# IIOP Failover in Dynamic Clusters

Harold Carr and Ken Cavanaugh, Sun Microsystems

*Abstract* —

***Requirement*: Clients should experience high-availability when accessing network services. Availability should be transparent and not require altering programs.**

***Problem*: Availability needs to work on multiple platforms and must not require additional hardware/software, other than the remoting system.**

***Solution*: Replicate the service in a cluster. Advertise all replica addresses. On client invocation, if a replica address fails, have the remoting infrastructure try another address.**

***Requirement*: The number of replicas in the cluster can change dynamically (e.g., more/less instances to handle heavier/lighter loads, instances failing, online upgrades).**

***Problem*: Clients need to know the current cluster membership but are not able to participate in group membership communication.**

***Solution*: Give the initial cluster membership a label. Whenever membership changes generate a new label. Advertise the label. When a client invokes a service, send the label (as "out-of-band" data) along with the request. When servicing a request *and* the label is out-of-date, have the server return the new membership label and addresses as out-of-band data in the reply.**

***IIOP*: Add label/addresses to IORs. When invoking an IOR try another address if one fails. Cache successful addresses to avoid fallback (important for stateful-sessions). Send the label with requests as a ServiceContext. If out-of-date return an up-to-date reference with the reply in a ServiceContext.**

***Experience*: In an application server, this failover mechanism did not degrade system performance.**

***Contribution*: This failover mechanism provides high-availability IIOP communication in dynamic clusters.**

*Index Terms* —**high-availability, fault-tolerance, failover, middleware, IIOP.**

## I. Introduction

HIGH-AVAILABILITY (HA) of Enterprise Java Beans (EJB) [1] motivated this work. The main ideas are:

- Provide a service by replicating it in a cluster of machines.
- (re)label the current cluster membership.
- Advertise all replica addresses and the membership label.
- Have the client-side remoting system transparently try another address if one fails (and cache and use the last successful address).
- When the number of replicas change update the membership label in each replica.
- Have the client-side remoting system send its last seen membership label along with requests.
- When servicing requests, have the server-side remoting system compare the client's membership label with the server's label. If the client's is out-of-date then return current addresses and the current label along with the reply.

We apply those main ideas to IIOP ([2] 15.7) communications by:

- using an OMG `IORInterceptor` ([2] 21) to add alternate cluster addressing and label information as `TaggedComponent`s to IORs (i.e., object references [2] 13.6.2) created by servers;
- using PEPT [3], [4], [5], [6] `ContactInfo` to choose and failover to usable addresses transparently to client programs;
- providing a plug-in client-side cache to control how usable addresses are remembered;
- using an OMG `ClientRequestInterceptor` ([2] 21.3.5) to add membership labels as `ServiceContext`s ([2] 13.7) to requests;
- and by using an OMG `ServerRequestInterceptor` ([2] 21.3.8) to compare the client's membership label with the current label. If out-of-date the interceptor creates an up-to-date IOR and returns it as a `ServiceContext` in the response.
- The server-side ORB registers with a Group Membership Service to be notified of cluster membership changes.
- Server-side POAs ([2] 11), used to create object references containing the membership label and cluster addresses, are updated safely (with new labels/addresses when the cluster membership changes) while the system is handling requests by using the POAManager flow control mechanisms and POA adapter activators in a module known as the Reference Factory Manager (RFM).

This paper discusses how HA is accomplished in the RMI-IIOP [7] and CORBA IDL [8] programming models. These models are provided by systems available from a number of vendors. In particular, they are available as part of Sun's JDK since Java 1.2. The RMI-IIOP programming model is also used as a communication mechanism for EJBs in Java EE-compliant [9] application servers, the use-case for this work.

## II. Server configuration

Each instance (i.e., service replica) in a cluster is configured such that object references created on that instance, besides containing the address of the instance, also contain the addresses of the other instances in the cluster and a membership label. This is accomplished by arranging to have the Object Reference Template (ORT [2] 21.5.2) associated with each Portable Object Adapter (POA [2] 11) contain the instance's address in the IIOPProfile primary address portion of the ORT, the addresses of the other instances as `TAG_ALTERNATE_IIOP_ADDRESS` components ([2] 13.6.6.2), and the membership label as a custom `TaggedComponent` as shown in Figure 1.

In Figure 1 we show only the ORT fields of interest to our design. The ORT is used to create IORs. The ORT has an empty ObjectKey field that will be filled in when `POA.create_reference` is called to create an IOR. Addresses are the hostname and port number on which the

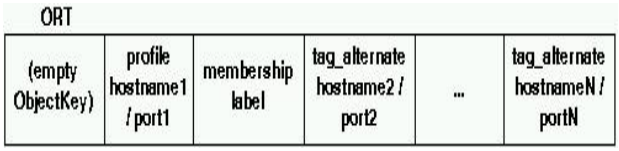


Fig. 1. ORT with multiple addresses

service is available. The details of how this is accomplished are as follows.

## III. The Reference Factory Manager

Dynamic IIOP failover must be able to reconfigure the ORT to contain a different membership label and list of addresses whenever the cluster membership changes. The ORT is created when a POA is created, so changing the ORT is most easily accomplished by (re-)creating a POA.

In our application server, each deployed EJB type corresponds to two POAs, one for the EJB home interface, and one for the EJB itself. When the cluster membership changes, reconfiguration must be done for all POAs created for EJBs in the application server.

### A. Non-interference between EJB POAs and user POAs

However, the ORB is itself a shared resource, and may be used by user code in the application server for any purpose, including creating POAs. We only provide failover capabilities for EJB, since the general failover problem cannot be solved transparently to the POA user.

Our failover solution requires POAs that participate in failover to be persistent (since transient POAs do not survive a server instance restart) and to only use POA object activation models that are stateless (which means that the POA's Active Object Map cannot be used). (Note, the EJB layer is responsible for managing the EJB instance state, so the EJB layer uses the `ServantLocator` model in its POAs.)

To manage EJB POAs without interfering with or being interfered by user POAs, we created a Reference Factory Manager (RFM) abstraction.

The RFM creates a parent POA that is a child of the root POA with an internal name that is unlikely to be used by another user of the shared ORB instance. All POAs in the RFM share a single RFM POA manager, which is not used by any non-RFM POAs.

The RFM also has a private POA policy that is used to indicate that a particular POA is being created in the RFM. This policy is checked by an `IORInterceptor` that is part of the RFM implementation. The `components_established` method throws an exception if the parent POA of the POA being created is the RFM parent POA, and if the POA being created does not have the RFM policy. This prevents user code from creating children of the RFM parent POA.

The `components_established` method also checks that the POA manager of the POA being created is the RFM POA Manager if and only if the POA has the internal policy.

### B. RFM implementation

The RFM acts as a factory for ReferenceFactory instances. A ReferenceFactory is simply a thin wrapper for the state needed to create a POA: the `ServantLocator`, appropriate policies and a repository ID. The repository ID is needed (although not needed to create a POA) to create updated IORs when stale membership labels arrive on requests. There are two Reference Factories (and hence POAs) per EJB instance, the home and object interfaces—each with a different repository ID.

Creating a reference factory does not create a POA. The POA is created on demand using an `AdapterActivator`. The `AdapterActivator` is triggered either by creating a new reference in a call to the reference factory, or by a request to the application server that references the required POA.

The use of an `AdapterActivator` is required to prevent spurious non-recoverable errors that could be observed by a client. This could happen between the time a POA is destroyed and re-created, even if the POAManager is holding requests, because an attempt to dispatch an incoming request to a momentarily non-existent POA will fail unless an `AdapterActivator` is used. The `AdapterActivator` mechanism is integrated with the POA destroy and request dispatch code to prevent such an occurrence.

### C. RFM operation

The RFM is responsible for managing the ORT update during cluster membership change. This works as follows:

- The Server Group Manager (SGM) is notified of a cluster membership change by the Group Membership Service (GMS).
- SGM invokes the RFM suspend method. This acquires a lock on the RFM state, and also invokes the `hold_requests( true )` method on the RFM's POA manager. After this call to `hold_requests( true )` returns, there are no requests actively executing in the RFM POA Manager, and all new requests are held. Requests not using the RFM continue normally.
- The SGM updates the membership label, so that newly created POAs will have both the new membership label and the updated set of addresses of the cluster members.
- The SGM calls the RFM restartFactories method. This method simply destroys all of the RFM POAs (this is why the RFM POAs need to be stateless).
- The SGM calls the RFM restart method. This method calls the RFM POA manager activate method, and normal request processing resumes. The RFM lock is dropped after this point.

## IV. Server initialization and reference creation

Server-side initialization uses both standard OMG CORBA interfaces and non-standard PEPT interfaces. Figure 2 shows the steps taken during initialization. We see that each application server instance has containers for EJBs. A container calls (0) `RFM.create` to obtain a ReferenceFactory with which to create object references. This

call takes place whenever an EJB is deployed. There are two ReferenceFactories per EJB: one for the EJB home, and one for the EJB remote interface (only one is shown in the diagram). `RFM.create` records the information that will be used later when a reference is created (notably, the repository ID). The RFM does not create a POA internally at this time.

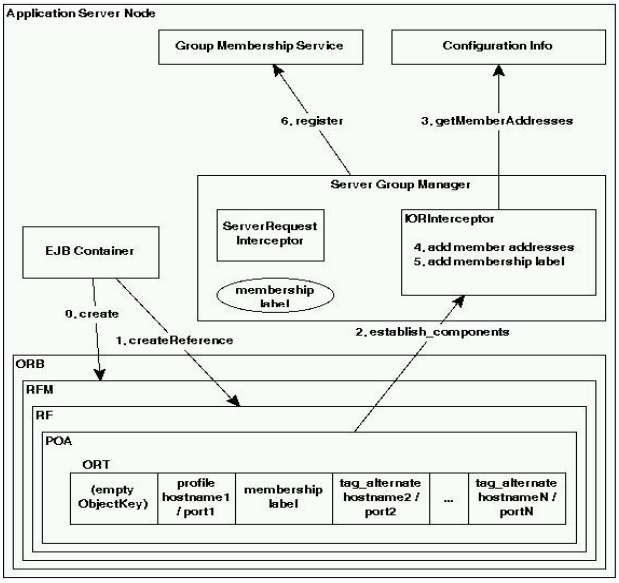


Fig. 2. Server initialization

The rest of the Server-side initialization takes place lazily when either the first EJB instance is created for a particular EJB type, or when the first request arrives for that EJB type (after a server restart). Here we will only look at the first creation case; the first request case is similar, but starts at the point in the request dispatch cycle where the appropriate POA is located.

Continuing in Figure 2, the EJB container calls (1) `RF.createReference` in order to create an instance of an EJB home or remote interface. The ReferenceFactory then calls `create_POA` (not shown), which calls (2) `establish_components` on the `IORInterceptor` that is part of the instance's Server Group Manager (SGM). The `IORInterceptor` calls (3) `getMemberAddresses` on an administration agent that contains cluster configuration information. The SGM associates a membership label with the list of addresses. It adds the (4) addresses and the (5) membership label to the ORT by responding to `establish_components`. This causes the ORT to contain the addresses of the other cluster instances as `TAG_ALTERNATE_IIOP_ADDRESS` tagged components and the membership label as a custom `TaggedComponent` in the IIOPProfile. The POA puts the address of the instance executing this code in the IIOPProfile primary. Finally, (6) the Server Group Manager registers with the Group Membership Service to receive cluster membership change notices. (The SGM also contains a `ServerRequestInterceptor` that will be discussed later.)

Figure 3 shows that after the RFM has created the POA, `RF.createReference` (1) creates the specific EJB by calling `POA.create_reference(objectid)` (not shown) with an Object ID identifying the EJB. After EJB references (to EJB "home" interfaces) with specific Object IDs have been created the resulting references are generally placed in a NameService (2).

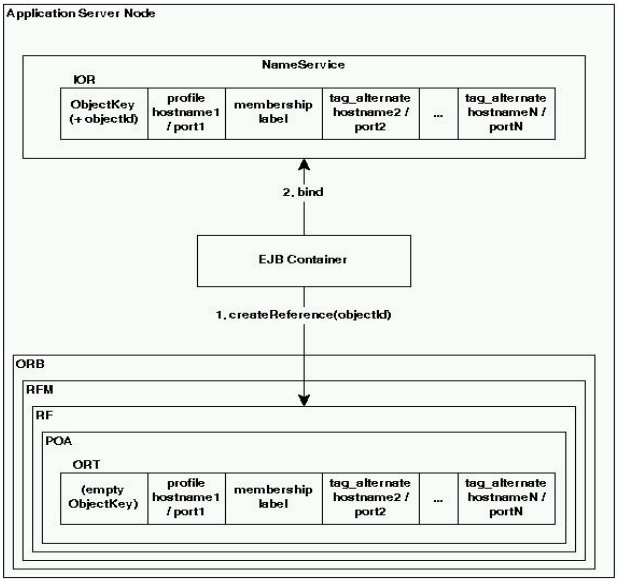


Fig. 3. Create and bind reference

The references (i.e., IORs) contain the contents of the ORT plus an ObjectKey. The ObjectKey contains information used by the ORB to dispatch requests. It also contains the Object ID specified by the EJB layer to map the request to a specific EJB. Creating EJB instances via the home interface goes through a similar process except the resulting reference is returned directly to the client rather than bound in naming.

## V. Client configuration

The client-side ORB is configured by plugging in an Client Group Manager (CGM) into the ORB. The CGM has a subsystem to convert IORs into an internal representation of a list of addresses; a cache to remember the last successful address for an IOR; and a subsystem to add membership labels to requests and to receive updated IORs (discussed below and shown in Figure 5).

## VI. Client send request

Sending a request consists of looking up an IOR, invoking a method on the resulting stub, choosing an address, failing over to an alternate address if necessary, and sending the membership label along with the request.

### A. Lookup an IOR

An HA IOR is an IOR containing multiple instance addresses and a membership label. Each instance is capable of handling requests for that service. To obtain an IOR the client program generally does a (1) `lookup` on a NameService as shown in Figure 4. That `lookup` call is (2) mediated

by the ORB. When the client-side ORB data serializer (3) reads the IOR returned from the NameService it (4) creates a stub for the service. When the ORB creates the stub it arranges to have the stub contain a reference to a `ContactInfoList` representing the HA service addresses contained in the IOR. The stub is returned to the client code. The stub acts as a proxy for the remote service.

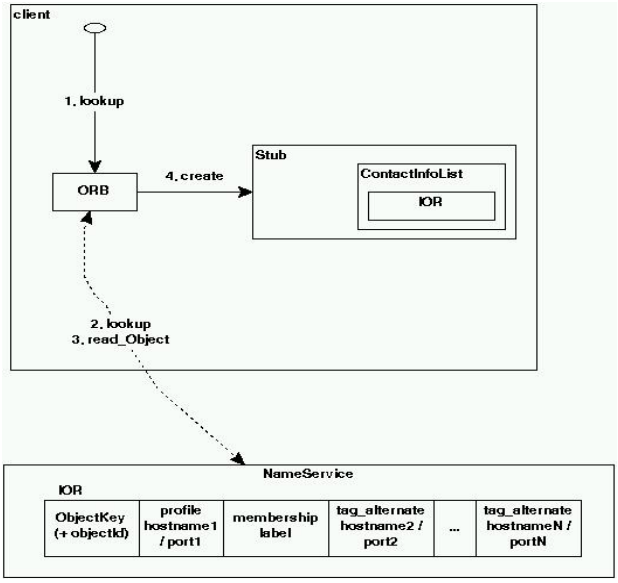


Fig. 4. Client lookup / `read_Object`

### *B. Invoke an IOR*

When the client program (1) invokes a method on the stub (shown in Figure 5) the ORB (2) mediates the call. The ORB obtains an `iterator` (3) for the HA addresses contained in the IOR, now represented by `ContactInfoList`. The list of `ContactInfo`s over which to iterate is created lazily on the first invoke on a stub by having the `ContactInfoList` call (4) `getInfo` on `IORToContactInfo` contained in the Client Group Manager (CGM). Each `ContactInfo` in the stub's `ContactInfoList` represents a hostname/port of a instance in the cluster, with the IIOPProfile hostname/port at the head of the list.

Note: `ContactInfo` allows new encoding, protocol and transport (EPT) combinations to be plugged into the ORB [3], [4], [5], [6]. For the purpose of this paper we are restricting the discussion to TCP/IP transport addressed by host/port and we are not discussing alternate EPTs. However, this failover technique works with alternate EPTs.

#### B.1 Choosing an address and the failover cache

After the `ContactInfoList` has been initialized (as shown in Figure 6) the ORB calls (1) `hasNext` and `next` on the iterator. The implementation of those methods can be a simple linear traversal of the list (the default). However, when the Client Group Manager is plugged into the ORB those methods are (2) handled by the `Failover Cache`. The cache remembers the last successful address

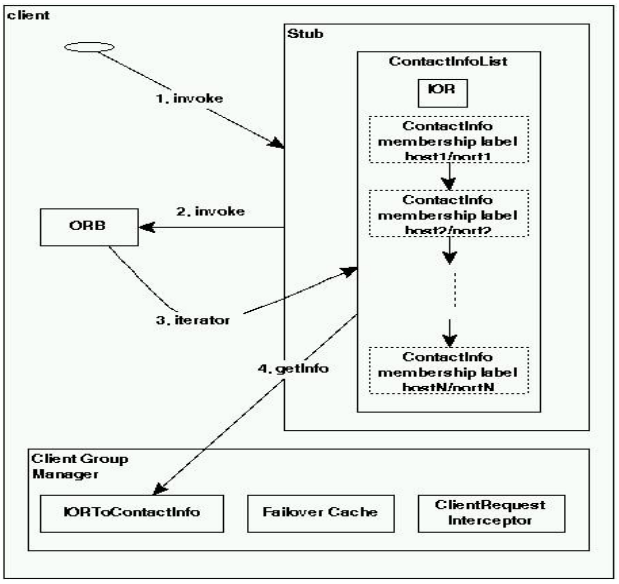


Fig. 5. Client invoke / iterator

(i.e., `ContactInfo`) and returns that in response to `next`. The ORB initializes the cache (keyed by host/port of the instance that created the IOR) with the `ContactInfo` at the head of the list - the address of the instance that created the IOR.

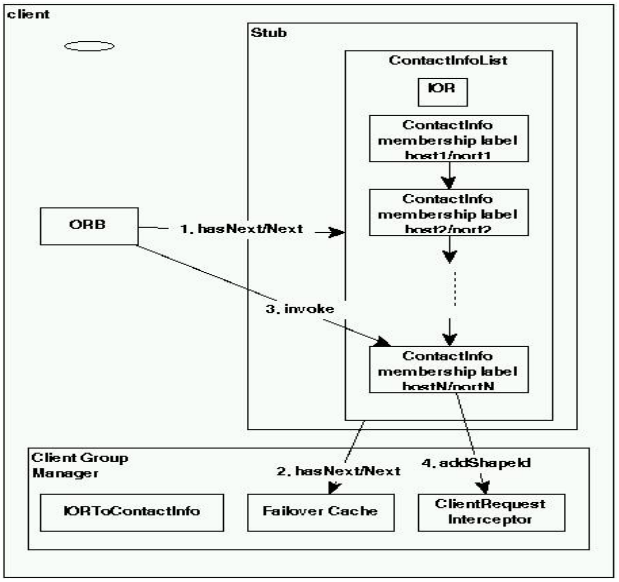


Fig. 6. Client `hasNext/Next`

If communication to an `ContactInfo` address fails, the failure information is feed back to the iterator, then the ORB transparently tries an alternate address by obtaining a different `ContactInfo` by calling `next` again. Regardless, the last successful `ContactInfo` is cached.

At the start of an invocation, `next` (2) always returns the cached `ContactInfo`. If that fails then `next` returns the `ContactInfo` in the next position with respect to where the cached `ContactInfo` is located in the list. `next` regards the list as circular, so if `ContactInfo`s in the list fail

for a particular invocation, then it will cycle back to the beginning of the list and keep trying all `ContactInfo`s in the list until one succeeds or a timeout is reached. In case of timeout, a `COMM_FAILURE` ([2] 4.12.3.5) is thrown to the client application.

The main purpose of the cache (besides avoiding the cost of retrying failed addresses) is to prevent invocations to stateful-session beans from falling back, mid-session, to previously failed instances that have come back up. That would result in the session being incorrectly spread on two instances, thus potentially corrupting the session data.

### B.2 Membership label

After the ORB obtains a `ContactInfo`, Figure 6, it (3) transfers control of the invocation to the `ContactInfo` (details of the transfer can be seen it [4]). The `ContactInfo` encodes and sends the request. Sending the request causes the `ClientRequestInterceptor` to (4) add the membership label from the IOR to the request as out-of-band data in a `ServiceContext`. Figure 7 shows the relevant parts of the message sent from the client to the server.


Fig. 7. Client request message

## VII. Cluster membership change

Independent of the server processing requests, the SGM responds to cluster membership change notifications signaled by the Group Membership Service (GMS) as shown in Figure 8. When the SGM (1) receives a notification from GMS it (2) suspends the RFM then (3) updates the membership label and (4) restarts the factories in the RFM, and finally (5) resumes the RFM. This results in the destruction of all RFM-managed POAs, which are lazily re-created by the adapter activator, typically on the first request that is handled by each RFM POA. Re-creating a POA causes the SGM `IORInterceptor` to execute `establish_components` (6), updating the POA's ORT with the (7) current list of valid address and the (8) membership label for that list. (Note: the technique of recreating POAs when responding to configuration changes relates the micro-reboot recovery strategy [10].)

### A. Membership label scope

Membership labels are scoped to each server instance (i.e., labels are *not* synchronized across all instances) so that out-of-date membership labels do not get recognized as valid on other instances.

Membership labels are UUIDs prepended with the instance's IP address. Updating a membership label means generating a new IP/UUID.

It is possible that a client could failover to an instance that has not yet received a notification of the failed instance. Since the membership label held by the client is scoped to the failed instance that membership label will be "out-of-date" compared to the failed-to instance. So the failed-to instance will return a new IOR containing out-of-date information. At some point the failed-to instance will be notified of the cluster membership change. The next invocation by the client after the change notification will then cause an up-to-date IOR to be returned to the client.

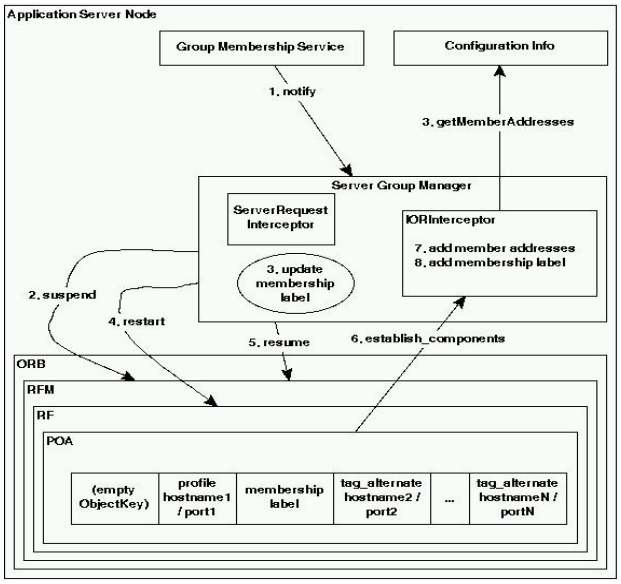


Fig. 8. Cluster membership change

It is also possible that after the client has failed over and received a new but out-of-date IOR *and* that client does not invoke on the IOR until after *all* cluster instance addresses contained in the out-of-date IOR are unreachable, then communication would fail. This is true in general, not just after failover. We do not support this use-case. We assume that a subset of instance addresses will remain available. In other words, we do not expect the addresses of cluster instances to become disjoint with initial or updated references over time. We expect instances to be added or go down and come back up, but we do not expect all instances to replaced by new instances with new addresses.

## VIII. Server request processing

When the server ORB receives a request (as shown in Figure 9 step 1) it handles that request as usual, dispatching it to the EJB layer, that dispatches to the EJB. When the response is being sent the ORB calls (2) `send_reply` on the SGM's `ServerRequestInterceptor`. The `ServerRequestInterceptor` compares the membership label sent by the client with the current membership label contained in the SGM. If they are identical then nothing further happens and the reply is sent.

If the client's membership label is out-of-date then the `ServerRequestInterceptor` uses the POA Name contained in the ObjectKey to (3) get the ReferenceFactory that initially created the reference. It then calls (4) `createReference` on that RF with the ob-

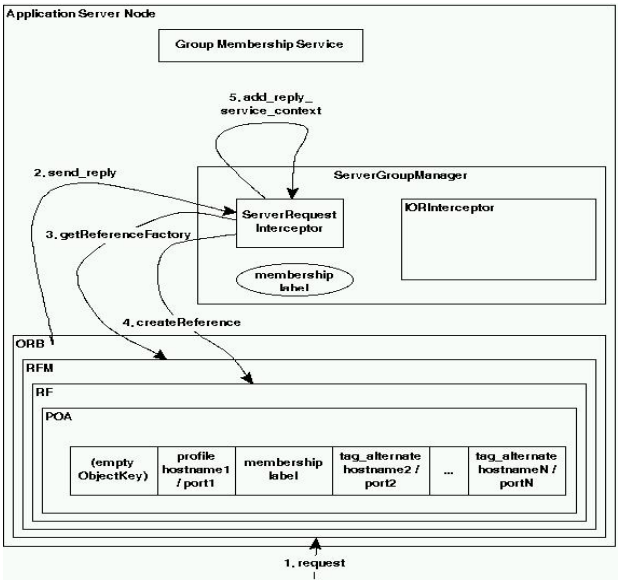


Fig. 9. Server message handling

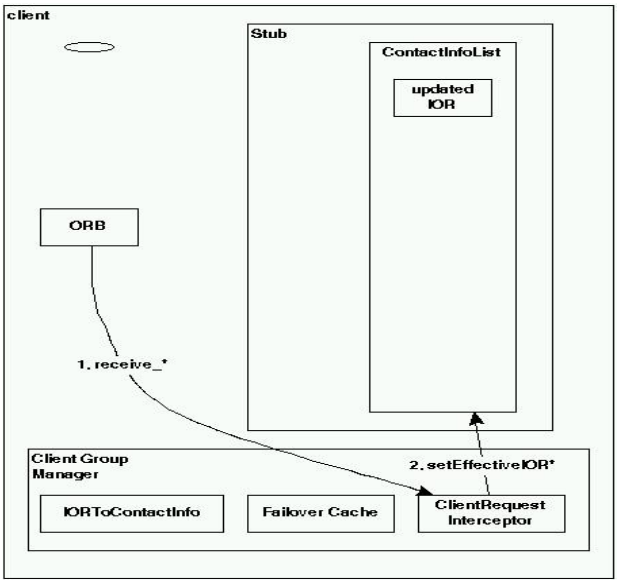


Fig. 11. Client receive reply

ject ID from the ObjectKey. This results in a new IOR, identical to the one used by the client to make the invocation, except that it has an up-to-date list of instance addresses and an up-to-date membership label. The SGM's `ServerRequestInterceptor` calls (5) `add_reply_service_context` to add the updated IOR to the reply resulting in a response message as shown in Figure 10.

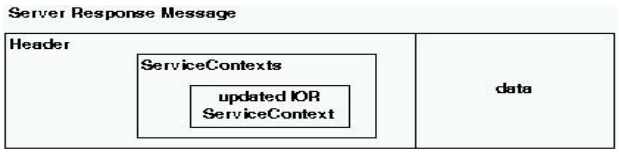


Fig. 10. Server response message

## IX. Client receive response

When the client ORB receives a reply it executes the Client Group Manager's `ClientRequestInterceptor` as shown in Figure 11 step 1. If the `ClientRequestInterceptor` does not find an updated IOR `ServiceContext` then nothing further happens at this level and the response is routed to the client.

If the `ServiceContext` is present then the original IOR is (2) replaced with the up-to-date IOR and the `ContactInfoList` is reset to its uninitialized state. This will eventually cause the list of `ContactInfo`s to be recreated with up-to-date information on the next invocation.

Note: the failover cache is *not* reset since we are receiving a response from a live instance. This means that when the next invocation occurs on the associated stub that it will use the cached address. If that address fails it will use the "next" one in the list with respect to the cached address as discussed in Section B.1.

## X. Limitations

Currently the EJB level has knowledge of sessions. We would like to use session information at the failover level to better manage the client-side failover cache. The client-side load-balancer causes client programs to be associated with particular server instances. However, this association is lost on failover. Since the CGM does not know about sessions it "sticks" to the failover instance to avoid fall-back. We would rather have the client fall-back to the original association after the session is over and when the original instance is available again.

Our failover mechanism is not portable. This means the Sun ORB must be used on both the server and client side since it contains the non-standard APIs that enable the failover system to be plugged in. One may argue that implementing the CORBA Fault Tolerance (FT) specification would provide more portability. This is true in theory but, in practice, very few ORBs provide CORBA FT. Providing CORBA FT requires changing the core ORB (or intercepting at a lower level [14]) whereas we want to provide FT as a plug-in to the core ORB.

Since our technique relies on adding up-to-date IORs to replies it does not work with `oneway` calls ([2] 3.13.1) unless using `SYNC_WITH_TARGET` ([2] 21.3.12.9). However, since our use-case is EJBs that use the RMI-IIOP programming model where all calls are synchronous, this is not an issue.

As we previously noted, if all cluster instance addresses change between the time a client obtains a reference and when a client invokes on that reference then communication will fail. Since this is not an expected use-case we have not attempted a solution to this limitation. A possible solution, should the need arise, would probably require multicast to a group address, but multicast cannot be deployed in many cases.

## XI. Performance

The primary performance measurement answers the question: what happens with "typical" applications? For example, does our failover technique have any measurable impact on a SPECjAppServer2002 [11] benchmark run? We have found that the performance impact of our failover mechanism is negligible. We have measured an IIOP system: 1. without the failover mechanism; 2. with the failover mechanism but without a cache; 3. with the failover mechanism and cache. In all cases the additional mechanisms are in the noise - invocation times are dominated by (un)marshaling followed by transport. Refer to [20] for detailed measurements.

Measurements of the performance impact of recreating POAs are still being done. We particularly want to know the impact of putting POAManagers into the holding state. For example: does waiting for a long running request to drain cause undo delay to new requests?

We do one optimization compared to Fault Tolerant CORBA ([2] 23). When a client's IOR is out-of-date an FT CORBA server returns a new IOR in a `LOCATION_FORWARD` to a request, causing the request to be remarshaled and resent. We return the new IOR in a custom `ServiceContext` in the reply, thus avoiding the remarshal and resend. However, given that server failures are not high-frequency events the different between `LOCATION_FORWARD` and a `ServiceContext` should be negligible.

## XII. Avoiding conflicts with general ORB usage

Since the ORB used by the application server is available via a JNDI call it is important to ensure that other usages of that ORB do not conflict with the SGM plug-in. For example, once a program obtains the ORB it could then obtain the `RootPOA` and find an arbitrary child POA and then create a child of that POA. One of the roles of the ReferenceFactoryManager is to use the ORB's POA in such a way as to make conflicts unlikely.

The RFM tags POAs used by EJB containers with an `EJB_Version` policy. The `EJB_Version` states that the POA may be recreated, that it must have the `PERSISTENT LifespanPolicy`, that no child POAs are allowed and that the POAManager of `EJB_Version` POAs may not manage non EJB POAs.

To ensure this, the RFM plugs in another `IORInterceptor` that is called on POA creation. The interceptor checks that the parent of the POA being created is *not* an RFM POA. If the parent is an RFM POA the interceptor throws an exception since no children are allowed.

Further, the interceptor checks that the POAManager of the POA being created is only managing either non RFM POAs or RFM POAs - intermixing is not allowed. This implies that RFM POAs cannot use the same POAManager as the `RootPOA`.

## XIII. Future work

We have not shown how EJB home reference are bound in replicated colocated NameServers. We have also not shown how clients are load-balanced across instances nor how the client-side load-balancer is updated with current cluster membership information. Future work will show how we have the client-side load-balancer register with the client-side Client Group Manager so that it receives updates on membership changes.

As noted in Section X, we want to make session information from the EJB level available to the Server Group Manager and Client Group Manager to better maintain load-balancing associations.

We are considering changing the RFM suspend mechanism from the POAManager hold_requests call to discard_requests. This would cause request to be rejected with a TRANSIENT CORBA system exception, and the clients could then retry the request. This may scale better under load, but further testing is required.

## XIV. Related Work

### A. Fault Tolerant CORBA

The Fault Tolerant (FT) CORBA (CFT) specification ([2] 23) covers much of the same ground as our IIOP failover mechanism (IFO). The primary differences are:

- CFT does replication on a per-object basis whereas we do replication on a per-IP-address basis;
- CFT enables multiple FT groups where as we only need a single group so we have simpler infrastructure;
- much of what is done in CFT is handled by other parts of our application server rather than at the IIOP/ORB level;
- CFT does not show how to plug-in or change existing ORBs to support FT whereas our work shows explicitly how to build IIOP failover in dynamic clusters using standard OMG ORB APIs extended with PEPT client-side `ContactInfo` APIs
- CFT is a broad specification whereas our work is focused on IIOP communications failover for a Java EE-compliant application server.

CFT defines an IOGR (Interoperable Object Group Reference) that uses the two-level multiple-profile structure of an IOR. Our use of IORs specifically avoids the two-level structure. We assume a single IIOP profile with alternate addresses in tagged components (where the IIOP profile host/port is the preferred address instead of IOGRs with multiple profiles and one tagged as preferred).

IFO uses a component similar to CFT's `TAG_FT_GROUP` component. However IFO has only one fault tolerance domain and a single object group so does not define data in this component corresponding to `ft_domain_id` and `object_group_id`. IFO's single field is similar to CFT's `object_group_ref_version` used to detect stale IORs. Both IFO and CFT send the information from the `TAG_FT_GROUP` component in a `FT_GROUP_VERSION ServiceContext` (or custom `ServiceContext` in IFO) on requests so the server can determine if the client's

reference is stale and return a new IOR. CFT uses `LOCATION_FORWARD` to update out-of-date IORs. IFO avoids the overhead introduced by `LOCATION_FORWARD` (i.e., resending the request) by placing the updated IOR in a custom `ServiceContext` in the reply.

CFT is involved with data replication, logging and recovery. In our system these features are handled by the EJB layer with an highly-available database independent of ORB communications failover. Similarly, CFT defines fault notification and detection whereas we register with an independent Group Membership Service (GMS) that is not part of the ORB. Since our replication is done on a per-server-instance basis, faults of interest to us are server host failures caused by hardware, software and network faults. Therefore GMS can be implemented by a simple heartbeat mechanism or by using something like JGroups [12].

CFT provides a replication manager to control object lifecycle, particularly, deciding which replica on which to deploy an object. Since, as noted above, we do replication on an IP-address basis, we assume that the EJB layer on all replicas will do identical initializations, resulting in a homogeneous cluster.

CFT supports client controlled replication through its ObjectGroupManager. IFO does not expose FT to clients. Likewise, IFO does not need CFT's PropertyManager to manage properties of object groups because there is a single group with a fixed set of properties. The CFT GenericFactory is not needed because there is no need for the FT infrastructure to create a new object replica due to a host failure in order to maintain the required `MinimumNumberReplicas` property.

CFT permits retries on `COMPLETED_MAYBE` ([2] 4.12.2), IFO does not. Consequently IFO does not use a `FT_REQUEST ServiceContext` to maintain a message log (potentially expensive in time and space) that can be used to re-transmit replies to failed operations.

CFT only considers standard IIOP. IFO works with CSIv2 ([2] 24) for replicated servers (and with other non-standard encodings, protocols and transports).

CFT defines a client model based on IOGR for portable FT. Unfortunately this model is not well supported by ORBs. IFO does not attempt to be portable.

### *B. Other FT work*

Grtner [13] provides a survey of FT issues. IFO covers its main points by: (1) Fault tolerance requires redundancy: redundancy is provided through administrative configuration of identical replicas across all instances in the cluster; (2) Safety requires detection: detection is provided by IFO's use of GMS; (3) Liveness requires correction: correction is provided by retrying failed requests transparently to the clients.

Narasimhan, Moser and Milliar-Smith [14] shows how to implement FT without modifying commercial ORBs. This is done primarily through library interposition on syscalls to intercept IIOP messages. IFO assumes the ORB may be modified.

Morgan and Ezilchelvan [15] consider different patterns of communication between a client and replicated objects, considering features like multicast may not scale across the internet, and the client cannot afford to contact each replica. In terms of this paper, IFO implements the following policies: (1) For request dissemination, a client sends to only one server replica (D1). (2) For reply collection, a client waits for one reply (C1). (3) IFO replica management is closest to passive replication (R1), but with the distinction that state replication is not handled at the CORBA layer. (4) IFO does not implement a total order policy (O1 and O2 in this paper). Instead IFO relies on a highly-available database implementation combined with executing requests at only one server replica and transparently retrying communication failures from the client side when a server replica fails.

Othman and Schmidt [16] discusses FT using CORBA FT in the context of a load balancing (LB) system. They make the observation that FT and LB are complementary and in fact can share a similar architecture based on replication: LB for handling greater loads and FT for surviving failure. We agree with the observation. In fact, our client-side LB system registers with our client IIOP Failover Manager to receive updates on the current cluster membership. Our LB design and implementation is outside of the scope of this paper.

[17] explores CORBA FT and the CORBA Component Model (CCM) [18]. It shows a component-based system that has a "home" interface that allows navigation to individual interfaces within the component. Consequently, this paper discusses two different approaches to enhancing CCM-like systems with FT: either replicate each operational interface of a component, or just replicate the home interface. They chose the latter approach. We can contrast this with our EJB-based replication system. EJB does not have a concept of an interface that can be used to find all interfaces for a single component. Instead, EJB (pre EJB 3.0) defines home interfaces that are used to control the lifecycle of the individual EJB objects. The EJB home interfaces are located through a replicated name service. To some extent this resembles the first choice discussed in the paper: replicating every object, rather than just the lookup service.

[19] is dedicated to composing adaptive middleware. Although not directly related to FT, one can view FT as a QoS-type property that affects the design of middleware. Our system uses APIs from PEPT [3], [4], [5], [6]. Previous papers on PEPT have focused on its ability to dynamically adapt to new encodings, protocols and transports. This paper focuses on using PEPTs client-side mechanisms to support failover.

## XV. Conclusion

This work focuses on providing highly-available IIOP communications. The Sun ORB has APIs that enable this failover mechanism to be plugged into the ORB without the ORB needing to be extended to support failover. By using an adaptable middleware architecture we can deliver our ORB into the Sun Java SE implementation without

the failover mechanism while, using the same code base, supporting failover in the application server by plug-ins.

We implemented and shipped an IIOP failover mechanism in Sun Java System Application Server 7.1 and 8.1 Enterprise Editions that worked for static clusters [20]. The current work extends failover to work in dynamic clusters. Initial HA and performance testing looks promising. The current work will be used in the Sun Java Application Server 9.1 Enterprise Edition.

## References


[1] Sun Microsystems, Inc. Enterprise JavaBeans http://java.sun.com/products/ejb/ docs.html

[2] Object Management Group: CORBA/IIOP Specification v3.0.3. (2004) http://www.omg.org/cgi-bin/doc?formal/04-03-12

[3] Carr, H.: Server-side Encoding, Protocol and Transport Extensibility for Remoting Systems. *Proceedings of the Second International Conference on Service Oriented Computing*. New York (November 2004) 329–334

[4] Carr, H.: Client-side Encoding, Protocol and Transport Extensibility for Remoting Systems. *Proceedings of the International Conference on Communications in Computing*. Las Vegas (June 2004) 51–57

[5] Carr, H.: PEPt - A Minimal RPC Architecture. *OTM Confederated International Workshops HCI-SWWA, IPW, JTRES, WORM, WMS and WRSM 2003 Proceedings*. Catania, Sicily (November 2003) 109–122

[6] Carr, H.: One-Page PEPt. *Middleware 2003 Workshop Proceedings*. Rio de Janeiro, Brazil. (June 2003)

[7] Object Management Group Java Language Mapping to OMG IDL, version 1.3 http://www.omg.org/cgi-bin/doc?formal/03-09-04

[8] Object Management Group OMG IDL to Java Language Mapping, version 1.2 http://www.omg.org/cgi-bin/doc?formal/02-08-05

[9] Sun Microsystems, Inc. Java 2 Platform Enterprise Edition Specification, v1.4 http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf (November 2003)

[10] Candea, G., Kawamoto, S., Fujiki, Y., Friedman, G., Fox, A.: Microreboot – A Technique for Cheap Recovery *Proc. 6th Symposium on Operating Systems Design and Implementation*. San Francisco (December 2004)

[11] Standard Performance Evaluation Corporation *SPECjAppServer2002* http://www.spec.org/jAppServer2002/

[12] Ban, B., et. al.: JGroups - A Toolkit for Reliable Multicasting http://www.jgroups.org/

[13] Grtner, F.C.: Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments *ACM Computing Surveys*. **31:1** (March 1999)

[14] Narasimhan, P., Moser, L. E, Melliar-Smith, P. M.: Gateways for Accessing Fault Tolerance Domains *Middleware 2000*. LNCS **1795** 88–103

[15] Morgan, G., Ezilchelvan, P. D.: Policies for using Replica Groups and their effectiveness over the Internet *Proceedings of NGC 2000 on Networked Group communication*. (2000)

[16] Othman, O, Schmidt, D.C.: Issues in the design of Adaptive Middleware Load Balancing *The First Workshop on Optimization of Middleware and Distributed Systems*. PLDI (2001)

[17] de Man, D., Millian, R., Weddam, M., Gokhale, A., Yajnik, S.: "Transparent Fault Tolerance for CORBA based Distributed Components" *OOPSLA 2000 Companion*. (2000)

[18] Object Management Group CORBA Component Model, V3.0 (2004) http://www.omg.org/cgi-bin/doc?formal/02-06-65

[19] The Association of Computing Machinery *Communications of the ACM*. (June 2002)

[20] Carr, H.: IIOP and SOAP Failover in Static Clusters to appear in the International Conference on Communications in Computing. Las Vegas (June 2005)