
The Object Reference Template

Revised Submission

IBM
Inprise Corporation
IONA Technologies
Object Oriented Concepts
Sun Microsystems

February 5, 2001
Document number orbos/01-01-04

© Copyright 2000,2001 by IBM
© Copyright 2000,2001 by Inprise Corporation
© Copyright 2000,2001 by IONA Technologies
© Copyright 2000,2001 by Object Oriented Concepts
© Copyright 2000,2001 by Sun Microsystems

The submitting companies listed above have all contributed to this submission. These companies recognize that this draft joint submission is the joint intellectual property of all the submitters, and may be used by any of them in the future, regardless of whether they ultimately participate in a final joint submission.

The companies listed above hereby grant a royalty-free license to the Object Management Group, Inc. (OMG) for world-wide distribution of this document or any derivative works thereof, so long as the OMG reproduces the copyright notices and the below paragraphs on all distributed copies.

The material in this document is submitted to the OMG for evaluation. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

NOTICE

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR PARTICULAR PURPOSE OR USE. In no event shall The Object Management Group or any of the companies listed above be liable for errors contained herein or for indirect, incidental, special, consequential, reliance or cover damages, including loss of profits, revenue, data or use, incurred by any user or any third party. The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Right in Technical Data and Computer Software Clause at DFARS 252.227.7013 OMG[®] and Object Management are registered trademarks of the Object Management Group, Inc. Object Request Broker, OMG IDL, ORB, CORBA, CORBAfacilities, CORBAservices, and COSS are trademarks of the Object Management Group, Inc. X/Open is a trademark of X/Open Company Ltd.

Table Of Contents

Preface	1
1.1 Introduction	1
1.2 Submission Contacts	1
1.3 Conventions	2
1.4 Guide to the submission	2
1.5 Items not addressed in this submission	3
1.6 Proof of Concept	3
1.7 References	3
Response to RFP Requirements	5
2.1 Mandatory Requirements	5
2.1.1 API for Object Reference Template	5
2.1.2 Object Reference Template Refresh	5
2.1.3 Server ID	6
2.1.4 Relationship with IORInterceptor	6
2.1.5 Endpoints	6
2.1.6 Security and Transaction services	6
2.2 Optional Requirements	6
2.2.1 Additional Resource Management APIs	6
2.3 Issues to be Discussed	7
Goals and Rationale	9
3.1 Goals	9
3.2 Rationale	10
Object Reference Template	11
4.1 An Abstract Model for Object Adapters	11
4.1.1 Adapter Names	11
4.1.2 Adapter States	12
4.1.3 Adapter Managers	13
4.1.4 Adapter State Changes	13
4.2 Object Reference Template	14
4.2.1 Definition	14
4.2.2 The ObjectReferenceFactory abstract valuetype	15
4.2.3 The ObjectReferenceTemplate abstract valuetype	15
4.2.4 Extending PortableInterceptor::IORInfo	16
4.2.5 Extending PortableInterceptor::IORInterceptor	18
4.2.6 Extending PortableInterceptor::ServerRequestInfo	21
4.2.7 Template Refresh	22
4.3 Server Configuration	24
4.3.1 Server ID	24
4.3.2 Server IIOP Endpoint	24
4.3.3 Starting Servers with no persistent server support	25

Conformance	27
Changes to Existing Specifications	29
6.1 Changes to CORBA	29
6.2 Changes to Services	30
Consolidated IDL	31

Preface

1

1.1 Introduction

IBM, Inprise, IONA Technologies, Object Oriented Concepts, and Sun Microsystems are pleased to submit to the Object Reference Template RFP orbos/00-09-30.

This proposal describes the changes to CORBA 2.4 and the Portable interceptors specification required to support the goals of the RFP, which is to enable the construction of portable server activation frameworks.

1.2 Submission Contacts

Randy Fox
IBM, Inc.
11400 Burnet Road
Austin, Texas 78758
Phone: 1 512 838 2310
Fax: 1 512 838 1032
Email: randyfox@us.ibm.com

Vijaykumar Natarajan
Inprise Corporation
951 Mariner's Island Blvd.
San Mateo CA 94404
Phone: 650 358 2412
Fax: 650 358 3098
Email: vijayn@inprise.com

Manfred Koethe
IONA Technologies Inc
200 West Street
Waltham, MA 02451
Phone: 1 781 902 8557
Fax: 1 781 902 8001
Email: manfred.koethe@iona.com

Michi Henning
Object Oriented Concepts
44 Manning Rd.
Billerica, MA 01821
Phone: +61 7 3324 9633
Fax: +61 7 3324 9799
Email: michi@ooc.com.au

Ken Cavanaugh
Sun Microsystems Inc
MS UCUP02-201.
901 San Antonio Road
Palo Alto, CA 94303
Phone: 1 408 863 3489
Fax: 1 408 863 3195
Email: ken.cavanaugh@sun.com

1.3 Conventions

IDL appears in this font.

1.4 Guide to the submission

Chapter 2 restates the requirements from the RFP and discusses how they are addressed in this submission

Chapter 3 describes the goals and the design rationale for this submission.

Chapter 4 is the main part of the submission. It describes the Object Reference Template and associated mechanisms.

Chapter 5 defines the conformance criteria.

Chapter 6 describes the changes to other specifications required by this specification.

Chapter 7 gives the consolidated IDL for this submission.

1.5 *Items not addressed in this submission*

This submission address all of the requirements raised in the RFP.

1.6 *Proof of Concept*

This submission is based on the experience of the submitters in building server activation frameworks using an approach similar to the IDL defined in this specification.

The IDL has been compiled against the current IDL compiler shipped in JDK 1.3.

1.7 *References*

[CORBA24] *CORBA Specification, version 2.4*, OMG, November 2000, formal/00-10-33
<http://cgi.omg.org/cgi-bin/doc?formal/00-10-33>

[PI] *Interceptors FTF Final Draft of CORBA Core Chapters, Chapter 21*, ptc/00-08-06
<http://cgi.omg.org/cgi-bin/doc?ptc/00-08-06>

Response to RFP Requirements

2

This section presents the RFP requirements described in orbos/00-09-30, together with our response.

Note – The RFP used the term LASD (location/activation service daemon) while the submission uses the more standard term IMR (implementation repository). The LASD terminology is only used here for consistency with the RFP.

2.1 Mandatory Requirements

2.1.1 API for Object Reference Template

Proposals shall provide an API for an object reference template that can be used to directly create object references. The object reference template must be marshallable.

This API is defined in Section 4.2.2, “The ObjectReferenceFactory abstract valuetype”, on page 15.

2.1.2 Object Reference Template Refresh

Proposals shall provide an API that allows an object reference template to be refreshed when a server is activated.

This issue is discussed in Section 4.2.7, “Template Refresh”, on page 22. No special API is necessary other than the basic Object Reference Template API defined in Section 4.2.1, “Definition”, on page 14.

2.1.3 Server ID

Proposals shall provide an API that specifies some form of Server ID that provide unambiguous identification for a server that is independent of server activations.

This is defined in Section 4.3.1, “Server ID”, on page 24.

2.1.4 Relationship with IORInterceptor

Proposals shall address the relationship between the object reference template and the IORInteceptor defined in Portable Interceptors.

A number of extensions to the Portable Interceptors API are defined in Section 4.2.5, “Extending PortableInterceptor::IORInterceptor”, on page 18.

2.1.5 Endpoints

Proposals shall provide an API that allows servers to be started on a particular endpoint. This is needed for starting an LASD that listens on a fixed endpoint.

This is discussed in Section 4.3.2, “Server IIOP Endpoint”, on page 24.

2.1.6 Security and Transaction services

Proposals shall address the requirements of the Security and Transaction services.

This proposal may be useful for implementing certain aspects of Security and Transactions in a more portable fashion. Nothing in this specification creates any problems for either service.

2.2 Optional Requirements

2.2.1 Additional Resource Management APIs

Proposals may address additional resource management requirements, such as a POA destroy interceptor that can be used to manage information registered with an LASD, and POAManager state change notification.

This submission includes a mechanism for detecting state changes in an Object Adapter described in Section 4.2.5, “Extending PortableInterceptor::IORInterceptor”, on page 18. The POAManager is expressed as a more abstract Adapter Manager here, with an API described in Section 4.2.4, “Extending PortableInterceptor::IORInfo”, on page 16 and Section 4.2.5, “Extending PortableInterceptor::IORInterceptor”, on page 18.

2.3 *Issues to be Discussed*

Most existing ORBs already provide some kind of server activation framework. Proposals should discuss how a persistent server may run outside of an ORB's existing activation framework.

This is addressed in Section 4.3.3, "Starting Servers with no persistent server support", on page 25.

3.1 Goals

This submission is intended to make it possible to write a portable server activation framework that can be used with any conformant ORB. There is no intent here to make a standard server activation framework that all ORBs must support. The application space in which this is useful is in building application servers that are based on CORBA. An application server may have complex requirements for scalability and reliability that require a carefully designed server activation framework. This submission defines sufficient mechanisms in conjunction with other CORBA specifications to allow a high performance portable server activation framework to be designed for a particular application server.

Most of the APIs that are needed for constructing a server activation framework are already available in the standard CORBA APIs, particularly in the Portable Object Adaptor and Portable Interceptors. This submission addresses a few small additions that are needed:

- An object reference template that can be used to delegate the capability of creating an object reference to a remote client.
- Extensions to the IOR interceptor that allow monitoring of object adapter state changes.
- Extensions to the Portable Interceptors APIs for the Object Reference Template.
- A mechanism that can be used to specify a particular endpoint or group of endpoints on which a server will listen for requests.
- A mechanism that allows an immutable identifier to be associated with a particular server.

3.2 *Rationale*

It is well established that many CORBA systems support both transient and persistent object references. A persistent object reference is one that remains valid even if its implementation is not always available. Typically this is accomplished through the use of some kind of Implementation Repository (IMR). The IMR is always running, and always listening for invocations on the same communication endpoint. An IMR can take advantage of the location forward mechanism provided in GIOP (or other proprietary mechanisms) to communicate an updated IOR back to the client. The client then invokes the updated IOR. This can be used to support a persistent server that listens on different endpoints each time it is activated.

Supporting such a mechanism requires that the IMR can map incoming requests to the correct IOR for sending a location forward. This IOR must have the correct profiles and components to support the services required for the particular object implementation. The Portable Object Adapter and Portable Interceptors specifications provide standard APIs that can create and manipulate the components associated with an object reference.

The Portable Interceptor specification has an implicit concept of a template used to create object references in the IOR interceptor. This submission provides interfaces that expose and standardize this template for the purpose of supporting activation frameworks for persistent servers.

It is also the intent that this submission support scalable IMR implementations. One particular area where this is important is in the amount of persistent state information that the IMR requires. At a minimum, an IMR must have some way of storing enough information so that it can start a server when required. For example, in Java the class name, arguments, and class path may be required. We want to ensure that this submission does not require storing more persistent data than this minimum requirement. In particular, this submission does not require an implementation to store persistent state for each CORBA object or POA in a server managed by an IMR.

4.1 An Abstract Model for Object Adapters

Using the IORInterceptor to support the object reference template imposes certain requirements on Object Adapters. While the POA is the only (current) standard object adapter, we do not wish to impose the POA architecture on all possible proprietary object adapters. Consequently we will focus on the abstract properties that are required, and discuss how these map to the particular case of the POA.

We have the following requirements:

- We need an Adapter name so that different instances of a particular object adapter may be identified.
- Object adapters typically have some kind of request processing state to indicate whether the adapter is currently accepting, rejecting, or performing some other kind of action on incoming requests. We need to include some representation of adapter instance state so that a server activation framework built on the object reference template can correctly process requests as the adapter instances states change.
- If an object adapter supports large numbers of adapter instances, reporting state changes that affect a number of adapter instances simultaneously could be expensive in the amount of data required. The POA has the concept of an adapter manager (the POAManager) that controls the state of a number of POA instances. We can abstract this as an abstract adapter manager and use this for reporting relevant state changes.

4.1.1 Adapter Names

If an Object Adapter supports multiple adapter instances, we need some kind of adapter name to distinguish the instances. For this purpose, we define an adapter name as a sequence of strings. Several interpretations of an adapter name are possible:

- If the Object Adapter supports only a single instance, a fixed name can be used.

- If the namespace for an Object Adapter is flat, sequences of length 1 can be used.
- If the namespace is hierarchical (e.g. the POA), a more complex name sequence can be used.

In the case of the POA, the adapter name shall be the sequence of names starting with the root POA that is required to reach the POA using the `find_POA` call. The name of the root POA is the empty sequence.

4.1.2 Adapter States

Object adapters may be in one of several states that describe how the adapter behaves when a new request is dispatched to the adapter:

HOLDING

The request is held off temporarily in response to a transient resource limit or a application program request. An IMR could either choose to forward the request to the server and let the server hold it off, or else to hold off the request at the IMR until the state changes.

ACTIVE

The request is dispatched to the servant and processed. An IMR should forward the request to the server in this case.

DISCARDING

The request is discarded. This is indicated to the client with some kind of error. An IMR could either forward the request to the server, or else reject the request directly. The POA spec requires that a `TRANSIENT/1` system exception be returned to the client in this case.

INACTIVE

The request is discarded. The adapter is in the process of shutting down, and will eventually end up in the **NON_EXISTENT** state. An IMR could reject the request directly, typically with an `OBJ_ADAPTER/1` error.

NON_EXISTENT

The adapter has been destroyed. The IMR should attempt to reactivate the server and adapter as necessary to satisfy the request. The IMR should hold off the request until the adapter becomes active again.

In the case of the POA, **HOLDING**, **ACTIVE**, **DISCARDING**, and **INACTIVE** map to the same named states of the POAManager. **NON_EXISTENT** does not map directly to a particular POAManager state, but is used to indicate that a POA has been destroyed. A POA whose state is **INACTIVE** will transition to state **NON_EXISTENT** after the destruction process has completed.

While non-POA adapters may have different detailed states than the POA, it should be possible to map other adapter's states onto a subset of the above states.

Note – The exact state machine implemented by a particular object adapter is not relevant to this specification. What matters is the behavior of the IMR for request to an object adapter in a particular state.

4.1.3 Adapter Managers

Some object adapters have a concept of a group of adapters that undergo state transitions together. In such cases it is useful to capture the grouping abstractly. We define the *adapter manager* to represent this grouping. The only standard attribute of the adapter manager is the adapter manager id, which is an opaque id. This ID serves to distinguish different adapter manager instances, and to associate an adapter manager instance with its adapter instances. The adapter manager id is only locally significant within the ORB instance that defines the adapter manager. The id is transient, and can be compared for equality within the defining ORB instance. All adapter instances that share the same adapter manager must have the same adapter manager id.

Use of an adapter manager allows state transitions for all adapters managed by the same adapter manager to be efficiently reported. The only assumption made about the semantics of an adapter manager is that a state change reported for an adapter manager is reflected in all adapter instances managed by the adapter manager.

In the case of the POA, the POAManager is an adapter manager.

4.1.4 Adapter State Changes

Some adapters may support mechanisms independent of the adapter manager for changing states. In such cases, a means needs to be provided for reporting the state changes.

In the case of the POA, a subtree of POAs may all transition to the **NON_EXISTENT** state as a result of the POA::destroy call.

4.2 Object Reference Template

4.2.1 Definition

The Object Reference Template is defined in IDL as an abstract valuetype.

Note – Defining the Object Reference Template as an abstract valuetype allows a variety of implementations to be created and says little about the state that must be marshalled as part of the template. This does not permit templates to be exchanged between different ORB implementations. This is explicitly not a goal of the RFP, which calls only for creating interfaces for portable server activation frameworks including some kind of activation daemon (usually called an Implementation Repository, or IMR), not interoperation between IMRs and ORBs written by different vendors.

An object reference template is associated with an object adapter. Typically the template is created when the object adapter is created, used within the adapter to create object references, and destroyed when the adapter is destroyed. Different adapters may support very different styles of object creation.

The object reference template is defined as follows:

```

module PortableInterceptor {
    typedef string ServerId ;
    typedef string ORBId ;
    typedef CORBA::StringSeq AdapterName ;
    typedef CORBA::OctetSeq ObjectId ;

    abstract valuetype ObjectReferenceFactory {
        Object make_object( in string repositoryId, in ObjectId id ) ;
    };

    abstract valuetype ObjectReferenceTemplate :
        ObjectReferenceFactory {
            readonly attribute ServerId server_id ;
            readonly attribute ORBId orb_id ;
            readonly attribute AdapterName adapter_name;
        };

    typedef sequence<ObjectReferenceTemplate>
        ObjectReferenceTemplateSeq;
};

```

The **ObjectReferenceFactory** valuetype provides the capability to create new object references, while the **ObjectReferenceTemplate** valuetype extends the factory capability with the identity of the template. This division is convenient because the **current_factory** attribute in IORInfo (see Section 4.2.4, “Extending PortableInterceptor::IORInfo”, on page 16) only requires the capability to create an object reference, while the **adapter_template** attribute (also in Section 4.2.4) also requires identity information.

Note – This separation is also useful if a user of the factory valuetype implements their own concrete valuetype that extends the **ObjectReferenceFactory** valuetype. Since the only requirement of the object adapter is that **current_factory** supports **make_object**, there is no reason to require such a custom factory to support the identity operations required by the template.

4.2.2 *The ObjectReferenceFactory abstract valuetype*

The **ObjectReferenceFactory** provides only the capability to create an object reference. Note that a factory is immutable: after it has been created, it cannot be modified.

Also, note that it is possible to create a concrete valuetype (unknown to the ORB implementation) that subclasses the **ObjectReferenceFactory** valuetype, and to use this factory in the IOR interceptor as **current_factory** (see Section 4.2.4.4, “current_factory”, on page 17). In such cases, the implementation must either be immutable after it is created, or the implementation must not change the behavior of **make_object**. Failure to observe this requirement may result in undefined behavior.

4.2.2.1 *make_object*

make_object creates an Object Reference from this factory using the given repository ID and object ID. All object references created from the same factory share the same profiles and tagged components in their profiles. They may also share some of the data in their object keys, but this is not required.

4.2.3 *The ObjectReferenceTemplate abstract valuetype*

The **ObjectReferenceTemplate** extends the **ObjectReferenceFactory** with the identity of the object adapter. Note that the template, like the factory, is immutable: after it has been created, it cannot be modified.

4.2.3.1 *server_id*

The value of the **server_id** attribute is the value that was passed into the **ORB::init** call (see Section 4.3.1, “Server ID”, on page 24) using the **-ORBServerId** argument when the ORB was created.

4.2.3.2 *orb_id*

The value of the **orb_id** attribute is the value that was passed into the **ORB::init** call.

In Java, this is accomplished using the **-ORBid** argument in the **ORB.init** call that created the ORB containing the object adapter that created this template. What happens if the same ORBid is used on multiple **ORB::init** calls in the same server is currently undefined.

4.2.3.3 *adapter_name*

The **adapter_name** attribute defines a name for the object adapter that services requests for the invoked object.

4.2.4 *Extending PortableInterceptor::IORInfo*

All object adapter implementations provide some mechanism for creating object references. The construction of the object reference is influenced by all of the applicable server-side policies, which are used while assembling the tagged components required for the object reference. The IOR interceptors also influence the tagged components through the **IORInfo::add_component** and **IORInfo::add_component_to_profile** methods. After all of this construction has completed, the adapter conceptually has a template that can be used to create object references. We will refer to this template as the **adapter template**.

For example, in the POA, after **POA::create_POA** method has completed, there is a complete template in the POA that will be used to create individual object references when **create_reference** or any other method is called that needs to create an object reference.

We add the following methods to the **PortableInterceptor::IORInfo** interface:

```
typedef long AdapterManagerId;

typedef short AdapterState ;
const AdapterState  HOLDING      = 0 ;
const AdapterState  ACTIVE       = 1 ;
const AdapterState  DISCARDING   = 2 ;
const AdapterState  INACTIVE     = 3 ;
const AdapterState  NON_EXISTENT = 4 ;

local interface IORInfo {
    ...
    readonly attribute AdapterManagerId manager_id;
    readonly attribute AdapterState state;
    readonly attribute ObjectReferenceTemplate adapter_template ;
    attribute ObjectReferenceFactory current_factory ;
};
```

4.2.4.1 *manager_id*

The **manager_id** attribute provides an opaque handle to the manager of the adapter. This is used for reporting state changes in adapters managed by the same adapter manager.

4.2.4.2 *state*

The **state** attribute returns the current state of the adapter. This must be one of HOLDING, ACTIVE, DISCARDING, INACTIVE, NON_EXISTENT.

4.2.4.3 *adapter_template*

The **adapter_template** attribute provides a means to obtain an object reference template whenever an ior interceptor is invoked. There is no standard way to directly create an object reference template. The value of **adapter_template** is the template created for the adapter policies and IOR interceptor calls to **add_component** and **add_component_to_profile**. The value of the **adapter_template** attribute is never changed for the lifetime of the object adapter.

4.2.4.4 *current_factory*

The **current_factory** attribute provides access to the factory that will be used by the adapter to create object references. **current_factory** initially has the same value as the **adapter_template** attribute, but this can be changed by setting **current_factory** to another factory. All object references created by the object adapter must be created by calling the **make_object** method on **current_factory**.

The value of the **current_factory** attribute that is used by the adapter can only be set during the call to the **components_established** method.

Note – Since **current_factory** is a writable attribute, and we support multiple IOR interceptors, two or more IOR interceptors could write to **current_factory**. In this case, the order is likely to matter to the application. It is essentially impossible to define the order in which interceptors will be invoked in any standard, universally applicable manner. However, particular ORB implementations may choose to administratively order the interceptors in some manner to solve ordering problems.

4.2.5 Extending *PortableInterceptor::IORInterceptor*

We extend **PortableInterceptor::IORInterceptor** as follows:

```
local interface IORInterceptor {
    ...
    void components_established( in IORInfo info );
    void adapter_manager_state_changed( in AdapterManagerId id,
        in AdapterState state );
    void adapter_state_changed( in ObjectReferenceTemplateSeq
        templates, in AdapterState state );
};
```

A server side ORB calls the **establish_components** method on all registered IOR interceptors when it is assembling the list of components that will be included in the profile(s) in an object reference.

In the case of the POA, these calls are made each time **POA::create_POA** is called. In other adapters, these calls would typically be made when the adapter is initialized. The adapter template is not available at this stage since information (the components) needed in the adapter template is being constructed.

4.2.5.1 *components_established*

After all of the **establish_components** methods have been called, the **components_established** methods are invoked on all registered IOR interceptors. The adapter template is available at this stage. The **current_factory** attribute may be get or set at this stage.

Any exception that occurs in **components_established** is returned to the caller of **components_established**. In the case of the POA, this causes the **create_POA** call to fail, and an OBJ_ADAPTER exception with a minor code of TBD is returned to the invoker of **create_POA**.

Note – This is different from the usual handling of exceptions as currently specified in [PI] in 21.5.2.1 for **establish_components**. However, it is important to be able to stop the creation of a POA instance if it is not possible to perform the registration with the IMR. This requires throwing an exception in **components_established**.

4.2.5.2 *adapter_manager_state_changed*

Any time the state of an adapter manager changes, the **adapter_manager_state_changed** method is invoked on all registered IOR interceptors.

If a state change is reported through **adapter_manager_state_changed**, it is not reported through **adapter_state_changed**.

4.2.5.3 *adapter_state_changed*

Object adapter state changes are reported to this method any time the state of one or more adapters changes for reasons unrelated to adapter manager state changes. The templates argument identifies the object adapters that have changed state by the template ID information. The sequence contains the adapter templates for all object adapters that have made the state transition being reported.

4.2.5.4 Method Validity

The following table defines the validity of each attribute or operation in **IORInfo** in the methods defined in the **IORInterceptor**:

Table 4-1 IORInfo validity

	establish_components	components_established
get_effective_policy	yes	yes
add_component	yes	no
add_component_to_profile	yes	no
read manager_id	yes	yes
read state	yes	yes
read adapter_template	no	yes
read current_factory	no	yes
write current_factory	no	yes

If an illegal call is made to an attribute or operation in **IORInfo**, the **BAD_INV_ORDER** system exception is raised with a standard minor code value of 14.

4.2.6 Extending *PortableInterceptor::ServerRequestInfo*

PortableInterceptors::ServerRequestInfo is extended as follows:

```
local interface ServerRequestInfo {
    ...
    readonly attribute ServerId server_id ;
    readonly attribute ORBId orb_id ;
    readonly attribute AdapterName adapter_name;
};
```

The **server_id**, **orb_id**, and **adapter_name** attributes may be read from **receive_request**, **send_reply**, **send_exception**, and **send_other**. These attributes may not be read during calls to **receive_request_service_contexts**.

4.2.6.1 *server_id*

The value of the **server_id** attribute is the value that was passed into the **ORB::init** call (see Section 4.3.1, “Server ID”, on page 24) using the **-ORBServerId** argument when the ORB was created.

4.2.6.2 *orb_id*

The value of the **orb_id** attribute is the value that was passed into the **ORB::init** call.

In Java, this is accomplished using the **-ORBId** argument in the **ORB.init** call that created the ORB containing the object adapter that created this template. What happens if the same ORBId is used on multiple **ORB::init** calls in the same server is currently undefined.

4.2.6.3 *adapter_name*

The **adapter_name** attribute defines a name for the object adapter that services requests for the invoked object. In the case of the POA, the **adapter_name** is the sequence of names from the root POA to the POA that services the request. The root POA is not named in this sequence.

Note – We have discussed removing these attributes from the **ServerRequestInfo**. I think it is better not to do this, since the presence of these attributes allows an interceptor to know what part of the server is dispatching the request. This can be useful for applications such as visualization of server activity, and possible also measuring activity to perform some action.

4.2.7 Template Refresh

Note – This section is not normative. It is present only as an example of some of the implementation issues raised by this submission.

It is important to understand that not all parts of a server will necessarily initialize at the same time when the server is started. For example, a server may use POA adapter activators to activate persistent POAs on demand from invocations. In the case of IIOP, the exact port needed for a POA activated by an adapter activator may not be known in advance of the activation of the POA. Thus, any information that an IMR may have about the newly activated POA may be out-of-date and unusable.

When an invocation from a client is received by the IMR, the IMR must obtain a valid object reference template that it can use to construct a new object reference. It turns out that no special support is necessary to accomplish this. The fact that **PortableInterceptor::ObjectReferenceTemplate** provides access to **server_id**, **orb_id**, and **adapter_name** means that a framework can be built that registers templates based on their adapter names. The framework does need a way for the IMR to indicate to a server that a template for a particular object adapter is required. However, this can be implemented using only existing interfaces in the POA case.

In more detail, a typical implementation of this might work as follows:

1. When a server starts up, it registers its existence with the IMR so that the IMR can monitor the server lifecycle.
2. Each ORB that is created in the server registers an ORB callback object with the server in the IMR. This ORB callback supports a method that takes a **PortableInterceptor::AdapterName** as an argument. This method will cause the named object adapter to be activated if it is not already activated.
3. Each persistent adapter instance that is created in the server registers its adapter template with its IMR. The IMR returns a factory (for example, it could return its own adapter template). The IMR's factory is then written to the current factory in the adapter instance. The IMR's factory is then used whenever the adapter instance creates an object reference. Note that **IORInfo::get_effective_policy** can be used to determine whether an adapter instance supports persistent object references in the POA case.

4. During the registration of the server's template, the IMR must create a corresponding factory as noted in step 3. This can be done in at least a couple of ways. One way is to encode the **server_id/orb_id/adapter_name** into the name of a POA created in the IMR to correspond to the server's template. This has the advantage of reusing all of the POA machinery in the IMR for the IMR's functions (e.g. we can use the adapter activator and servant locator as needed). Another possibility is to create a custom **ObjectReferenceFactory** (that is, an implementation of a concrete value type that extends the standard **ObjectReferenceFactory**) that can encode the **server_id/orb_id/adapter_name** in the object ID. This has the advantage of not creating a lot of POAs in the IMR, but cannot take advantage of the POA machinery.
5. When the IMR receives a request, it obtains the **adapter_name**, the **orb_id**, and the **server_id** from the target object key in the request. This can be done using either of the methods described in step 4 (either we get the server IDs from the IMR POA name, or else from a specially encoded object ID). The IMR uses the (**server_id/orb_id/adapter_name**) information to see whether the adapter has registered since the server was started.
6. If the **server_id** is not registered with the IMR, the IMR returns an error to the client. This could happen if a client invoked an object reference for a server that was administratively destroyed.
7. If a server corresponding to the **server_id** is not currently running, the IMR starts the corresponding server and waits for the server registration.
8. If an ORB corresponding to the **orb_id** is not registered with the IMR for the server containing the ORB, the IMR waits for the ORB to register its callback object.
9. If an adapter instance corresponding to the **adapter_name** is not registered with a corresponding object reference template, the IMR invokes the ORB's callback object with the adapter name.
10. The callback object uses some mechanism to activate the object adapter, which then registers with the IMR. In the case of the POA, **POA::find_POA** can be used to activate the adapter.
11. The IMR now has a valid template and can complete the request by creating a new object reference and causing this to be sent in an object forward or location forward response.

4.3 Server Configuration

The concept of a server is not well-defined in the CORBA architecture today. However, we need to say a few things about servers in this proposal. Basically, a server is one or more implementations of CORBA Objects that can be activated together. This can include activation by an IMR or by some other means. A single instance of a server may contain multiple ORB instances, each of which may contain multiple object adapters. Typical implementations of servers include a process for C++ and a JVM in a process for Java.

Note – Whenever an **ORB::init** argument of the form **-ORBxxx** is specified, it is understood that the argument may be represented different ways in different languages. For example, in Java **-ORBxxx** is equivalent to a property named **org.omg.CORBA.ORBxxx**.

4.3.1 Server ID

A Server ID must uniquely identify a server to an IMR. This proposal only requires unique identification using a string of some kind. We do not intend to make more specific requirements for the structure of a server ID.

The server ID may be specified by an **ORB::init** argument of the form

-ORBServerId

The value assigned to this property is a **string**. All templates created in this ORB will return this server ID in the **server_id** attribute.

It is required that all ORBs in the same server share the same server ID. Specific environments may choose to implement **-ORBServerId** in ways that automatically enforce this requirement.

For example, the **org.omg.CORBA.ServerId** system property may be set to the server ID in Java when a Java server is activated. This system property is then picked up as part of the `ORB.init()` call for every ORB created in the server.

4.3.2 Server IIOP Endpoint

The server endpoint information is passed into **ORB::init** by an argument of the form

-ORBListenEndpoints <endpoints>

The format of the `<endpoints>` argument is proprietary. All that is required by this submission is that each time **ORB::init** is called with the same value for this argument, the resulting ORB will listen for requests on the same set of endpoints, so that persistent object references for the ORB will continue to function correctly.

4.3.3 Starting Servers with no persistent server support

Any server started with the flag:

-ORBNoProprietaryActivation

shall avoid the use of any proprietary activation framework.

Conformance

5

In order to be conformant with this specification, all interfaces described in Chapter 4 shall be implemented.

6.1 Changes to CORBA

Note – All “note” paragraphs are not normative and should not appear in the final edited specifications.

This submission updates the following specifications:

- [PI] is updated as follows:
 - Section 4.1, “An Abstract Model for Object Adapters”, is added as a new section between [PI] 21.5.1 and 21.5.2
 - Section 4.2.1, “Definition”, Section 4.2.2, “The ObjectReferenceFactory abstract valuetype”, and Section 4.2.3, “The ObjectReferenceTemplate abstract valuetype”, are added in a new section before [PI] 21.5.2, after the submission section 4.1 material.
 - Section 4.2.4, “Extending PortableInterceptor::IORInfo”, updates [PI] section 21.5.3
 - Section 4.2.4.1, “manager_id”, is added to [PI] as new section 21.5.3.4
 - Section 4.2.4.2, “state”, is added to [PI] as new section 21.5.3.5
 - Section 4.2.4.3, “adapter_template”, as added to [PI] as new section 21.5.3.6
 - Section 4.2.4.4, “current_factory”, is added to [PI] as new section 21.5.3.7
 - Section 4.2.5, “Extending PortableInterceptor::IORInterceptor”, updates [PI] section 21.5.2
 - Section 4.2.5.1, “components_established”, is added to [PI] as new section 21.5.2.2
 - Section 4.2.5.2, “adapter_manager_state_changed”, is added to [PI] as new section 21.5.2.3
 - Section 4.2.5.3, “adapter_state_changed”, is added to [PI] as new section 21.5.2.4
 - Section 4.2.5.4, “Method Validity”, is added to [PI] as new section 21.5.3.8

- Section 4.2.6, “Extending PortableInterceptor::ServerRequestInfo”, updates [PI] section 21.3.14 and table 21-2.
- Section 4.2.6.1, “server_id”, is added to [PI] as new section 21.3.14.9.
- Section 4.2.6.2, “orb_id”, is added to [PI] as new section 21.3.14.10
- Section 4.2.6.3, “adapter_name”, is added to [PI] as new section 21.3.14.11.
- Section 4.2.7, “Template Refresh”, is not normative and does not affect [PI] or [CORBA].
- [CORBA] is updated as follows:
 - Section 4.3.1, “Server ID”, is added to [CORBA] as new section 4.5.1.1
 - Section 4.3.2, “Server IIOP Endpoint”, is added to [CORBA] as new section 4.5.1.2.
 - Section 4.3.3, “Starting Servers with no persistent server support”, is added to [CORBA] as new section 4.5.1.3.

6.2 *Changes to Services*

No changes to any COS service are defined in this specification.

Here is the IDL from orbos/00-08-06 augmented with the changes introduced by this submission. The changes are indicated with **blue text**. Those interfaces that have been modified include **#pragma version** statements, with a version x.y, where x.y should be the appropriate (and currently unknown) CORBA version.

#pragma prefix "omg.org"

#include "IOP.idl"

#include "Messaging.idl"

#include "corba.idl"

module Dynamic {

struct Parameter {

any argument;

CORBA::ParameterMode mode;

};

typedef sequence<Parameter> ParameterList;

typedef CORBA::StringSeq ContextList;

typedef sequence<CORBA::TypeCode> ExceptionList;

typedef CORBA::StringSeq RequestContext;

};

module IOP {

typedef sequence<IOP::TaggedComponent> TaggedComponentSeq;

```
local interface Codec {
    exception InvalidTypeForEncoding {};
    exception FormatMismatch {};
    exception TypeMismatch {};

    CORBA::OctetSeq encode (in any data)
        raises (InvalidTypeForEncoding);
    any decode (in CORBA::OctetSeq data)
        raises (FormatMismatch);
    CORBA::OctetSeq encode_value (in any data)
        raises (InvalidTypeForEncoding);
    any decode_value (in CORBA::OctetSeq data,
        in CORBA::TypeCode tc)
        raises (FormatMismatch, TypeMismatch);
};

typedef short EncodingFormat;
const EncodingFormat ENCODING_CDR_ENCAPS = 0;

struct Encoding {
    EncodingFormat format;
    octet major_version;
    octet minor_version;
};

local interface CodecFactory {
    exception UnknownEncoding {};
    Codec create_codec (in Encoding enc) raises (UnknownEncoding);
};

module PortableInterceptor {
    local interface Interceptor {
        readonly attribute string name;
        void destroy();
    };

    exception ForwardRequest {
        Object forward;
    };

    typedef short ReplyStatus;

    // Valid reply_status values:
    const ReplyStatus SUCCESSFUL = 0;
    const ReplyStatus SYSTEM_EXCEPTION = 1;
    const ReplyStatus USER_EXCEPTION = 2;
    const ReplyStatus LOCATION_FORWARD = 3;
    const ReplyStatus TRANSPORT_RETRY = 4;
```

```
typedef unsigned long SlotId;

exception InvalidSlot {};

local interface Current : CORBA::Current {
    any get_slot (in SlotId id) raises (InvalidSlot);
    void set_slot (in SlotId id, in any data) raises (InvalidSlot);
};

local interface RequestInfo {
    readonly attribute unsigned long request_id;
    readonly attribute string operation;
    readonly attribute Dynamic::ParameterList arguments;
    readonly attribute Dynamic::ExceptionList exceptions;
    readonly attribute Dynamic::ContextList contexts;
    readonly attribute Dynamic::RequestContext operation_context;
    readonly attribute any result;
    readonly attribute boolean response_expected;
    readonly attribute Messaging::SyncScope sync_scope;
    readonly attribute ReplyStatus reply_status;
    readonly attribute Object forward_reference;
    any get_slot (in SlotId id) raises (InvalidSlot);
    IOP::ServiceContext get_request_service_context (
        in IOP::Serviceld id);
    IOP::ServiceContext get_reply_service_context (
        in IOP::Serviceld id);
};

local interface ClientRequestInfo : RequestInfo {
    readonly attribute Object target;
    readonly attribute Object effective_target;
    readonly attribute IOP::TaggedProfile effective_profile;
    readonly attribute any received_exception;
    readonly attribute CORBA::RepositoryId received_exception_id;
    IOP::TaggedComponent get_effective_component (
        in IOP::ComponentId id);
    IOP::TaggedComponentSeq get_effective_components (
        in IOP::ComponentId id);
    CORBA::Policy get_request_policy (in CORBA::PolicyType type);
    void add_request_service_context (
        in IOP::ServiceContext service_context,
        in boolean replace);
};

// new from orbos/01-01-04 ORT
typedef string ServerId ;
typedef string ORBId ;
typedef CORBA::StringSeq AdapterName ;
typedef CORBA::OctetSeq ObjectId ;
```

```
local interface ServerRequestInfo : RequestInfo {
    #pragma version ServerRequestInfo x.y
    readonly attribute any sending_exception;

    // new from orbos/01-01-04 ORT
    readonly attribute ServerId server_id ;
    readonly attribute ORBId orb_id ;
    readonly attribute AdapterName adapter_name ;

    readonly attribute CORBA::OctetSeq object_id;
    readonly attribute CORBA::OctetSeq adapter_id;
    readonly attribute CORBA::RepositoryId
        target_most_derived_interface;
    CORBA::Policy get_server_policy (in CORBA::PolicyType type);
    void set_slot (in SlotId id, in any data) raises (InvalidSlot);
    boolean target_is_a (in CORBA::RepositoryId id);
    void add_reply_service_context (
        in IOP::ServiceContext service_context,
        in boolean replace);
};

local interface ClientRequestInterceptor : Interceptor {
    void send_request (in ClientRequestInfo ri)
        raises (ForwardRequest);
    void send_poll (in ClientRequestInfo ri);
    void receive_reply (in ClientRequestInfo ri);
    void receive_exception (in ClientRequestInfo ri)
        raises (ForwardRequest);
    void receive_other (in ClientRequestInfo ri)
        raises (ForwardRequest);
};

local interface ServerRequestInterceptor : Interceptor {
    void receive_request_service_contexts (in ServerRequestInfo ri)
        raises (ForwardRequest);
    void receive_request (in ServerRequestInfo ri)
        raises (ForwardRequest);
    void send_reply (in ServerRequestInfo ri);
    void send_exception (in ServerRequestInfo ri)
        raises (ForwardRequest);
    void send_other (in ServerRequestInfo ri)
        raises (ForwardRequest);
};
```

```
// new from orbos/01-01-04 ORT
abstract valuetype ObjectReferenceFactory {
    Object make_object( in string repository_id, in ObjectId id ) ;
};

abstract valuetype ObjectReferenceTemplate :
    ObjectReferenceFactory {
        readonly attribute ServerId server_id ;
        readonly attribute ORBId orb_id ;
        readonly attribute AdapterName adapter_name ;
    };

typedef sequence<ObjectReferenceTemplate>
    ObjectReferenceTemplateSeq;

typedef long AdapterManagerId;

typedef short AdapterState ;
const AdapterState    HOLDING           = 0 ;
const AdapterState    ACTIVE            = 1 ;
const AdapterState    DISCARDING        = 2 ;
const AdapterState    INACTIVE          = 3 ;
const AdapterState    NON_EXISTENT      = 4 ;

local interface IORInfo {
    #pragma version IORInfo x.y
    CORBA::Policy get_effective_policy (in CORBA::PolicyType type);
    void add_ior_component (
        in IOP::TaggedComponent tagged_component);
    void add_ior_component_to_profile (
        in IOP::TaggedComponent tagged_component,
        in IOP::ProfileId profile_id);

    // new from orbos/01-01-04 ORT
    readonly attribute AdapterManagerId manager_id ;
    readonly attribute AdapterState state ;
    readonly attribute ObjectReferenceTemplate adapter_template ;
    attribute ObjectReferenceFactory current_factory ;
};
```

```
local interface IORInterceptor : Interceptor {
    #pragma version IORInterceptor x.y
    void establish_components (in IORInfo info);

    // new from orbos/01-01-04 ORT
    void components_established( in IORInfo info ) ;
    void adapter_manager_state_changed(
        in AdapterManagerId id, in AdapterState state ) ;
    void adapter_state_changed(
        in ObjectReferenceTemplateSeq templates,
        in AdapterState state ) ;
};

local interface PolicyFactory {
    CORBA::Policy create_policy (
        in CORBA::PolicyType type, in any value)
    raises (CORBA::PolicyError);
};

local interface ORBInitInfo {
    typedef string ObjectId;
    exception DuplicateName {
        string name;
    };
    exception InvalidName {};

    readonly attribute CORBA::StringSeq arguments;
    readonly attribute string orb_id;
    readonly attribute IOP::CodecFactory codec_factory;
    void register_initial_reference (in ObjectId id, in Object obj)
        raises (InvalidName);

    Object resolve_initial_references (in ObjectId id)
        raises (InvalidName);
    void add_client_request_interceptor (
        in ClientRequestInterceptor interceptor)
        raises (DuplicateName);
    void add_server_request_interceptor (
        in ServerRequestInterceptor interceptor)
        raises (DuplicateName);
    void add_ior_interceptor (in IORInterceptor interceptor)
        raises (DuplicateName);
    SlotId allocate_slot_id ();
    void register_policy_factory (
        in CORBA::PolicyType type,
        in PolicyFactory policy_factory);
};
```

```
local interface ORBInitializer {
    void pre_init (in ORBInitInfo info);
    void post_init (in ORBInitInfo info);
};
```

