Functional Specification

RMI-IIOP Load Balancing and Failover

AS 9.0 EE

Author(s): Harold Carr
email(s): harold.carr@Sun.COM
Status: [Internal Draft]

Change Log

Version	Comments	Date	Author
0.1	Initial draft (links are not up-to-date).	03/11/05	Harold Carr
0.2	Updated based on reviews by Larry White, Sreeram Duvur and Pankay Jairath. (some links to external documents not up-to-date).	04/07/04	Harold Carr

Table of Contents

1 Introduction

This document is the functional specification for the RMI-IIOP Load Balancing and Failover Subsystem (IIOP LB/FO) in AS 9.0EE.

The following IIOP High-Availability (HA) features are supported in 9.0 EE:

1. Load Balancing of IIOP requests in Dynamic Clusters.

2. HA of remote references in Dynamic Clusters. HA of remote references is supported for RMI-IIOP invocations from Standalone Clients and Application Client Container (ACC) Clients. This is ensured by detecting failed requests and redirecting these requests to another AS instance in the cluster.

IIOP LB/FO features on the CSIv2/SSL path may not be implemented depending on resources. The backgrounds of the affected SSL features are highlighted with yellow.

The primary new feature discussed in this document is the Client Reconfiguration Notification Protocol used to update clients (not able to participate in group communication) with the addresses of the current cluster membership when it changes. This is discussed in section 2.1.5 and thoroughly documented in the IIOP LB/FO design document (1.1). The design document and sections 2.1.5, 2.3 and 3 are the center of the work to support dynamic clusters.

1.1 Reference Documents

Reference Document	Location (i.e. URL, file, owner, etc.)
[1] AS 9.0 PRD / PCD	http://appserver.red.iplanet.com/apollo/apollo-artemis-pteam/PteamDocuments/EngPlanNotes/jsas8.1-requirements.sxc
[2] IIOP LB/FO design document	http://javaweb.sfbay.sun.com/~hcarr/pept/doc/FODynamicCluster/FODynamicCluster.pdfi
[3] AS 9.0 Architectural Overview	http://appserver.red.iplanet.com/apollo/architecture/appserv8ee_fsd.sxw
[5] OMG CORBA Specification 3.02	http://www.omg.org/technology/documents/formal/corba_iiop.htm
[6] sun-domain_1_1.dtd	http://iaseng.red.iplanet.com/apollo/config/sun-domain_1_1.dtd
[7] sun-application-client-container_1_1.dtd	http://appserver.red.iplanet.com/dtds/sun-application-client-container_1_0.dtd
[8] EJB Container Functional Specification for AS 9.0	http://appserver.red.iplanet.com/apollo/ejb/ejb_fsd.sxw

[9] Common Secure Interoperability Specification	http://www.omg.org/cgi-bin/doc?ptc/2001-06-17
[10] CORBA COSNaming Service Specification	http://www.omg.org/cgi-bin/doc?formal/00-06-19
[11] EJB 3.0 Specification	http://java.sun.com/products/ejb/docs.html
[12] Administration Functional Specification for AS 9.0	http://iaseng.red.iplanet.com/glaucus/documents/Glaucus_Admin_fs.sxw
[12] Group Membership Service	http://appserver.sfbay.sun.com/as933/eng/gms/GMSOnePager.txt

1.2 Glossary

This section provides a brief description of the terminology used in this document.

1. Remote Reference

A Remote Reference is an Interoperable Reference (IOR), for either an EJB [11] or a_ NameService [10] object. It is used by client programs to invoke remote operations. The Java object encapsulating the IOR is an RMI-IIOP stub.

2. Client

Remote References from the following client types are supported:

- Java applications executing in the ACC accessing EJBs deployed on an AS instance.
- Java Applications, not running in the ACC, accessing EJBs deployed on an AS instance. These are referred to as “standalone clients.”
- Servlets/JSPs in web applications executing in a different JVM than the target AS instance. The web application could also be running in a non-AS web container, for example, WS *. This case is is the same a standalone client accessing an AS instance.
- Clients started via Webstart (JNLP). As long as a correct sun-acc.xml is delivered to the client via JNLP the IIOP LB/FO should work without additional configuration.

These different client types for remote references are referred to as “clients” in this document.

3. Cluster-aware IORs

Remote References that are cluster-aware and can be failed over to an alternate AS instance in the cluster are referred to as Cluster-aware IORs. Cluster-aware references work on any AS instance within the cluster.

The terms “Remote References”, “References”, and “IORs” are used interchangeably to mean Cluster-aware IORs in this document.

4. Endpoint

The term endpoint refers to the host and port identifying an IIOP Listener on a target AS instance or NameService in the cluster. The ORB listens on the endpoint for incoming IIOP requests. A remote reference contains one or more endpoints.

The terms endpoint and IIOP Listener are used interchangeably in this document.

5. Primary Endpoint

This is the endpoint of the AS instance that created the IOR. By default, all method invocations on the reference are initially dispatched to this endpoint by the ORB, provided the endpoint is reachable. If the primary endpoint becomes unreachable then an alternate endpoint is tried.

6. Server Replicas

All AS instances participating in the cluster need to have the same applications deployed. Hence, instances in a cluster are also referred to as replicas.

The terms servers, instances, and replicas are used interchangeably in this document.

7. Load balancing

This refers to the process of distributing requests from clients to different server replicas or Name Servers. The goal is to spread the load evenly across the cluster to enable higher scalability.

8. HA of Remote References

A remote reference obtained by clients is highly available if invocations on it continue to work even after the instance that published the primary endpoint of the reference becomes unavailable. The invocations are failed over to another accessible endpoint in the cluster. As long as at least one instance is available the invocations on the HA reference will succeed.

1.3 Supported Features

The features supported by IIOP LB/FO are described in [1, 2]. These are summarized below:

1. InitialContext Load Balancing of IIOP requests in Dynamic Clusters.

   Whenever a client program does “new InitialContext” the IIOP LB/FO system will use randomized (perhaps weighted) round-robin to pick an instance in the cluster to which lookup requests, EJB Home lookups and EJB Object creation and method invocations are directed.

2. HA (failover) of Remote References in Dynamic Clusters.

3. IIOP-over-SSL HA is provided for secure remote references that contain a CSIv2 [9] tag component that provides the security configuration including endpoints for the target of the reference.

4. Dynamic Cluster Reconfiguration.

   Dynamic Cluster Reconfiguration is the addition or removal of AS instances when the cluster is online. Such cluster membership reconfiguration results in the client-side IIOP LB/FO system receiving an updated list of available instances.

1.4 Features Not Supported

1. Online upgrades.

2. Non-homogenous (in terms of applications, not hardware) clusters.

1.5 Differentiation

IIOP LB/FO was a newly supported feature in AS 7.1 EE. AS 8.1 EE was feature compatible with AS 7.1 EE. AS 9.0 EE supports the same functionality but within dynamic clusters.

2 Principles of Operation

2.1 Operational Overview

This section gives an overview of the user experience impact for this feature, namely, the additional configuration, and the client programming model.

2.1.1 Server-side Configuration

The user, via the Administration CLI or the GUI, updates domain.xml to define IIOP endpoints that constitute the cluster of AS instances. This change is required regardless of the type of the client. The relevant domain.xml DTD elements are described in section 2.3.3.3.

2.1.2 ACC Configuration

A description of the relevant properties is presented in section 2.3.3.2. Once these properties are set, no change in the application code (executing within the ACC) is necessary.

2.1.3 Standalone Client Application Configuration

For standalone clients, prior to instantiation of the InitialContext, certain environment properties need to be set. These properties can be set as JNDI SPI environment properties or as System Properties. The properties are:

- "java.naming.factory.initial" - The specialized initial context factory needed to support load balancing.
- “com.sun.appserv.iiop.endpoints” - The list of one or more IIOP endpoints. The endpoints represent listeners on AS instances or the NameService. Specifying more than one endpoints ensures that look up of references are highly available.
- “com.sun.appserv.iiop.loadbalancingpolicy” - The type of IC-based load balancing policy to be used. One policy is supported: randomized (possibly weighted) round-robin.
- “com.sun.CORBA.transport.ORBIIOPPrimaryToContactInfoClass' - The sticky manager.
- “com.sun.CORBA.transport.ORBIORToSocketInfoClass” - This picks out endpoints from an IOR.

A detailed description of the JNDI environment properties is presented in section 2.3.3.1.

The Sample Code snippet below illustrates how the environment properties are set when passed using JNDI SPI. Client code instantiates the JNDI InitialContext Object by calling "new InitialContext(env)", where env is the list of JNDI SPI properties listed above.

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Set the properties to be passed for creating the Initial Context. Note that
// these properties need to only be set for Standalone Clients. Alternatively,
// these settings can also be made by passing them as System properties.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
Properties env = new Properties();
env.put("java.naming.factory.initial",”com.sun.appserv.naming.S1ASCtxFactory");
env.put(" com.sun.appserv.iiop.endpoints ","trident:3600, exodus:3700”);
env.put(“com.sun.appserv.iiop.loadbalancingpolicy”, “ic-based”);

// Create an initial naming context
Context initial = new InitialContext(env);
```

2.1.4 Configuration and Backward Compatibility

All of the properties used in AS 9.0 are identical to those used in AS 7.x and AS 8.x. Therefore standalone and ACC clients from AS 7 and 8 should work in AS 9 without modification. Note: even if a client used the “com.sun.appserv.iiop.endpoints” property in AS 9, it will still do load balancing and failover with respect to the current cluster membership. This is accomplished by an internal Cluster Reconfiguration Notification Protocol (CRNP) that is discussed next.

2.1.5 Cluster Reconfiguration Notification Protocol

The following steps shows the Cluster Reconfiguration Notification Protocol (CRNP) used to inform clients outside the group communication domain of cluster member changes. The server ORB keeps up-to-date on the shape of the cluster by registering with the Group Membership Service (GMS). The ORB subsystem that handles CRNP is call the “IIOP Group Agent” (IGA).

1 Server IGA labels each cluster membership “shape” (i.e., each time a machine is added or removed from the cluster give the current membership “shape” a unique id).

2 Server IGA embeds that membership shape id in IORs along with addresses of all app server instances.

3 Client IGA sends the membership shape id (out-of-band in a ServiceContext) on each request.

4 Server handles the request as usual. Before returning the result, the server IGA intercepts the request response. If the shape id from the client is out-of-date then Server IGA returns (out-of-band in a ServiceContext) a new IOR that contains the current list of member addresses and an up-to-date membership shape id.

5 If a new membership list/id is received by the client IGA it will update its load-balancing and failover lists with the new info (but will stay "stuck" to the last server node).

The server IGA is registered with GMS. Each time it is notified of a membership change it creates a new shape id and updates its list of replica addresses.

Each server IGA maintains its own unique shape id. A shape id is just a UUID prepended with the server's IP address. That way shape ids do not need to be synchronized between server instances and, when a client fails over, an out-of-date shape id will not be incorrectly seen as valid by the failed to server instance. For a more detailed discussion of the scope of the shape id see the design document (1.1).

If GMS fails to function then the instance's shape ids will remain constant. This means that client's holding references that contain that shape id will not be updated with new references. If/when GMS functions again and assuming the shape has changed, the server IGA expects to be notified of any shape changes that occurred while GMS was not functioning. Alternatively, GMS can just notify the server IGA that it has come back up and that the IGA should reinitialize. In both cases this will cause a new shape id that will then cause client's to receive up-to-date information.

Note: The IGA is not built into the ORB (so we can deliver the ORB into J2SE
and SJSAS 9.0 PE without this feature). It plugs into the ORB via both standard OMG IORInterceptor/ServerRequestInterceptor interfaces and proprietary PEPT interfaces.

2.1.6 InitialContext-based Load-balancing

1. Each time client code calls “new InitialContext” the load-balancer connects the InitialContext to a different app server instance. This is done in a randomized (possibly weighted) round-robin fashion. If the instance picked by the load-balancer is not available then failover tries the next instance until an available instance is found.

2. Client code instantiates a JNDI InitialContext Object by calling "new InitialContext". If COSNaming is used instead of JNDI, a reference to the NamingContext is obtained via formatting the list of NameService replica endpoints as a corbaloc and giving that corbaloc to string_to_object method on the ORB. This results in a cluster-aware reference to the NameService.

3. Client code calls the lookup method on the JNDI InitialContext Object and passes the name of the object being looked up as a parameter. If COSNaming is used instead of JNDI, the resolve method on the NamingContext is called.

4. The resolve method invocation is directed to an appropriate endpoint on an AS instance in the cluster. The AS instance includes a colocated NameService. The NameService returns a remote reference.

5. When client code calls either the “create” or the “findByPrimaryKey” method on the EJBHome object, that call is directed to the AS instance that created the EJBHome object – the same instance that served the lookup of the EJBHome object.

6. When an EJB Object is created (via “findByPrimaryKey” or “create” on the EJBHome object) that object is created on the same AS instance where the EJBHome object resides. A remote reference to the EJB Object is returned to the client.

7. The client application uses the EJB Object reference to invoke business methods. Those invocations will initially be directed to the same AS instance that created the reference.

2.1.7 Failover

8. If an app server communication endpoint becomes unavailable, then subsequent invocations are redirected to an alternate app server endpoint in the cluster. All invocations to objects from the failed app server will be dispatched to the same alternate app server endpoint, thereby maintaining locality.

2.2 Architectural Overview

The conceptual model of an AS cluster and the common deployment scenarios are explained in the 9.0 EE Architectural Overview document [3].

2.3 Subsystem Overview

The subsystem diagram (Figure 1) depicts the interfaces between the IIOP LB/FO and external subsystems and interfaces. The association lines are labeled with interface numbers that match the particular entries in the interface table described in Section 3. IIOP LB/FO contains the following logical subsystems:

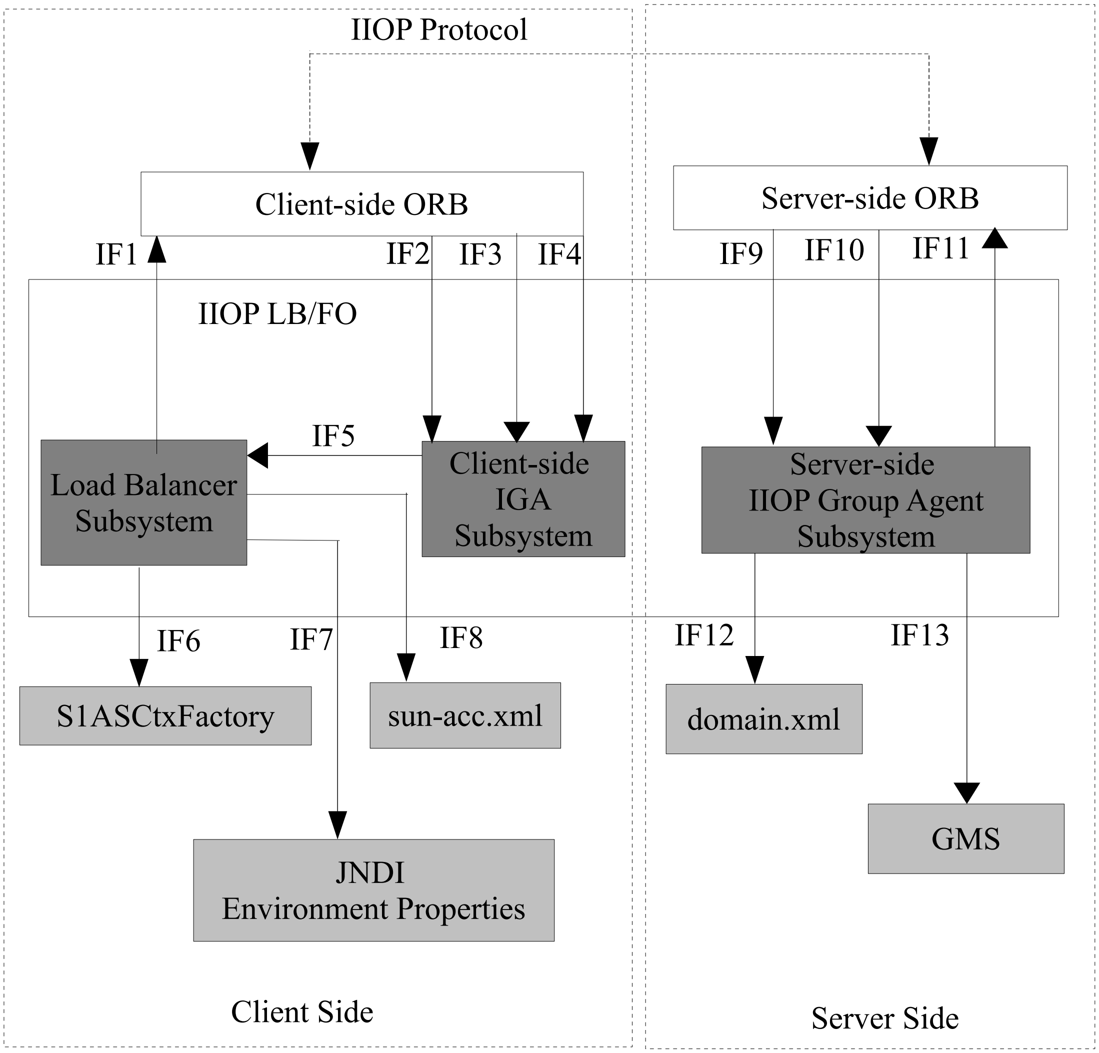
1. Client-side ORB. Handles IIOP communication.

2. Load Balancer. Handles InitialContext-based LB.

3. Client-side IIOP Group Agent. Responsible for group management and failover.

4. Server-side ORB. Handles IIOP communication.

5. Server-side IIOP Group Agent. Responsible for group management.



Interfaces

IF1: ORB API
IF2: IORToSocketInfo
IF3: IIOPPrimaryToContactInfo
IF4: ClientRequestInterceptor
IF5: UpdateEndpointList
IF6: S1ASCtxFactory
IF7: JNDI Environment Properties
IF8: sun-application-client-container_1_1.dtd

IF9: IORInterceptor
IF10: ServerRequestInterceptor
IF11: POA
IF12: domain.xml
IF13: Group Membership Service

Figure 1. Subsystem Overview for IIOP LB/FO

2.3.0.1 InitialContext (IC) based Load Balancing

The Load Balancer is responsible for distributing requests from clients to different server replicas to enable higher scalability.

IC-based LB makes the AS instance used to create an InitialContext service all subsequent requests on the InitialContext. Further, all objects looked up or created by objects looked up from a given InitialContext will reside on the same AS instance. This mechanism ensures "locality" for all the EJBHome and EJB Objects created using a particular InitialContext.

One load balancing policy is supported by the Load Balancer Subsystem:

- Randomized (possibly weighted) Round-Robin Load Balancing

2.3.1 Server-side IIOP Group Agent

The Server-side IGA provides the server-side infrastructure for supporting HA of Remote References in dynamic clusters. The core functionality is make IORs cluster-aware an to update IORs that are out-of-date with respect to the cluster membership. The role of the Server-side Failover Subsystem involves performing the following tasks:

- Generating Cluster-Aware IORs
- Updating out-of-date IORs

2.3.1.1 Generating cluster-aware IORs

Interoperable References are made cluster-aware by inserting endpoints of the server replicas in the cluster. The list of endpoints are derived from the cluster configuration information (domain.xml, described in section 2.3.4.3). The endpoints are categorized as those configured for non-SSL and those configured for SSL.

Non-SSL
IIOP endpoints are inserted in the IOR as a sequence of tag components of the type TAG_ALTERNATE_IIOP_ADDRESS. The IORInterceptor interface, provides the mechanism to insert the TAG_ALTERNATE_IIOP_ADDRESS tag components in the IOR. Details pertaining to IORInterceptor are provided below.

SSL*

SSL enabled IIOP endpoints are inserted in the IOR by the IORInterceptor interface as a sequence of TransportAddress within the CSIv2 tag component of type TAG_TLS_SEC_TRANS.

In both cases, an IORInterceptor adds tag components to the tagged IIOP profile (TAG_INTERNET_IOP) of the template used to create IORs. Additionally, the POA used to create these IORs is made persistent. This ensures that all IORs generated by the create_reference API of the POA are cluster-aware and persistent (survive server shutdowns and crashes). The generated IORs contains a TAG_ALTERNATE_IIOP_ADDRESS (or TAG_TLS_SEC_TRANS tag components in the SSL case*) component with host and port for each endpoint. Figure 2 illustrates a cluster-aware IOR.

Note: The IORInterceptor class is registered during server initialization as follows:

```
-Dorg.omg.PortableInterceptor.ORBInitializerClass.<initializer>=dummy
```

where <initializer> adds an IORInterceptor.

2.3.1.2 Updating out-of-date IORs

The server IGA registers with GMS. Whenever GMS notifies the server IGA that an instance has joined or left the cluster the server IGA recreates POAs. This causes the POAs to run IORInterceptors again. The IORInterceptors then add the current shape id and list of instance addresses to the Object Reference Template. Therefore, if the EJB level creates IORs they will contain the latest membership list.

However, clients may contain out-of-date IORs. When a client sends a request, the client IGA adds the shape id (from the IOR held by the client) as out-of-band data (i.e., in a header ServerContext) to the request. After the server side has serviced the request, the server IGA examines the client's shape id. If it is out-of-date, it uses the Object ID contained in the request to find the appropriate POA and create a new up-to-date IOR. It then returns this IOR to the client as out-of-band data. The client IGA updates replaces the old IOR with the up-to-date IOR.

2.3.1.3 Supporting Failover of NameService References

In AS 9.0, NameService support for publishing EJBHome is provided via the following transient object implementations:

TransientNamingContext

This is the implementation of the COSNaming service provided in AS 9.0. ACC clients interface with this object through the JNDI InitialContext for NameService operations.

SerialContextProviderImpl

In AS 9.0, this object is published to the CosNaming Service as a sub-context and is implemented as an RMI-IIOP object. Standalone Java based clients interface with this object through the JNDI InitialContext, for NameService operations. This class is also used by ACC clients to lookup non-EJB objects (e.g.: JDBC DataSource).

Both TransientNamingContext and SerialContextProviderImpl must be instantiated using a persistent POA. This ensures that the reference for the Initial NameService survives server restarts.

Further, each context has a (different) constant objectid. This ensures that, during failover of the NameService Remote Reference, the RMI-IIOP invocations on the reference succeed on the failed over AS instance. It also ensures that if the NameService goes down and then comes back up, previous references will continue to work with the restarted NameService.

2.3.2 Client-side IIOP Group Agent

The client-side IGA works in tandem with the Server-side IGA to support HA of Remote References. The role of the client-side IGA involves:

- Selecting an alternate endpoint during failover
- Ensuring “stickiness” of the failed over Reference
- Updating out-of-date IORs

2.3.2.1 Selecting an alternate endpoint during failover

The functionality described in this section is contingent upon the server-side IGA generating cluster-aware IORs (discussed previously in section 2.3.2.1). Failover happens only for those requests that cannot reach the server and cause a CORBA COMM_FAILURE exception with return status of COMPLETED_NO on the client.

The IORToSocketInfo interface is responsible for picking all endpoints out of the IOR and ordering them in the order they should be tried if failover occurs. When SSL is plugged into the ORB it must supply an implementation of this interface which picks out SSL endpoints. This class is registered during client initialization as follows:

```
-Dcom.sun.CORBA.ior.ORBIORToSocketInfoClass=SSLIORToSocketInfoImpl
```

When a COMM_FAILURE occurs on the client-side, the ORB's ContactInfoListIterator is responsible for picking the next endpoint in the ordered endpoint list provided by IORToSocketInfo. IORToSocketInfo constructs this endpoint list from TAG_ALTERNATE_IIOP_ADDRESS IOR tags (TAG_TLS_SEC_TRANS IOR tags for the SSL case *) present in the IOR (placed there by an IORInterceptor on the server side). Once the iterator selects an alternate endpoint, a new IIOP connection socket is opened to it, and the request is failed over to the alternate endpoint. The implementation details for the IORToSocketInfo are described in the Appendix (Section 9).

When SSL is being used there must be an SSL socket factory plugged into the ORB to create SSL sockets. The AS 9.0 ORBSocketFactory is registered during client initialization as follows:

```
-Dcom.sun.CORBA.transport.ORBSocketFactoryClass=IIOPSSLSocketFactory
```

2.3.2.2 Ensuring “stickiness” of a failed over Reference

An IOR is published by any one of the AS instances in the cluster. This AS instance is identified by the primary endpoint published in the IOR. When the primary endpoint becomes unreachable due to the AS instance going down, the request is failed over to an alternate endpoint. All subsequent requests using the IOR are sent to the same alternate endpoint. For example, if the remote reference was on Server 1 and Server 1 becomes unavailable, the remote reference will be failed over to Server 2. Later, if Server 1 becomes available again, requests on the IOR are still sent to Server 2. This is done so as to ensure that object underlying the IOR does not simultaneously exist on two server instances. For instance, if the IOR was for a SFSB, it would be erroneous to have the SFSB state being updated and accessed from two instances concurrently. Hence, it is mandatory that “stickiness” or “localization” is maintained for failed over requests.

The IIOPPrimaryToContactInfo interface is responsible for maintaining stickyness. It maintains a dynamic mapping of primary endpoints to alternate endpoints for all failed AS instances. For all failed over requests, it retrieves the primary endpoint from the IOR and uses it to retrieve the alternate endpoint. The request is then dispatched to the alternate endpoint.

The AS 9.0 EE IIOPPrimaryToContactInfo is registered in ORBManager during client initialization as follows:

```
-Dcom.sun.CORBA.transport.ORBIIOPPrimaryToContactInfoClass=MapIIOPPrimary
```

2.3.2.3 Updating out-of-date IORs

When the client IGA receives an updated IOR from the server IGA it updates the stub holding the reference with the new IOR. This causes the ContactInfoList to get updated to the current list of addresses. It does NOT update the sticky manager (i.e., the client stays stuck to the same node).

The client IGA also notifies the IC Load Balancer of the new IOR. The load balancer then uses the new list as the set of instances to load balance across.

Note, during the name service bootstrap the client side will not contain a shape id. When the server IGA sees a message from a SUN ORB that does not contain a shape ID it will send the current IOR that contains the addresses for the current shape and the shape id itself. This causes the load-balancer to load-balance across all available instance even if only a subset of instances were specified in the initial bootstrap endpoint list.

2.3.3 IIOP LB/FO Configuration Changes

This section describes the configuration changes required for IIOP LB/FO. The configuration changes are:

- Properties to support LB/FO for Standalone Clients
- Properties to support LB/FO for ACC Clients
- Modifications to sun-domain.xml DTD

2.3.3.1 Properties to support LB/FO for Standalone Clients

These properties are required to be set ONLY for Standalone Clients. The properties can be set either on the JNDI InitialContext (as SPI Environment Properties) or passed as System Properties (via the -D flag). The table below provides a description for each of the properties:

Property	Description
java.naming.factory.initial	This specifies the Context Factory that should be used for load balancing IIOP requests. The property has to be set to com.sun.appserv.naming.S1ASCtxFactory. This property is only needed for load balancing.
com.sun.appserv.iiop.endpoints	A comma separated list of one or more IIOP endpoints. An IIOP endpoint is specified as host:port, where host is an IP address or a hostname resolvable by DNS, and the port specifies the port number.
com.sun.appserv.iiop.loadbalancingpolicy	If the endpoints property is specified, then the property is used to specify the load balancing policy. This property can have one value, "IC-based".
com.sun.CORBA.transport.ORBIIOPPrimaryToContactInfoClass	This property specifies the class used by ContactInfoListIterator to ensure stickyness.
com.sun.CORBA.ior.ORBIORToSocketInfoClass	This property specifies the class used to extract clear-text and SSL endpoint information.

2.3.3.2 Properties to support LB/FO for ACC Clients

No properties need to be set. One specifies one or more app server instance iiop endpoints in the **target-server** element. The default load-balancing policy is "ic-based".

In AS7 and AS8 load balancing was enabled by specifying a list of endpoints via the "com.sun.appserv.iiop.endpoints" property. In that case the <target-server> element in the DTD is ignored. However, this is confusing to developers. To avoid confusion, the <target-server> element has been changed to <target-server+>. With this change the developer does not need to specify the "com.sun.appserv.iiop.endpoints" property for an ACC. Instead, all endpoints can be specified as part of the <target-server+> element. ("com.sun.appserv.iiop.endpoints" will be supported in 9.0 for backward compatibility.)

The relevant change of the sun-application-client-container_1_1.dtd[7] that will be used to specify the <target-server> element is:

<!ELEMENT client-container (**target-server**,auth-realm?,
client-credential?, log-service?,message-security-config*, property*)>

changes to:

```
<!ELEMENT client-container (target-server+,auth-realm?,
client-credential?, log-service?,message-security-config*, property*)>
```

The following properties are supported (but not required) in sun-acc.xml ONLY for ACC Clients to be compatible with AS 8. The properties are:

- com.sun.appserv.iiop.endpoints
- com.sun.appserv.iiop.loadbalancingpolicy

The description of these properties is identical to the corresponding environment properties described in the preceding section.

2.3.3.3 sun-domain_1_1.dtd elements

This section has not changed from AS 8.1 EE. It is included to make this document self-contained.

The IIOP LB/FO subsystem needs information about all IIOP endpoints in a cluster as specified within domain.xml. This is automatically by obtained by looking at the named <config> element corresponding to the cluster in which the instance manufacturing the IOR belongs to.

The following snippets of domain.xml DTD will clarify the structure. A cluster is defined as follows:

```
<!ELEMENT cluster (server-ref*, ..........)>
<!ATTLIST name CDATA #REQUIRED
              config-ref CDATA #REQUIRED>
```

server-ref sub-elements list all server instances that belong to the **cluster and config-ref** points to a named configuration profile which is shared by all instances in the cluster.

```
<!ELEMENT config (http-service, iiop-service,.......)>
<!ATTLIST   config name #REQUIRED
                      .......                >
```

iiop-service element describes the IIOP listeners at each server instance.

```
<!ELEMENT iiop-service (orb, ssl-client-config, iiop-listener*)>

<!ELEMENT iiop-listener (ssl? , property*)>
<!ATTLIST    iiop-listener id CDATA #REQUIRED
                           address CDATA #REQUIRED
                           port       CDATA “1072”
```

```
security-enabled %boolean;  "false"
enabled  %boolean; "true">
```

Multiple IIOP listeners are permitted per instance (which means that multiple IIOP ports are opened, per instance).

3 Interfaces

3.1 Interface Table

3.1.1 Imported Programmatic Interfaces

Interface Name	Stability Classification	Former Stability Classification	Specified in Document
IF1: ORB API	Standard	Standard	[5]
IF2: IORToSocketInfo	Project Private	Project Private	This document, Section 2.3.3.1
IF3: IIOPPrimaryToContactInfo	Project Private	Project Private	This document Section 2.3.3.1
IF4: ClientRequestInterceptor	Standard	Standard	[5]
IF9: IORInterceptor	Standard	Standard	[5]
IF10: ServerRequestInterceptor	Standard	Standard	[5]
IF11: PortableObjectAdapter	Standard	Standard	[5]
IF13: Group Membership Service	Porject Private	Project Private	??

3.1.2 Exported Programmatic Interfaces

Interface Name	Stability Classification	Former Stability Classification	Specified in Document
IF6: S1ASCtxFactory	Evolving	Evolving	This document, Sections 2.1.1 and 2.3.1

3.1.3 Exported Configuration Interfaces

Interface Name	Stability Classification	Former Stability Classification	Specified in Document
IF7: JNDI Environment Properties	Evolving	Evolving	This document, Section 2.3.4.1
IF8 : sun-application-client-container_1_1.dtd	Unstable	Unstable	This document, Section 2.3.4.2
IF12: sun-domain_1_1.dtd	Unstable	Unstable	This document, Section 2.3.4.3

4 Design Overview

4.1 Proposed Approach

A detailed discussion of the design issues has been presented in Section 2.3 and the Appendix.

4.2 Alternatives Considered

The following were alternative approaches considered to support failover of IIOP requests:

- Exposing Cluster Configuration to the Client
- Implementing OMG Fault Tolerant CORBA
- Using Smart Stubs

4.2.1 Exposing Cluster Configuration to the Client

One alternative approach is to expose the entire cluster configuration, the endpoints for each server replica, to the client. The ContactInfoListIterator uses this information to failover the request to another server IIOP endpoint, whenever a communication failure occurs with the primary endpoint.

The advantage of this approach is that IORs need not be cluster-aware. The downside to this approach is that the entire cluster configuration has to be exposed to the client. Further, whenever the cluster configuration changes, the endpoints exposed to the clients will no longer be valid unless the client is able to contact the Group Membership Service, an assumption we cannot make.

4.2.2 Implementing Fault Tolerant CORBA

The design document (1.1) discusses our approach to LB/FO with respect to FT CORBA. The main difference is scope: we are concentrating on LB/FO of IIOP communications whereas FT CORBA is a framework that covers all aspects of FT.

4.2.3 Using Smart Stubs

This approach requires building intelligence into the stubs to detect IIOP communication failure on the primary endpoint and to retry the request to another alternate endpoint. Even in this alternative, the cluster configuration is exposed to the client and the smart stubs interface use this information to locate the alternate endpoint.

This alternative has all the advantages and disadvantages of the previous alternative. Further, it has the following additional disadvantages:

- There would be type-independent behavior in type-dependent stub code.
- The size of the stub code increases.
- Smart stubs are hard to manage if an application uses multiple implementations of the same type.

4.2.4 Monitoring

AS 9.0 EE IIOP LB/FO will support a limited amount of monitoring. To be determined. Monitoring will be done on a per-ORB basis. Since the app server uses only one ORB this means that monitoring will be done on a per-app server instance basis on the server-side. Similarly on the client-side, assuming clients uses ORBManager to get an ORB. If clients explicity create an ORB then each ORB is monitored separately.

5 Performance

5.0.1 Performance Goals

The performance goal for IIOP LB/FO is that there should no degradation in performance, as compared to AS 9.0 PE, while processing requests that are not failed over.

5.0.2 Benchmarks

The SpecJAppserver benchmark will be used to measure the performance.

5.0.3 Performance Risks

There are no specific performance risks associated with the IIOP LB/FO.

5.0.4 Scalability

The horizontal scalability of IIOP LB/FO is directly proportional to the number of AS instances in the cluster. The vertical scalability of the system depends on the scalability of the underlying ORB.

6 Reliability, Availability, Serviceability

6.0.1 Reliability

The system will detect failures during IIOP invocation. It will validate successful failover of remote references. However, there can be cases when the request fails but there would be no failover. For example, requests that cannot reach the server and cause a CORBA COMM_FAILURE exception with return status of COMPLETED_MAYBE on the client.

6.0.2 Availability

The client should specify at least two endpoints using the “com.sun.appserv.iiop.endpoints” property to ensure that the NameService does not become the single point of failure. When the client IGA contacts an AS instance it is then informed of ALL AS instances in the cluster and kept up-to-date with respect to cluster configuration changes.

The availability of the system will be higher if more AS instances are configured in the cluster.

6.0.3 Serviceability

Informational messages, errors, and warnings will be logged using the standard logging mechanism in AS 9.0:

- For non-ACC Java Client applications, the log information is directed to standard output.
- For ACC Clients, the log information will be directed to the log file specified in sun-acc.xml. If no log file is configured, then the log information is directed to standard output.

7 Administration and Tools

The following Admin capabilities will be supported:

1. The CLI and GUI will populate modified domain.xml.
2. asadmin supports cluster-wide administration.
3. The CLI will support monitoring of parameters specific to IIOP LB/FO.

The administration functionality is described in detail in [12].

8 Feature Delivery

This feature will be delivered in AS 9.0EE. The same JDK versions supported in 9.0 PE will be supported. The operating system platforms supported are:

- Solaris SPARC 10 and 9
- Red Hat Linux ??
- More platforms considered per product, JES requirements and schedule

8.1 Packaging, Files, and Location

A modified AS runtime library will be delivered in 9.0 EE to support this feature. The names of the libraries, their location in the installable, and also their location post-installation, will be unchanged with respect to 9.0 PE .

8.2 Upgrades

The upgrade from 9.0 PE to 9.0 EE will take care of replacing the existing AS library with the upgraded library. The existing applications would work as is after the upgrade. Backward compatibility with respect to AS 8.1 EE will be maintained for the “com.sun.appserv.iiop.loadbalancingpolicy” property.

8.3 Installation

There are no specific installation requirements for IIOP LB/FO. It would be installed as part of AS 9.0 EE install.

8.4 Licensing

There are no special licensing requirements specific to this feature.

8.5 Internationalization

The aspect of IIOP LB/FO that relies internationalization is logging. Since the AS 9.0 Logging Framework will be used, support for internationalization is already built in. Hence, there are no new internationalization requirements for IIOP LB/FO.

8.6 Documentation Requirements

The following documents will be modified or newly created for this feature:

- Recommended deployment and administration practices for ensuring HA
- Administration Guidelines to support HA
- HA and Load Balancing Features
- Client Programming Guidelines

9 User Experience

The issues that relate to the user experience for IIOP LB/FO have been previously described in sections 2.1 and 5.

10 Dependencies

This feature depends on the Group Membership Service notifying the IIOP Group Agent when AS instances join or leave the cluster.

This feature depends on the SFSB Failover Support [8] that was implemented in 8.1 EE.

11 Open Issues

CSIv2/SSL/IIOP working with load-balancing and failover.

```
* IIOP-FOLB/Security meeting/tasks

To: Ken.Cavanaugh@Sun.COM, Shing-Wai.Chan@Sun.COM,
Ronald.Monzillo@Sun.COM, Mimi.Hills@Sun.COM, Vella.Raman@Sun.COM
Cc: sreeram.duvur@sun.com, bill.shannon@sun.com
Subject: Secure LB/FO phased work items
--text follows this line--
Hello all,

We decided to partition the work into two phases:

- Phase one: CSIv2 failover (but non-secure bootstrap).
- Phase two: CSIv2 bootstrap to NameService.

Phase two *MAY* possibly may *NOT* be done for 9.0 depending on resources.

ASSUMPTION:

        *NOT* doing phase two ASSUMES that requests to Secure EJBs
        that arrive on a CLEAR_TEXT connetion will be REJECTED by CSIv2.

------------------------------------------------------------------------
PHASE ONE:  CSIv2 failover

------------------------------------------------
Task: Server side: add all CSIv2 host/port addresses to IOR

        1. IIOP-FOLB team:
          Get list of CSIv2 host/port addresses of all cluster
```

instances from admin (like we do for CLEAR_TEXT now).

Give that list to CSIv2 module.

2. Security team:

Given list of addresses, return CSIv2 component(s) that must be added to an IOR.

3. IIOP-FOLB team:

Add CSIv2 components to IOR.

Notes:

Steps 1 & 3 will be coded in:
com.sun.enterprise.iiop.TxSecIORInterceptor.addCSIv2Components

The hostname/ssl-port of the node executing the TxSecIORInterceptor *MUST* be the first address in the list.

Task: Client side: extract all CSIv2 host/port addresses from IOR

1. IIOP-FOLB team:

On each invocation give IOR to CSIv2.

2. Security team:

Given an IOR, return List of SocketInfo containing CSIv2 addresses and socket type.

If IOR does not contain CSIv2 information return null.

3. IIOP-FOLB team:

If step two returns null then return List of CLEAR_TEXT SocketInfo.
Otherwise return CSIv2 List from step two.

4. IIOP-FOLB Team:

Use the List returned from above for failover.

Note:

Step 1 & 3 will be coded in:
com.sun.enterprise.iiop.IORToSocketInfoImpl.getSocketInfo

The order of the return List should be the same
as the order in the IOR.

Task: Debugging:

1. IIOP-FOLB team:

Add property to disable load-balancing.

Add property to disable failover.

PHASE TWO: secure bootstrap

Task: Server side:

1. Security Team:

Determine if/how the NameServer needs to be configured for SSL.

Task: Client side:

1. Security and IIOP-FOLB teams:

When the client is given CSIv2 host/ports it needs to
create an IOR to make a CSIv2 invocation.

Notes:

Possible design:

1. IIOP-FOLB Team:

Implement a proprietary "secure corbaloc" to create
a secure bootstrap IOR for initial communication
with the NameService. We will create a OA from the
secure corbaloc. The creation of the OA will call
TxSecIORInterceptor.addCSIv2Components that will add

the necessary components to the ORT to enable creation of a secure IOR.

2. Security Team:

Ensure that com.sun.enterprise.iiop.TxSecIORInterceptor.addCSIv2Components will operate correctly if called from a "client" ORB.

12 Appendix – IIOPPrimaryToContactInfo and IORToSocketInfo Implementations Details

IORToSocketInfo is used by the Client-side ORB to obtain a list of endpoint addresses. IIOPPrimaryToContactInfo is used to ensure stickyness after failover. The following sequence of steps describe the interaction between the Client-side ORB and these classes.

1. The Client makes method call on a remote reference.

2. Client-side ORB calls ContactInfoList.getContactInfoListIterator.

3. Client-side ORB calls iterator.next() to obtain a ContactInfo that represents the host/port from the cluster-aware IOR (discussed in detail in section 2.3.3.1).

4. If this is the first call to the reference then the iterator calls IORToSocketInfo to obtain a list of endpoint addresses. The iterator picks the first endpoint (i.e., the primary), calls IIOPPrimaryToContactInfo.put, encapsulates it in a ContactInfo, and returns that ContactInfo.

5. If this is a subsequent call to this reference then the iterator calls IIOPPrimaryToContactInfo.get, encapsulates the mapped endpoint in a ContactInfo and returns that ContactInfo.

6. Using the ContactInfo (i.e., host/port information in the endpoint), the Client-side ORB checks if a connection already exists, and if so, it reuses the existing connection.

7. If no connection exists, the Client-side ORB calls createSocket method on the ORBSocketFactory class to create a new connection (this enables CSIv2 to create SSL sockets).

8. If the createSocket method is successful, it returns a connection to the indicated host/port.

9. If createSocket cannot create a connection, the ORBSocketFactory class throws the exception raised by the underlying network.

10. The Client ORB catches the exception.

11. The Client-side ORB calls iterator.hasNext. If false go to step 13.

12. Otherwise Client-side ORB calls iterator.next() again to obtain a different ContactInfo that represents a different host/port. “next” will pick the next endpoint in the endpoint list, call IIOPPrimaryToContactInfo.put, then encapsulate and return a ContactInfo. Go to step 6.

13. If no connection is established even after trying all available endpoints, a COMM_FAILURE exception is returned by the Client-side ORB to the Client.