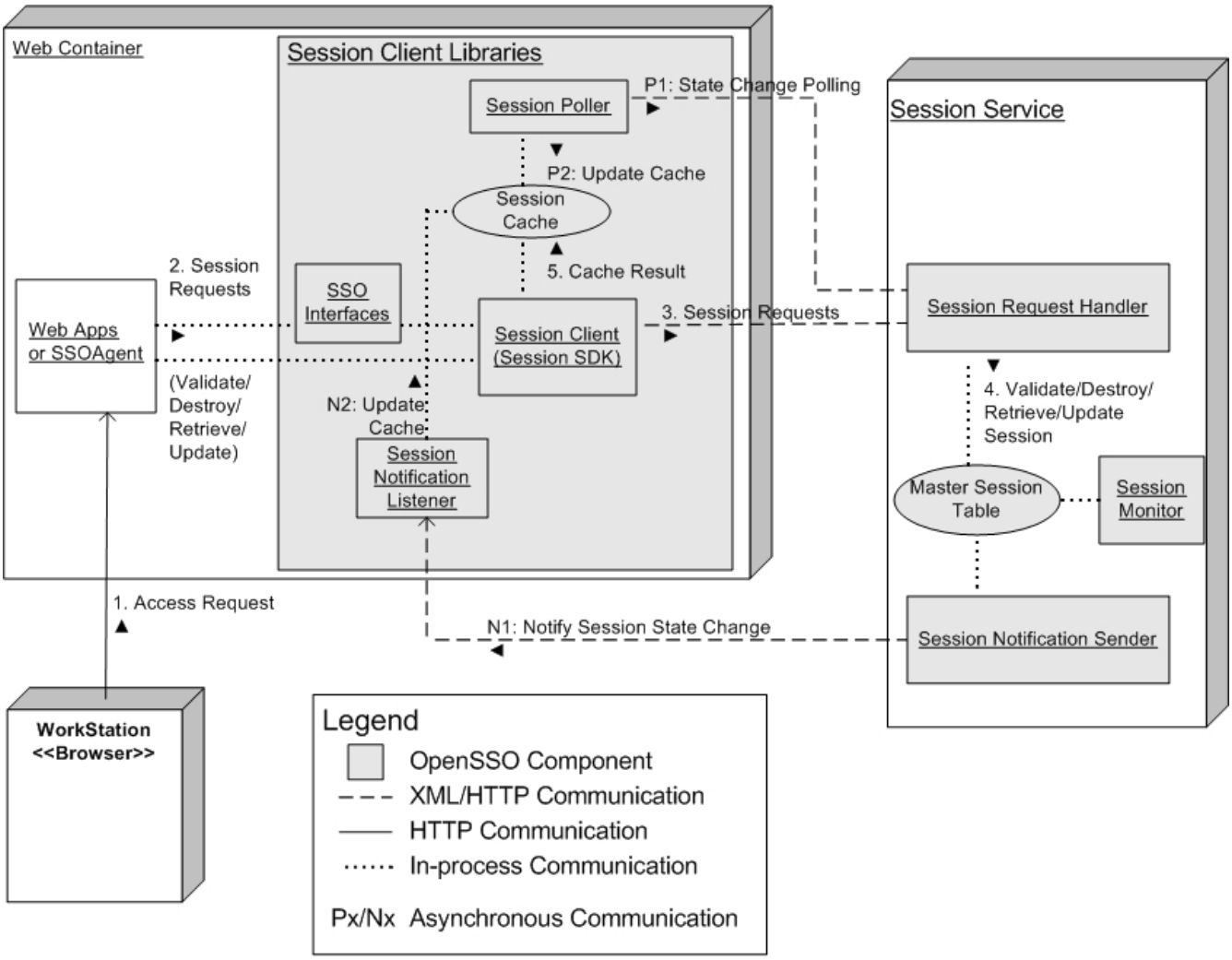

Figure 2: Session Client View

3.3.3 Description

This view illustrates the interactions between the Session Client and the Session Service when the session requests are issued from the client end point. All the requests associated with the same session share an unique session identifier. This session identifier is passed from the browser to the SSO enabled web applications or SSO Agents which then convey the request to the Session Service. Depending on the actual deployment scenarios, the client web applications and the SSO Agents can communicate with the Session Service through the Session Client libraries (also known as Session SDK) directly or via the public SSO interfaces which in turn will invoke the Session Client to submit the requests to the Session Service.

The session state maintained on the client side is essentially a cached view of the actual session object, and this cache can be updated by either the active polling mechanism or the session notification triggered by the Session Service.

3.3.4 Extension

This view can be extended to include the scenario where the Session Client can communicate to the Session Service on different OpenSSO systems based on which server the respective user session belongs to.

3.4 High Availability Deployment View

3.4.1 Viewpoint Specification

Name	High Availability Deployment View
Stakeholders	Administrators, Web Application Administrator, System Integrators
Concerns	High Availability and Service Continuation, Security, Performance and Scalability
Modeled As	Deployment
Viewpoint Source	System Requirements

3.4.2 Detail

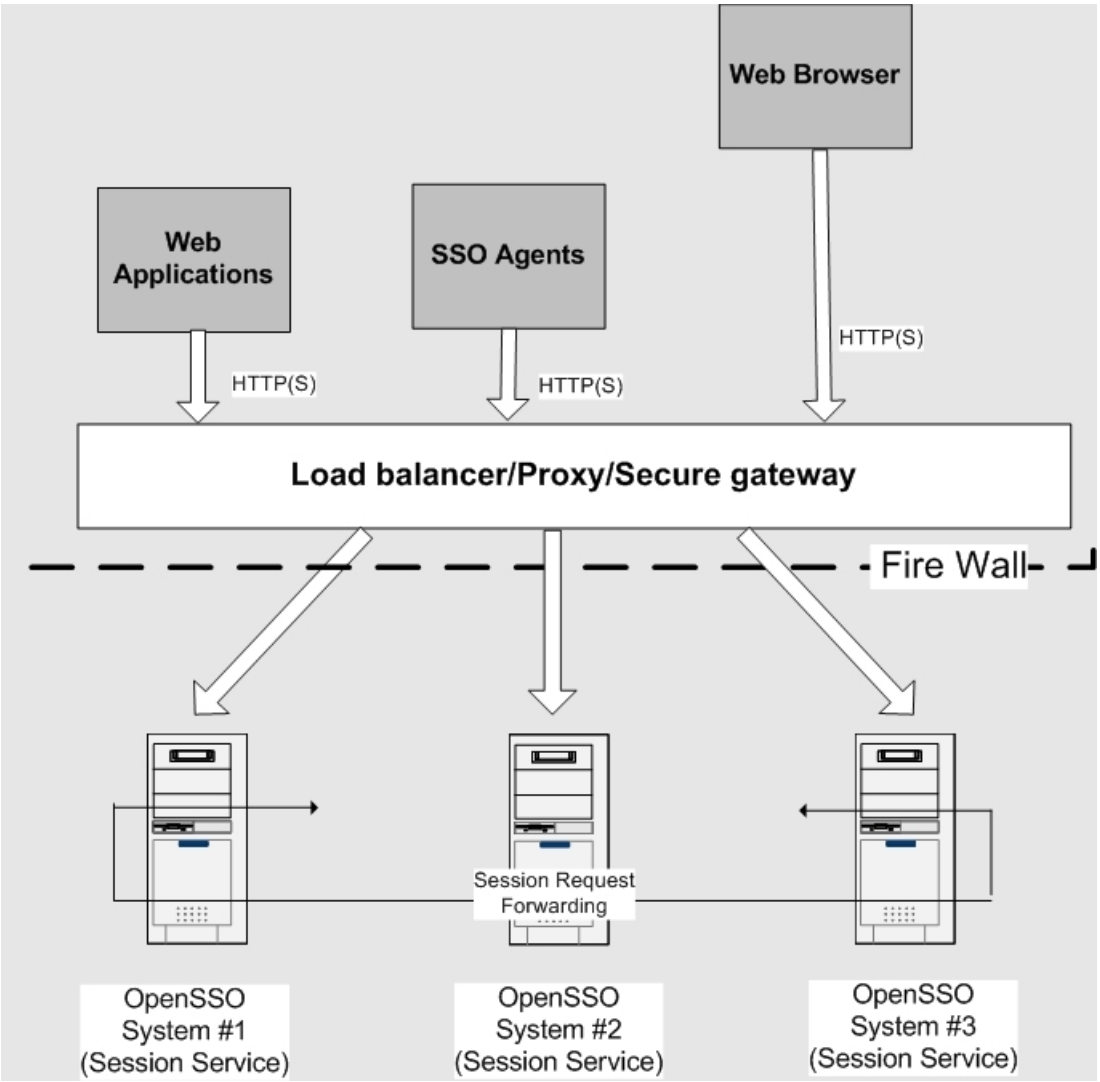

Figure 3: High Availability Deployment View

3.4.3 Description

A cluster deployment of Session Service is mandated when the high system availability and the service continuation need to be guaranteed. This view illustrates the mechanism used by the Session Service to process the session requests in the cluster environment. Since session requests may not always be distributed to the system where the respective session was originally created, the Session Service applies the request forwarding logic to redirect the request to the correct system based on the unique and self-contained session identifier.

3.4.4 Extension

This view can be further extended to address the requirements of infrastructure and session state redundancy with the fault tolerance capabilities.

3.5 Session Failover Deployment View

3.5.1 Viewpoint Specification

Name	Session Failover Deployment View
Stakeholders	Administrators, Web Application Administrator, System Integrators
Concerns	High Availability and Service Continuation, Security, Performance, and Scalability
Modeled As	Deployment
Viewpoint Source	Derived from High Availability View and System Requirements

3.5.2 Detail

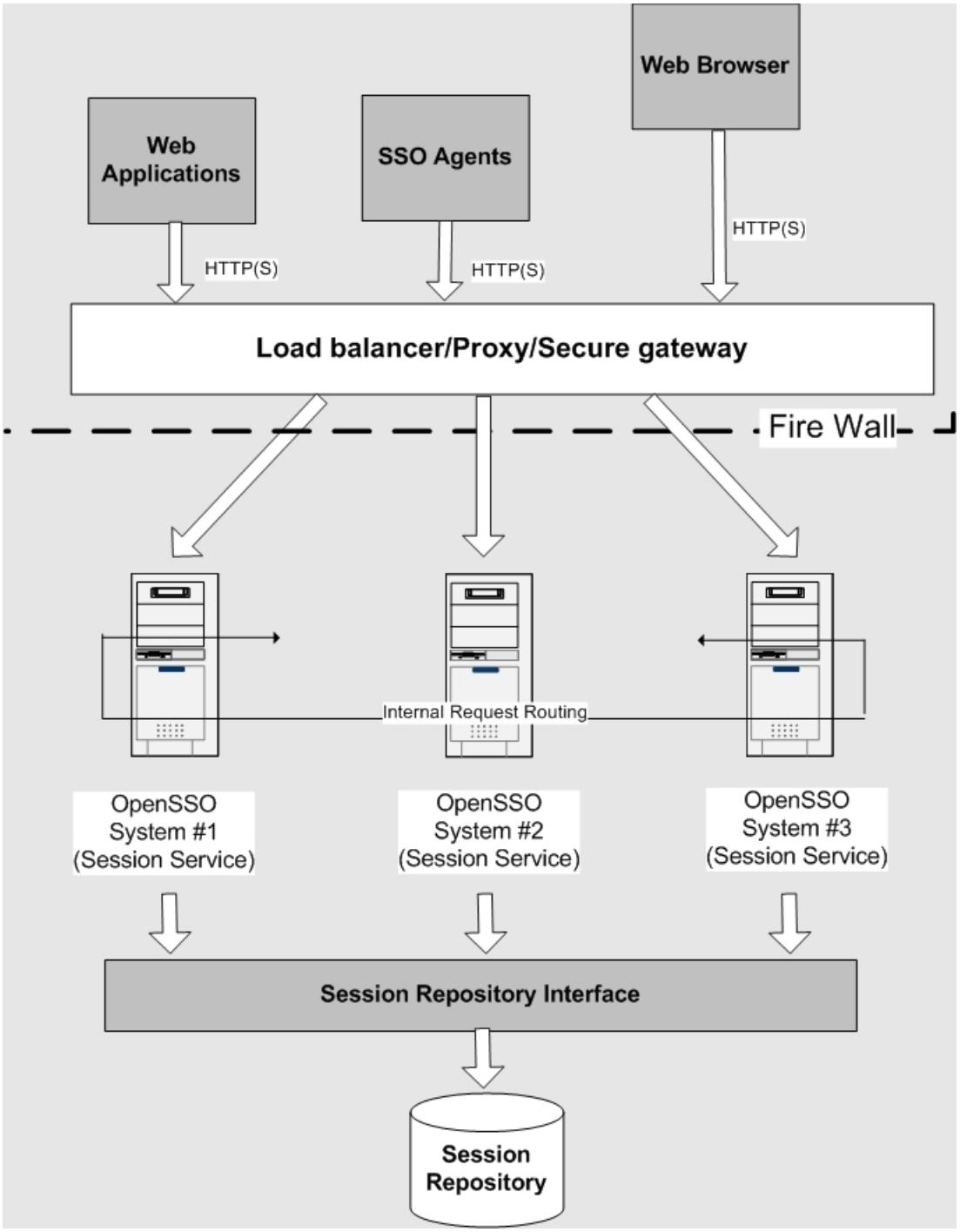

Figure 4: Session Failover Deployment View

3.5.3 Description

The session failover deployment view extends the high availability view and addresses the requirement of system fault tolerance, which is essentially a backup operation that automatically switches to a standby system if the primary system fails or is temporarily shut down for service. This view illustrates the mechanisms by which the Session Service can retain the authenticated session states in case of a hardware or software failure of a single server and continue processing new client requests after the failure.

The Session Service leverages the persistent repository to keep the session state which can be retrieved and recovered when the failover activities occur. The internal request routing logic is performed to select the backup system based on a deterministic permutation generation algorithm.

4 Session Service Conceptual Implementation

In the OpenSSO project, the SSO interfaces and the resulting Session Service framework as a whole can be implemented with different approaches and using various methods. The end goals here are to address the overall system requirements as described in the OpenSSO system architecture and adhere to the viewpoint specifications as detailed within this Session Service architecture document.

This chapter presents an overview of the conceptual implementation of the Session Service infrastructure with certain level of details on all the sub-components and services.

4.1 Session Service Software Implementation

The main function of the Session Service framework is to provide facilities for keeping information about authenticated user session across a number of applications participating in the SSO environment.

4.1.1 System Architecture and SSO Interfaces

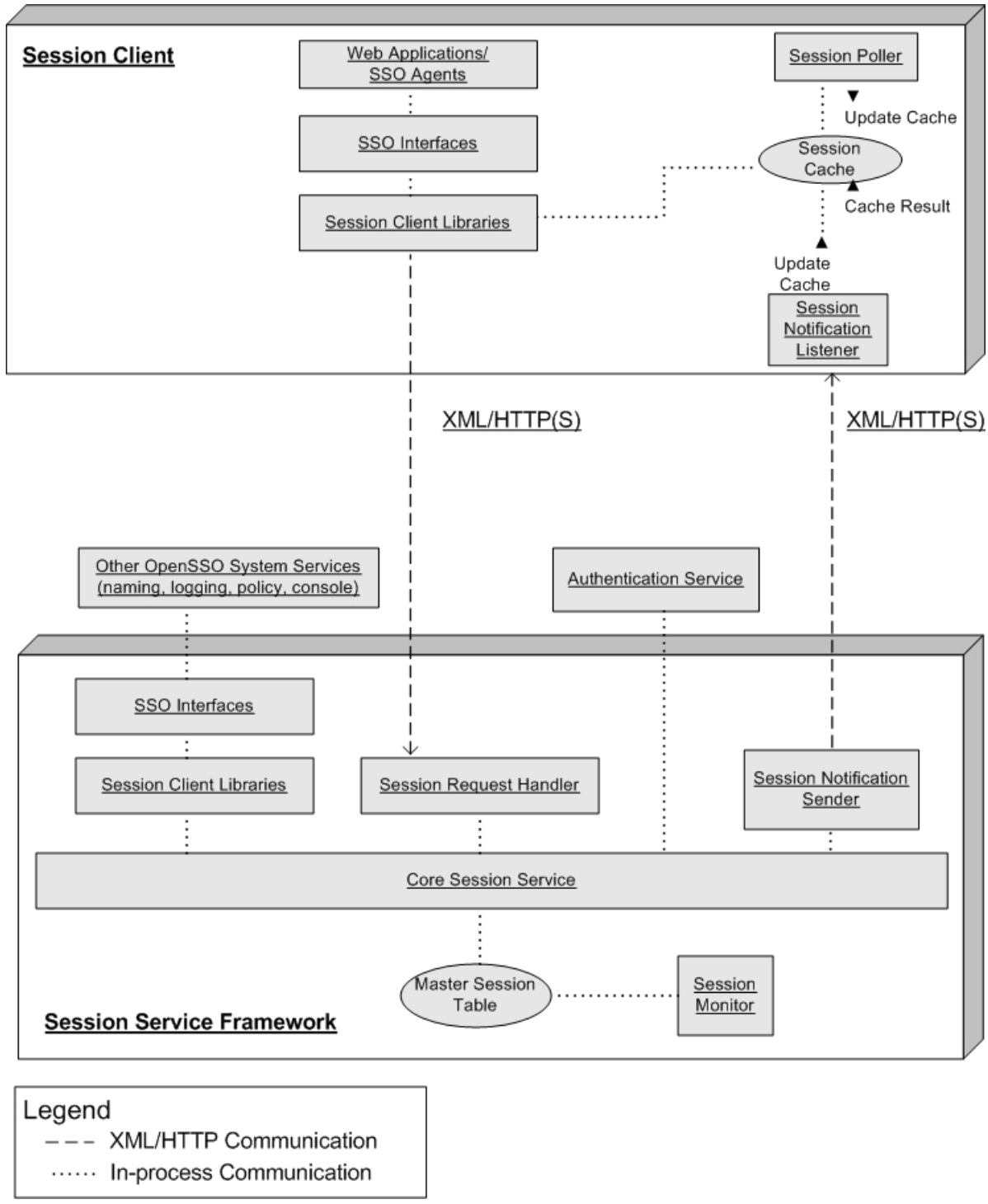

Figure 5: Session Service Software Conceptual Implementation

Interface Summary	
SSOToken	The `SSOToken` class represents an SSO token.
SSOTokenEvent	The `SSOTokenEvent` class represents an SSO token event.
SSOTokenID	The `SSOTokenID` class is used to identify an SSO token object.
SSOTokenListener	The `SSOTokenListener` interface needs to be implemented by the applications to receive SSO token events.

Class Summary	
SSOTokenManager	The class `SSOTokenManager` is a `final` class that provides interfaces to create and validate `SSOTokens`.

Exception Summary	
SSOException	An `SSOException` is thrown when there are errors related to SSO token operations.

Table 1: Public SSO Interfaces

The SSO interface view describes the public interfaces which external applications can use to participate in the SSO environment. These public interfaces are implemented by the chosen SSO provider which gets plugged into the OpenSSO system to provide the necessary functionality required for the Single Sign-On to work.

All of the services (except for the Authentication Service) in the OpenSSO system require a valid session (programmatically referred to as SSOToken) to process an HTTP request. External applications wishing to use the SSO functionality must also use the SSOToken to authenticate the user's identity. With the SSO APIs, an external application can retrieve it and, in turn, the user's identity, session, and other information. The application then uses this information to determine whether to provide user access to a protected resource.

The main classes of the SSO APIs defined in OpenSSO system are SSOTokenManager, SSOToken, and SSOTokenListener. The SSOTokenManager class is used to get, destroy, validate, and refresh a session token which is represented by the SSOToken class. The SSOTokenListener class allows the application to be notified when a SSOToken has become invalid (for example when a session has timed out).

4.1.2 Session Client

The SSO API layer is implemented by classes from the Java package `com.iplanet.sso`. In the case of a remote client, the invocation is forwarded by Session Client libraries using XML messages over HTTP(S). The same Java implementation for Session Client is used both by the remote Java clients and the Session Service framework.

Session Client APIs contain classes from the Java package `com.iplanet.dpro.session`. The Session Client API layer is responsible for:

- resolving session ID to session state information by figuring out location of the session service and retrieving session information from it
- caching session information and periodically refreshing it (either via polling or session notifications)
- forwarding session requests to the Session Service using either local or remote invocation.

4.1.3 Session Service Framework

The Session Service framework implementation contains classes from the Java package `com.iplanet.dpro.session.service`. It is mainly responsible for:

- generating session IDs
- maintaining master copy of session state information
- implementing time-dependent behavior of sessions
- implementing life cycle events in session (e.g., logout/destruction)
- generating session event notifications
- implementing session failover and other security facilities

The Authentication Service queries and updates the session state by calling methods on Session Service classes directly. Other modules interact with the session state indirectly via the SSO interfaces or the Session Client libraries (Session SDK).

4.1.4 Session Request Handling

The Session Request Handler implements the processing of Session Service methods over the remote interface. It is implemented by Java class `com.iplanet.dpro.session.service.SessionRequestHandler`. The client requests can be originated by the web applications/SSO Agents or can be automatically triggered by the Session Poller built in the Session Client.

4.1.5 Session Notification Sender

The Session Notification Sender is responsible for sending the session notification to all the

registered clients when there is a state or property change occurring in the master session object. It is implemented as an inner class within the core Session Service class `com.iplanet.dpro.session.service.SessionService`.

4.1.6 Session Monitor

The Session Monitor uses a background thread which checks locally hosted sessions to find ones which have expired due to the maximum or idle session time limits. Those expired sessions will be removed from the master session table after the purge delay time has elapsed (as illustrated in Figure 1). The design idea behind the purge delay is to improve the overall user experience by keeping the timeout related information for the expired sessions.

The system launched time-based transitions in session state are enforced by this background thread executing the specialized cleanup logic. It periodically sweeps through the entire session table checking for sessions whose maximum or idle time is exceeded. Authentication framework is responsible for destroying sessions which were created but never moved to the VALID state before the maximum login timeout expired.

4.2 Session State Structure

4.2.1 Session ID

The session ID (or SSOToken ID) is basically an opaque handle used to locate the session object. In order to accommodate the requirements for advanced features such as the internal request routing scheme implemented for session failover, the session ID format is designed to allow utmost capacity for future extensibility. The format has the following structure:

<opaque core session prefix>@<session extension fields>#<http session id>

<opaque is core session prefix> is simply an encrypted ID string. <session extension fields> is base64-encoded sequence of pairs (field tag, field value). The existing extension field tags include the primary server instance ID and session storage key but the structure allows the inclusion of any new fields if necessary. <http session id> is appended for reference only and is not used by the Session Service.

4.2.2 Session Attributes and Properties

Session state contains a number of fixed attributes and an extensible set of properties. The existent session attributes include the following:

- UUID (universal unique identifier)

- Client Domain
- Client ID (set to user DN or application principal name)
- Type (USER or APPLICATION)
- Session State
- Max Idle Time
- Max Session Time
- Max Caching Time
- Latest Access Time
- Session Creation Time

Session properties are divided into two subsets: protected (or core) properties and custom properties. Protected properties are only modifiable by the server-side modules (primarily the Authentication Service). Custom properties are modifiable remotely by any application which possesses the session ID.

The session property implementation can actually be extended to provide the private property scope for each of the clients participating in the SSO environment. This is to address the requirement of having an independent property space for each client so that it can store and retrieve its own session properties without any interference from other clients sharing the same user session.

It is important to understand that values of the most of the attributes and properties are controlled by the modules outside of the Session Service framework (primarily determined by the Authentication Service). The Session Service framework takes on the passive role of providing the storage for session information and enforcing some of the time-dependent behavior (e.g., invalidating and destroying sessions which exceeded their maximum Idle time or maximum session time).

4.3 Authentication and Authorization

In the Session Service framework, the authentication of the session owner is ultimately provided by demonstrating knowledge of the hard-to-guess session ID. The Session Service is at the heart of the session id management and provides secure association between session id and other information describing the owner of the session in tandem with the Authentication Service.

Some of the Session Service framework interfaces expect the session ID or a direct reference to the

InternalSession object as a parameter. These interfaces are very sensitive as they allow the local code to manipulate with the session state directly. Authentication for remote Session Service method relies on client supplying a valid session ID of the requester. There are two groups of methods. The first group uses session ID as both the authentication credential and the argument of the operation. This group includes:

- GetSession
- Logout
- AddSessionListener
- SetProperty

The second group uses a separate session ID as the requester's credential:

- GetValidSessions
- DestroySession
- AddSessionListenerOnAllSessions

Authorization checks used for remote session service methods are described in the following table.

Method	Authorization check
GetSession	Applies to requester's own session
Logout	Applies to requester's own session
AddSessionListener	Applies to requester's own session
SetProperty	Applies to requester's own session, but only non-core properties are modifiable. Attempts to change core properties will be rejected.
GetValidSessions	Returns all sessions if the requester is a top-level administrator, or nothing if the request is not.
DestroySession	Requester's own session is the same as one being destroyed, or requester is a top-level administrator.
AddSessionListenerOnAllSessions	Is allowed if requester is the same as super user specified by the system configuration file.

4.4 Cluster Technologies

The Session Service in the OpenSSO project introduces two mechanisms to address the requirements of the high system availability, service continuation, and ease of cluster environment management.

4.4.1 Site Configuration

Together with the Naming Service, the Session Service introduces the "site concept," which provides centralized configuration management for a cluster deployment. When the OpenSSO system is configured as a site, client requests always go through the load balancer, which simplifies the deployment as well as resolves issues such as a firewall between the client and the back-end OpenSSO systems (see Figure 3: High Availability Deployment View). A site includes the following components:

- Multiple (two or more) OpenSSO systems are deployed on at least two different host servers. For example, you might deploy two instances on one server and a third instance on another server. Or you might deploy all instances on different servers. You can also configure the deployment in the session failover mode (next section), if required for your deployment.
- One or more load balancers route client requests to the various OpenSSO systems. Each load balancer can be configured according to the deployment requirements (for example, to use round-robin or load average) to distribute the load between the OpenSSO system instances.
- All the OpenSSO systems share the same platform profile and can access the same configuration repository.

4.4.2 Session Failover and Internal Request Routing Scheme

4.4.2.1 Overview

The Session Service introduces the functionality of session failover to satisfy the requirements of retaining the authenticated session states in case of a hardware or software failure of a single server and being able to continue processing new client requests after the failure. There are basically two fundamental aspects of the requirements:

- Infrastructure redundancy: The system must not have a single point of failure and is capable of achieving continuous service availability for new sessions.
- State redundancy: It is required to protect against the loss of information for the existing sessions.

The session failover implementation should also address the following architectural concerns:

- There should be no dependency on the web container on which the OpenSSO system is deployed.
- The abstraction of the session repository used to persist the session data should be ensured so that different types of persistent stores are allowed to be plugged into the Session Service.

4.4.2.2 Internal Request Routing Scheme

The Session Service introduces a basic routing scheme to determine the backup system when the session failover activities occur. Here are the details of this routing algorithm.

First, with the request routing scheme, each session is assigned a primary server instance and a parameter (called session storage key) which is used to produce a sequence of alternative server instances used in the case of failure. The primary server instance ID and the session storage key are chosen at the time of the session creation and recorded as part of the session ID.

The primary server ID for a session is the ID of the server instance which created the session. This server instance will be the one hosting the session state normally (in the absence of failures).

If the primary server instance hosting the session is down, the failover framework needs a method to determine the alternative server instance to host the session. This is accomplished by using the session storage key together with a permutation generator to deterministically derive the entire sequence of alternative server instances to be used in case previous server instances in the sequence are down. Instead of using the session storage key and the permutation generator, the entire failover sequence could have been recorded in the session ID. However, with the approach the size of the session ID would grow with the total number of server instances in the cluster. With the permutation generator-based method session ID size is fixed (albeit for the price of the insignificant computation overhead).

The failover sequence generator uses a pseudo-random number generator (PRNG) to produce a permutation of server IDs using the session storage key as a seed. The property of deterministic pseudo-random number generators is that given the same seed value, they produce the same pseudo random number sequence. The standard implementation of java.util.Random PRNG implementation falls into this category.

Given the session storage key (sKey) and the sorted array of server instance ids (serverList), random permutation can be produced by using the following Java code:

```
        random = new Random();

        random.setSeed(sKey.hashCode());
        for(int i = 0 ; i < serverList.length; ++ i) {
            // generate uniformly distributed number
            // between 0 and serverList.length - 1 (inclusive)

            int r = i + random.nextInt(serverList.length - i);

            // swap serverList[i] and serverList[r]

            String tmp = serverList[r];
            serverList[r] = serverList[i];
            serverList[i] = tmp;
        }
```

This permutation generation algorithm has the following properties:

- Given the same sessionId and serverList values, it will always generate the same permutation
- If sKey is a uniformly distributed random value, than permutations will also be uniformly distributed.

The first property guarantees that the sequence can be deterministically reproduced from just the session storage key and statically configured list of servers so that any server instance in the cluster performing the computation will get the same answer. The second property ensures that session load is spread evenly among surviving server instances.

The generated session storage key is an uniformly distributed 32-bit integer (a 16-bit integer should be adequate as well, if the total session ID size becomes an issue).

The routing logic which determines the current server instance hosting a given session can be described by the following pseudo-code:

```
if (isUp(session.primary_server_id)) {
      //use primary server instance
} else {
      generate server id permutation from session id
      foreach (candidateServer in permutation) {
            if (isUp(candidateServer)) {
                  // use candidateServer
            }
      }
}
```

Since the current server instance executing the pseudo-code is also part of the permutation (and also has the “up” status) the selection procedure is guaranteed to always return a server instance id.

This server instance selection logic is run every time the session framework code needs to decide which server instance is currently hosting the session state and direct processing of requests using that session to the corresponding server instance.

4.4.2.3 Session Repository Abstraction

The session failover functionality requires the persistent repository to store the states of user sessions. In order to preserve the maximum flexibility and extensibility of the Session Service framework, the system is designed to provide an abstraction of the session repository layer (`com.iplanet.dpro.session.service.AMSessionRepository` interface) so that different provider implementations can be plugged into the session service and perform the tasks.

JMS based implementation is the default provider of this session repository interface, but a JDBC 2.0 compliant repository (database) or any other persistent storage can also be used as the session store as long as the corresponding implementation of session repository interface is provided.

4.5 Session Quota Constraints

The enforcement of Session Quota Constraints allows an administrator to limit a user to a specific number of active/concurrent sessions based on either the constraint settings at the global level or

the configuration associated with the user. When there is an attempt to create a new user session, the checking of the session quota constraints is enforced, so that if the session quota for that particular login user has been exhausted, certain actions will be taken based on the desired behavior configured by the administrator. These actions include denying the login request, accommodating the new session creation request by destroying the oldest existing session, or applying customized implementation. Session Quota Constraints can be enforced in a single OpenSSO system deployment or in a cluster environment.

4.6 Security

Apart from the specific checking with regard to authentication and authorization as described in Section 4.3, there are several general security related practices introduced in the Session Service:

- SSL is strongly recommended to prevent the session ID from being stolen by passive network snooping.
- The encryption for session state serialization into the session repository is used to prevent other software applications sharing the same system resources from subverting it.
- The implementation of a restricted token (a handle to the master session id) generated for each client decreases the possibility of occurrence of the session hijacking issue.

5 Conclusion

The software architecture described in this document addresses the architectural concerns and provides the viewpoints with certain level of details with regard to the Session Service in the OpenSSO project. Although comparatively low level implementation details are also included in the context, the purpose of this document is not to provide the ultimate solution to the Session Service implementation but to serve as the generic guideline which may be used to facilitate any design and implementation practice pertaining to the Session Service framework.