

Australian/New Zealand Standard™

**Information technology—Open
Distributed Processing**

**Part 2: General Inter-ORB Protocol
(GIOP)/Internet Inter-ORB Protocol
(IIOP)**



AS/NZS ISO/IEC 19500.2:2007

This Joint Australian/New Zealand Standard was prepared by Joint Technical Committee IT-015, Software and Systems Engineering. It was approved on behalf of the Council of Standards Australia on 3 November 2006 and on behalf of the Council of Standards New Zealand on 20 November 2006.
This Standard was published on 24 January 2007.

The following are represented on Committee IT-015:

Australian Computer Society
Australian Electrical and Electronic Manufacturers Association
Australian Society of Technical Communications
Australian Software Metrics Association
Engineers Australia/ACTS Joint Board in Software Engineering
Griffith University
National Association of Testing Authorities Australia
National ICT Australia
New Zealand Organisation for Quality
Software Quality Association, ACT
Software Quality Association, NSW
Systems Engineering Society of Australia
The University of Queensland
University of Auckland, NZ
University of South Australia
University of Technology, Sydney
Vendor Interests, NZ

Keeping Standards up-to-date

Standards are living documents which reflect progress in science, technology and systems. To maintain their currency, all Standards are periodically reviewed, and new editions are published. Between editions, amendments may be issued. Standards may also be withdrawn. It is important that readers assure themselves they are using a current Standard, which should include any amendments which may have been published since the Standard was purchased.

Detailed information about joint Australian/New Zealand Standards can be found by visiting the Standards Web Shop at www.standards.com.au or Standards New Zealand web site at www.standards.co.nz and looking up the relevant Standard in the on-line catalogue.

Alternatively, both organizations publish an annual printed Catalogue with full details of all current Standards. For more frequent listings or notification of revisions, amendments and withdrawals, Standards Australia and Standards New Zealand offer a number of update options. For information about these services, users should contact their respective national Standards organization.

We also welcome suggestions for improvement in our Standards, and especially encourage readers to notify us immediately of any apparent inaccuracies or ambiguities. Please address your comments to the Chief Executive of either Standards Australia or Standards New Zealand at the address shown on the back cover.

This Standard was issued in draft form for comment as DR 06508.

Australian/New Zealand Standard™

Information technology—Open Distributed Processing

Part 2: General Inter-ORB Protocol (GIOP)/Internet Inter-ORB Protocol (IIOP)

First published as AS/NZS ISO/IEC 19500.2:2007.

COPYRIGHT

© Standards Australia/Standards New Zealand

All rights are reserved. No part of this work may be reproduced or copied in any form or by any means, electronic or mechanical, including photocopying, without the written permission of the publisher.

Jointly published by Standards Australia, GPO Box 476, Sydney, NSW 2001 and Standards New Zealand, Private Bag 2439, Wellington 6020

ISBN 0 7337 7996 4

PREFACE

This Standard was prepared by the Joint Standards Australia/Standards New Zealand Committee IT-015, Software and Systems Engineering.

The objective of this Standard is to provide software and systems engineers with a standard that specifies the General Inter-ORB Protocol (GIOP) for object request broker (ORB) interoperability. (GIOP can be mapped onto any connection-oriented transport protocol that meets a minimal set of assumptions defined by this standard).

This Standard is identical with, and has been reproduced from ISO/IEC 19500-2:2003, *Information technology—Open Distributed Processing—Part 2: General Inter-ORB Protocol (GIOP)/Internet Inter-ORB Protocol (IIOP)*.

As this Standard is reproduced from an international standard, the following applies:

- (a) Its number appears on the cover and title page while the international standard number appears only on the cover.
- (b) In the source text ‘this Standard’ should read ‘this Australian/New Zealand Standard’.
- (c) A full point substitutes for a comma when referring to a decimal marker.

References to International Standards should be replaced by references to Australian or Australian/New Zealand Standards, as follows:

<i>Reference to International Standard</i>	<i>Australian/New Zealand Standard</i>
ISO/IEC	AS/NZS ISO/IEC
14750 Information technology—Open Distributed Processing—Interface Definition Language	14750 Information technology—Open distributed processing—Interface definition language
14752 Information technology—Open distributed processing—Protocol support for computational interactions.	14752 Information technology—Open distributed processing—Protocol support for computational interactions

Only normative references in the source document that have been adopted as Australian or Australian/New Zealand Standards have been listed.

CONTENTS

Page

1	Scope	1
2	Normative references	1
2.1	Identical Recommendations International Standards	1
2.2	Other Specifications	2
3	Definitions	2
3.1	Recommendations International Standards	2
3.2	Other Specifications	3
3.2.1	adapter	3
3.2.2	Attribute	3
3.2.3	client	3
3.2.4	data type	3
3.2.5	domain	3
3.2.6	dynamic invocation	3
3.2.7	dynamic skeleton	3
3.2.8	implementation	3
3.2.9	interface repository	3
3.2.10	ORB core	3
3.2.11	repository	4
3.2.12	request	4
3.2.13	results	4
3.2.14	server	4
3.2.15	signature	4
3.2.16	skeleton	4
3.2.17	synchronous request	4
3.2.18	interface type	4
3.2.19	interoperability	4
3.2.20	language binding or mapping	4
3.2.21	method	4
3.2.22	object adapter	5
3.2.23	object implementation	5
3.2.24	object reference	5
3.2.25	objref	5
3.2.26	value	5
3.3	Abbreviations	5
4	Introduction to GIOP/IIOP	6
5	ORB Interoperability Architecture	6
5.1	Overview	6
5.1.1	Domains	6
5.1.2	Bridging Domains	7

5.2 ORBs and ORB Services	7
5.2.1 The Nature of ORB Services	7
5.2.2 ORB Services and Object Requests	8
5.2.3 Selection of ORB Services	8
5.3 Domains	9
5.3.1 Definition of a Domain	9
5.3.2 Mapping Between Domains: Bridging	10
5.4 Interoperability Between ORBs	10
5.4.1 ORB Services and Domains	10
5.4.2 ORBs and Domains	11
5.4.3 Interoperability Approaches	11
5.4.4 Policy-Mediated Bridging	13
5.4.5 Configurations of Bridges in Networks	13
5.5 Object Addressing	14
5.5.1 Domain-relative Object Referencing	14
5.5.2 Handling of Referencing Between Domains	15
5.6 An Information Model for Object References	16
5.6.1 What Information Do Bridges Need?	16
5.6.2 Interoperable Object References: IORs	16
5.6.3 Standard IOR Components	19
5.6.4 Profile and Component Composition in IORs	21
5.6.5 IOR Creation and Scope	22
5.6.6 Stringified Object References	22
5.6.7 Object Service Context	23
5.7 Code Set Conversion	27
5.7.1 Character Processing Terminology	27
5.7.2 Code Set Conversion Framework	29
5.7.3 Mapping to Generic Character Environments	35
5.8 Example of Generic Environment Mapping	37
5.8.1 Generic Mappings	37
5.8.2 Interoperation and Generic Mappings	37
5.9 Relevant OSFM Registry Interfaces	37
5.9.1 Character and Code Set Registry	37
5.9.2 Access Routines	38
6 General Inter-ORB Protocol	42
6.1 Goals of the General Inter-ORB Protocol	42
6.1.1 GIOP Overview	43
6.1.2 Common Data Representation (CDR)	43

6.1.3	GIOP Message Overview	44
6.1.4	GIOP Message Transfer	44
6.2	CDR Transfer Syntax	44
6.2.1	Primitive Types	45
6.2.2	OMG IDL Constructed Types	49
6.2.3	Value Types	52
6.2.4	Pseudo-Object Types	59
6.2.5	Object References	64
6.2.6	Abstract Interfaces	64
6.3	GIOP Message Formats	64
6.3.1	GIOP Message Header	65
6.3.2	Request Message	67
6.3.3	Reply Message	70
6.3.4	CancelRequest Message	73
6.3.5	LocateRequest Message	74
6.3.6	LocateReply Message	75
6.3.7	CloseConnection Message	77
6.3.8	MessageError Message	78
6.3.9	Fragment Message	78
6.4	GIOP Message Transport	79
6.4.1	Connection Management	79
6.4.2	Message Ordering	80
6.5	Object Location	81
6.6	Internet Inter-ORB Protocol (IIOP)	82
6.6.1	TCP/IP Connection Usage	82
6.6.2	IIOP IOR Profiles	83
6.6.3	IIOP IOR Profile Components	85
6.7	Bi-Directional GIOP	86
6.7.1	Bi-Directional IIOP	87
6.8	Bi-directional GIOP policy	88
6.9	OMG IDL	89
6.9.1	GIOP Module	89
6.9.2	IIOP Module	95
6.9.3	BiDirPolicy Module	96

INTRODUCTION

The rapid growth of distributed processing has lead to a need for a coordinating framework for the standardization of Open Distributed Processing (ODP). ITU-T Recommendations X.901-904 | ISO/IEC 10746, the Reference Model of Open Distributed Processing (RM-ODP) provides such a framework. It defines an architecture within which support of distribution, interoperability and portability can be integrated.

Within the framework provided by the RM-ODP, ITU-T Rec. X.931 | ISO/IEC 14752, ODP - Protocol Support for Computational Interactions, defines how interactions between computational objects in a computational specification of a system relate to protocol support for those interactions in an engineering specification of that system.

Annex A to ITU-T Rec. X.931 | ISO/IEC 14752 defines a mapping to the General Inter-ORB Protocol (GIOP) and the Internet Inter-ORB Protocol (IIOP) which are specified by this International Standard.

GIOP is the base for all interoperability and support for all object request broker (ORB) functionality in the Common Object Request Broker Architecture (CORBA) specified by the Object Management Group (OMG). IIOP is the mapping of GIOP for the Internet.

Note: This document is technically aligned with the OMG CORBA GIOP and IIOP specifications.

AUSTRALIAN/NEW ZEALAND STANDARD

Information technology — Open Distributed Processing —**Part 2:
General Inter-ORB Protocol (GIOP)/Internet Inter-ORB Protocol (IIOP)****1 Scope**

This standard specifies the General Inter-ORB Protocol (GIOP) for object request broker (ORB) interoperability. GIOP can be mapped onto any connection-oriented transport protocol that meets a minimal set of assumptions defined by this standard.

This standard also defines the Internet Inter-ORB Protocol (IIOP), a specific mapping of the GIOP which runs directly over connections that use the Internet Protocol and the Transmission Control Protocol (TCP/IP connections).

This standard provides a widely implemented and used particularization of ITU-T Rec. X.931 | ISO/IEC 14752, *Information technology — Open Distributed Processing— Protocol support for computational interactions*. It supports interoperability and location transparency in ODP systems.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, *Information technology — Open Distributed Processing — Reference Model: Foundations*
- ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, *Information technology — Open Distributed Processing — Reference Model: Architecture*
- ITU-T Recommendation X.920 (1997) | ISO/IEC 14750:1999, *Information technology — Open Distributed Processing — Interface Definition Language*
- ISO/IEC 14752:2000, *Information technology — Open Distributed Processing— Protocol support for computational interactions*
- ISO/IEC 8859-1: 1998, *Information technology — 8-bit single-byte coded graphic character sets— Part 1: Latin alphabet No. 1*
- ISO/IEC 10646-1:2000, *Information technology — Universal Multiple-Octect Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*

2.2 Other Specifications

- CORBA 2.3 - The Common Object Request Broker: Architecture and Specification, Revision 2.3, Object Management Group, June 1999 (OMG Doc Number: Formal/98-12-01, <ftp://ftp.omg.org/pub/docs/formal/98-12-01.pdf>)
- CORBAServices: Common Object Services Specification, Object Management Group, December 1998 (OMG Doc Number: Formal/98-12-09, <ftp://ftp.omg.org/pub/docs/formal/98-12-09.pdf>)
- Java™ to IDL Language Mapping, Object Management Group, July 1999 (OMG Doc Number: Formal/99-07-59, <ftp://ftp.omg.org/pub/docs/formal/99-07-59.pdf>)
- STD 007 (also, RFC 793), Transmission Control Protocol, J. Postel, Internet Engineering Task Force, Sept. 1981
- STD 005 (also, RFC 791), Internet Protocol, J. Postel, Internet Engineering Task Force, Sept. 1981
- OSF Character and Code Set Registry, OSF DCE FRC 40.1 (Public Version), S. (Martin) O'Donnell, June 1994.
- RPC Runtime Support For I18N Characters — Functional Specification, OSF DCE SIG RFC 41.2, M. Romagna, R. Mackey, November 1994.

3 Definitions

For the purposes of this International Standard, the following definitions apply:

3.1 Recommendations | International Standards

This International Standard makes use of the following terms defined in ITU-T Rec. X.902 | ISO/IEC 10746-2:

behavior
interface
instance
object
state
type

This International Standard makes use of the following terms defined in ITU-T Rec. X.903 | ISO/IEC 10746-3:

operation
stub

3.2 Other Specifications

3.2.1 adapter

Same as object adapter.

3.2.2 Attribute

An identifiable association between an object and a value. An attribute **A** is made visible to clients as a pair of operations: **get_A** and **set_A**. Readonly attributes only generate a **get** operation.

3.2.3 client

The code or process that invokes an operation on an object.

3.2.4 data type

A categorization of values operation arguments, typically covering both behavior and representation (i.e., the traditional no-OO programming language notion of type.)

3.2.5 domain

A concept important to interoperability, it is a distinct scope, within which common characteristics are exhibited, common rules observed, and over which a distribution transparency is preserved.

3.2.6 dynamic invocation

Constructing and issuing a request whose signature is possibly not known until run-time.

3.2.7 dynamic skeleton

An interface-independent kind of skeleton, used by servers to handle requests whose signatures are possibly not known until run-time.

3.2.8 implementation

A definition that provides the information needed to create an object and allow the object to participate in providing an appropriate set of services. An implementation typically includes a description of the data structure used to represent the core state associated with an object, as well as definitions of the methods that access that data structure. It will also typically include information about the intended interface of the object.

3.2.9 interface repository

A storage place for interface information.

3.2.10 ORB core

The ORB component which moves a request from a client to the appropriate adapter for the target object.

3.2.11 repository

See interface repository and implementation repository.

3.2.12 request

A client issues a request to cause a service to be performed. A request consists of an operation and zero or more actual parameters.

3.2.13 results

The information returned to the client, which may include values as well as status information indicating that exceptional conditions were raised in attempting to perform the requested service.

3.2.14 server

A process implementing one or more operations on one or more objects.

3.2.15 signature

Defines the parameters of a given operation including their number order, data types, and passing mode; the results if any; and the possible outcomes (normal vs. exceptional) that might occur.

3.2.16 skeleton

The object-interface-specific ORB component which assists an object adapter in passing requests to particular methods.

3.2.17 synchronous request

A request where the client pauses to wait for completion of the request. Contrast with deferred synchronous request and one-way request.

3.2.18 interface type

A type satisfied by any object that satisfies a particular interface.

3.2.19 interoperability

The ability for two or more ORBs to cooperate to deliver requests to the proper object. Interoperating ORBs appear to a client to be a single ORB.

3.2.20 language binding or mapping

The means and conventions by which a programmer writing in a specific programming language accesses ORB capabilities.

3.2.21 method

An implementation of an operation. Code that may be executed to perform a requested service. Methods associated with an object may be structured into one or more programs.

3.2.22 object adapter

The ORB component which provides object reference, activation, and state related services to an object implementation. There may be different adapters provided for different kinds of implementations.

3.2.23 object implementation

Same as implementation.

3.2.24 object reference

A value that unambiguously identifies an object. Object references are never reused to identify another object.

3.2.25 objref

An abbreviation for object reference.

3.2.26 value

Any entity that may be a possible actual parameter in a request. Values that serve to identify objects are called object references.

3.3 Abbreviations

For the purposes of this International Standard, the following abbreviations apply:

ADT	Abstract Data Type
CCCS	Client Conversion Code Sets
CCS	Conversion Code Sets
CDR	Common Data Representation
CMIR	Client Makes it Right
CNCS	Client Native Code Set
CORBA	Common Object Request Broker Architecture
DCE	Distributed Computing Environment
OMG	Object Management Group
GIOP	General Inter-ORB Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
ORB	Object Request Broker
SCCS	Server Conversion Code Sets
SMIR	Server Makes It Right

SNCS	Server Native Code Set
TCS	Transmission Code Set
TCS-C	Char Transmission Code Set
TCS-W	Wchar Transmission Code Set
VSCID	Vender Service Context codeset ID

4 Introduction to GIOP/IIOP

This standard specifies the General Inter-ORB Protocol (GIOP) for object request broker (ORB) interoperability. GIOP can be mapped onto any connection-oriented transport protocol that meets a minimal set of assumptions. This standard also defines a specific mapping of the GIOP which runs directly over TCP/IP connections, called the Internet Inter-ORB Protocol (IIOP). The IIOP must be supported by conforming networked ORB products regardless of other aspects of its implementation. Such support does not require using it internally; conforming ORBs may also provide bridges to this protocol.

A definition of interoperability material necessary for the protocol is in clause 5; the definition of GIOP/IIOP is in clause 6. This standard is technically aligned with sections 13 and 15 of CORBA 2.3.

5 ORB Interoperability Architecture

For the purposes of this standard interoperability is defined as the ability for a client on ORB A to invoke an IDL-defined operation on an object on ORB B, where ORB A and ORB B are independently developed. It further identifies general requirements including in particular:

- Ability for two vendors' ORBs to interoperate without prior knowledge of each other's implementations;
- Support of all ORB functionality;
- Preservation of content and semantics of ORB-specific information across ORB boundaries (for example, security).

In effect, the requirement is for invocations between client and server objects to be independent of whether they are on the same or different ORBs, and not to mandate fundamental modifications to existing ORB products.

5.1 Overview

5.1.1 Domains

The CORBA Object Model identifies various distribution transparencies that must be supported within a single ORB environment, such as location transparency. Elements of ORB functionality often correspond directly to such transparencies. Interoperability can be viewed as extending transparencies to span multiple ORBs.

In this architecture a *domain* is a distinct scope, within which certain common characteristics are exhibited and common rules are observed over which a distribution transparency is preserved. Thus, interoperability is fundamentally involved with transparently crossing such domain boundaries.

Domains tend to be either administrative or technological in nature, and need not correspond to the boundaries of an ORB installation. Administrative domains include naming domains, trust groups, resource management domains and other "run-time" characteristics of a system. Technology domains identify common protocols, syntaxes and similar

“build-time” characteristics. In many cases, the need for technology domains derives from basic requirements of administrative domains.

Within a single ORB, most domains are likely to have similar scope to that of the ORB itself: common object references, network addresses, security mechanisms, and more. However, it is possible for there to be multiple domains of the same type supported by a given ORB: internal representation on different machine types, or security domains. Conversely, a domain may span several ORBs: similar network addresses may be used by different ORBs, type identifiers may be shared.

5.1.2 Bridging Domains

The abstract architecture describes ORB interoperability in terms of the translation required when an object request traverses domain boundaries. Conceptually, a mapping or *bridging mechanism* resides at the boundary between the domains, transforming requests expressed in terms of one domain’s model into the model of the destination domain.

The concrete architecture identifies two approaches to inter-ORB bridging:

- At application level, allowing flexibility and portability.
- At ORB level, built into the ORB itself.

5.2 ORBs and ORB Services

The ORB Core is that part of the ORB which provides the basic representation of objects and the communication of requests. The ORB Core therefore supports the minimum functionality to enable a client to invoke an operation on a server object, with (some of) the distribution transparencies required by *CORBA*.

An object request may have implicit attributes which affect the way in which it is communicated - though not the way in which a client makes the request. These attributes include security, transactional capabilities, recovery, and replication. These features are provided by “ORB Services,” which will in some ORBs be layered as internal services over the core, or in other cases be incorporated directly into an ORB’s core. It is an aim of this specification to allow for new ORB Services to be defined in the future, without the need to modify or enhance this architecture.

Within a single ORB, ORB services required to communicate a request will be implemented and (implicitly) invoked in a private manner. For interoperability between ORBs, the ORB services used in the ORBs, and the correspondence between them, must be identified.

5.2.1 The Nature of ORB Services

ORB Services are invoked implicitly in the course of application-level interactions. ORB Services range from fundamental mechanisms such as reference resolution and message encoding to advanced features such as support for security, transactions, or replication.

An ORB Service is often related to a particular transparency. For example, message encoding – the marshaling and unmarshaling of the components of a request into and out of message buffers – provides transparency of the representation of the request. Similarly, reference resolution supports location transparency. Some transparencies, such as security, are supported by a combination of ORB Services and Object Services while others, such as replication, may involve interactions between ORB Services themselves.

ORB Services differ from Object Services in that they are positioned below the application and are invoked transparently to the application code. However, many ORB Services include components which correspond to conventional Object Services in that they are invoked explicitly by the application.

Security is an example of service with both ORB Service and normal Object Service components, the ORB components being those associated with transparently authenticating messages and controlling access to objects while the necessary administration and management functions resemble conventional Object Services.

5.2.2 ORB Services and Object Requests

Interoperability between ORBs extends the scope of distribution transparencies and other request attributes to span multiple ORBs. This requires the establishment of relationships between supporting ORB Services in the different ORBs.

In order to discuss how the relationships between ORB Services are established, it is necessary to describe an abstract view of how an operation invocation is communicated from client to server object.

1. The client generates an operation request, using a reference to the server object, explicit parameters, and an implicit invocation context. This is processed by certain ORB Services on the client path.
2. On the server side, corresponding ORB Services process the incoming request, transforming it into a form directly suitable for invoking the operation on the server object.
3. The server object performs the requested operation.
4. Any result of the operation is returned to the client in a similar manner.

The correspondence between client-side and server-side ORB Services need not be one-to-one and in some circumstances may be far more complex. For example, if a client application requests an operation on a replicated server, there may be multiple server-side ORB service instances, possibly interacting with each other.

In other cases, such as security, client-side or server-side ORB Services may interact with Object Services such as authentication servers.

5.2.3 Selection of ORB Services

The ORB Services used are determined by:

- Static properties of both client and server objects; for example, whether a server is replicated.
- Dynamic attributes determined by a particular invocation context; for example, whether a request is transactional.
- Administrative policies (e.g., security).

Within a single ORB, private mechanisms (and optimizations) can be used to establish which ORB Services are required and how they are provided. Service selection might in general require negotiation to select protocols or protocol options. The same is true between different ORBs: it is necessary to agree which ORB Services are used, and how each transforms the request. Ultimately, these choices become manifest as one or more protocols between the ORBs or as transformations of requests.

In principle, agreement on the use of each ORB Service can be independent of the others and, in appropriately constructed ORBs, services could be layered in any order or in any grouping. This potentially allows applications to specify selective transparencies according to their requirements, although at this time CORBA provides no way to penetrate its transparencies.

A client ORB must be able to determine which ORB Services must be used in order to invoke operations on a server object. Correspondingly, where a client requires dynamic attributes to be associated with specific invocations, or administrative policies dictate, it must be possible to cause the appropriate ORB Services to be used on client and

server sides of the invocation path. Where this is not possible - because, for example, one ORB does not support the full set of services required - either the interaction cannot proceed or it can only do so with reduced facilities or transparencies.

5.3 Domains

From a computational viewpoint, the OMG Object Model identifies various distribution transparencies which ensure that client and server objects are presented with a uniform view of a heterogeneous distributed system. From an engineering viewpoint, however, the system is not wholly uniform. There may be distinctions of location and possibly many others such as processor architecture, networking mechanisms and data representations. Even when a single ORB implementation is used throughout the system, local instances may represent distinct, possibly optimized scopes for some aspects of ORB functionality.

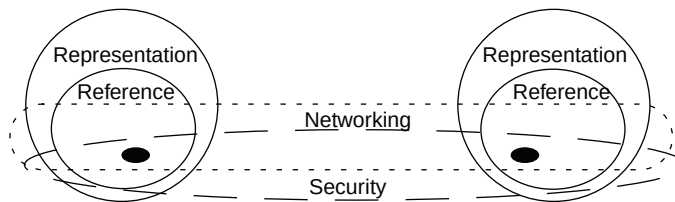


Figure 1 - Different Kinds of Domains can Coexist

Interoperability, by definition, introduces further distinctions, notably between the scopes associated with each ORB. To describe both the requirements for interoperability and some of the solutions, this architecture introduces the concept of *domains* to describe the scopes and their implications.

Informally, a domain is a set of objects sharing a common characteristic or abiding by common rules. It is a powerful modelling concept which can simplify the analysis and description of complex systems. There may be many types of domains (e.g., management domains, naming domains, language domains, and technology domains).

5.3.1 Definition of a Domain

Domains allow partitioning of systems into collections of components which have some characteristic in common. In this architecture a domain is a scope in which a collection of objects, said to be members of the domain, is associated with some common characteristic; any object for which the association does not exist, or is undefined, is not a member of the domain. A domain can be modelled as an object and may be itself a member of other domains.

It is the scopes themselves and the object associations or bindings defined within them which characterize a domain. This information is disjoint between domains. However, an object may be a member of several domains, of similar kinds as well as of different kinds, and so the sets of members of domains may overlap.

The concept of a domain boundary is defined as the limit of the scope in which a particular characteristic is valid or meaningful. When a characteristic in one domain is translated to an equivalent in another domain, it is convenient to consider it as traversing the boundary between the two domains.

Domains are generally either administrative or technological in nature. Examples of domains related to ORB interoperability issues are:

- Referencing domain – the scope of an object reference
- Representation domain – the scope of a message transfer syntax and protocol

- Network addressing domain – the scope of a network address
- Network connectivity domain – the potential scope of a network message
- Security domain – the extent of a particular security policy
- Type domain – the scope of a particular type identifier
- Transaction domain – the scope of a given transaction service

Domains can be related in two ways: containment, where a domain is contained within another domain, and federation, where two domains are joined in a manner agreed to and set up by their administrators.

5.3.2 Mapping Between Domains: Bridging

Interoperability between domains is only possible if there is a well-defined mapping between the behaviors of the domains being joined. Conceptually, a mapping mechanism or bridge resides at the boundary between the domains, transforming requests expressed in terms of one domain's model into the model of the destination domain. Note that the use of the term "bridge" in this context is conceptual and refers only to the functionality which performs the required mappings between distinct domains. There are several implementation options for such bridges and these are discussed elsewhere.

For full interoperability, it is essential that all the concepts used in one domain are transformable into concepts in other domains with which interoperability is required, or that if the bridge mechanism filters such a concept out, nothing is lost as far as the supported objects are concerned. In other words, one domain may support a superior service to others, but such a superior functionality will not be available to an application system spanning those domains.

A special case of this requirement is that the object models of the two domains need to be compatible. This specification assumes that both domains are strictly compliant with the CORBA Object Model and the *CORBA* specifications. This includes the use of OMG IDL (technically aligned with ISO/IEC 14750) when defining interfaces, the use of the CORBA Core Interface Repository, and other modifications that were made to *CORBA*. Variances from this model could easily compromise some aspects of interoperability.

5.4 Interoperability Between ORBs

An ORB "provides the mechanisms by which objects transparently make and receive requests and responses. In so doing, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments..." ORB interoperability extends this definition to cases in which client and server objects on different ORBs "transparently make and receive requests..."

Note that a direct consequence of this transparency requirement is that bridging must be bidirectional: that is, it must work as effectively for object references passed as parameters as for the target of an object invocation. Were bridging unidirectional (e.g., if one ORB could only be a client to another) then transparency would not have been provided, because object references passed as parameters would not work correctly: ones passed as "callback objects," for example, could not be used.

Without loss of generality, most of this specification focuses on bridging in only one direction. This is purely to simplify discussions, and does not imply that unidirectional connectivity satisfies basic interoperability requirements.

5.4.1 ORB Services and Domains

In this architecture, different aspects of ORB functionality - ORB Services - can be considered independently and associated with different domain types. The architecture does not, however, prescribe any particular decomposition

of ORB functionality and interoperability into ORB Services and corresponding domain types. There is a range of possibilities for such a decomposition:

1. The simplest model, for interoperability, is to treat all objects supported by one ORB (or, alternatively, all ORBs of a given type) as comprising one domain. Interoperability between any pair of different domains (or domain types) is then achieved by a specific all-encompassing bridge between the domains. (This is all *CORBA* implies.)
2. More detailed decompositions would identify particular domain types - such as referencing, representation, security, and networking. A core set of domain types would be pre-determined and allowance made for additional domain types to be defined as future requirements dictate (e.g., for new ORB Services).

5.4.2 ORBs and Domains

In many respects, issues of interoperability between ORBs are similar to those which can arise with a single type of ORB (e.g., a product). For example:

- Two installations of the ORB may be installed in different security domains, with different Principal identifiers. Requests crossing those security domain boundaries will need to establish locally meaningful Principals for the caller identity, and for any Principals passed as parameters.
- Different installations might assign different type identifiers for equivalent types, and so requests crossing type domain boundaries would need to establish locally meaningful type identifiers (and perhaps more).

Conversely, not all of these problems need to appear when connecting two ORBs of a different type (e.g., two different products). Examples include:

- They could be administered to share user visible naming domains, so that naming domains do not need bridging.
- They might reuse the same networking infrastructure, so that messages could be sent without needing to bridge different connectivity domains.

Additional problems can arise with ORBs of different types. In particular, they may support different concepts or models, between which there are no direct or natural mappings. *CORBA* only specifies the application level view of object interactions, and requires that distribution transparencies conceal a whole range of lower level issues. It follows that within any particular ORB, the mechanisms for supporting transparencies are not visible at the application-level and are entirely a matter of implementation choice. So there is no guarantee that any two ORBs support similar internal models or that there is necessarily a straightforward mapping between those models.

These observations suggest that the concept of an ORB (instance) is too coarse or superficial to allow detailed analysis of interoperability issues between ORBs. Indeed, it becomes clear that an ORB instance is an elusive notion: it can perhaps best be characterized as the intersection or coincidence of ORB Service domains.

5.4.3 Interoperability Approaches

When an interaction takes place across a domain boundary, a mapping mechanism, or bridge, is required to transform relevant elements of the interaction as they traverse the boundary. There are essentially two approaches to achieving this: mediated bridging and immediate bridging. These approaches are described in the following subsections.

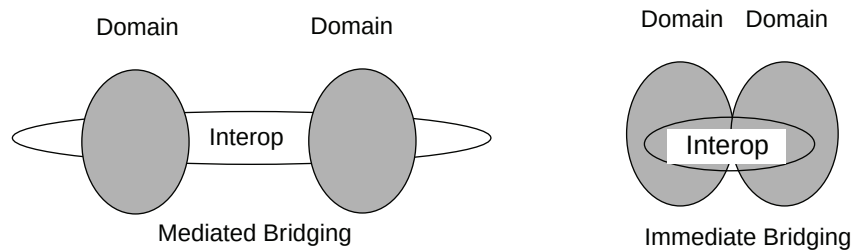


Figure 2 - Two bridging techniques, different uses of an intermediate form agreed on between the two domains.

5.4.3.1 Mediated Bridging

With mediated bridging, elements of the interaction relevant to the domain are transformed, at the boundary of each domain, between the internal form of that domain and an agreed, common form.

Observations on mediated bridging are as follows:

- The scope of agreement of a common form can range from a private agreement between two particular ORB/domain implementations to a universal standard.
- There can be more than one common form, each oriented or optimized for a different purpose.
- If there is more than one possible common form, then which is used can be static (e.g., administrative policy agreed between ORB vendors, or between system administrators) or dynamic (e.g., established separately for each object, or on each invocation).
- Engineering of this approach can range from in-line specifically compiled (compare to stubs) or generic library code (such as encryption routines), to intermediate bridges to the common form.

5.4.3.2 Immediate Bridging

With immediate bridging, elements of the interaction relevant to the domain are transformed, at the boundary of each domain, directly between the internal form of one domain and the internal form of the other.

Observations on immediate bridging are as follows:

- This approach has the potential to be optimal (in that the interaction is not mediated via a third party, and can be specifically engineered for each pair of domains) but sacrifices flexibility and generality of interoperability to achieve this.
- This approach is often applicable when crossing domain boundaries which are purely administrative (i.e., there is no change of technology). For example, when crossing security administration domains between similar ORBs, it is not necessary to use a common intermediate standard.

As a general observation, the two approaches can become almost indistinguishable when private mechanisms are used between ORB/domain implementations.

5.4.3.3 Location of Inter-Domain Functionality

Logically, an inter-domain bridge has components in both domains, whether the mediated or immediate bridging approach is used. However, domains can span ORB boundaries and ORBs can span machine and system boundaries;

conversely, a machine may support, or a process may have access to more than one ORB (or domain of a given type). From an engineering viewpoint, this means that the components of an inter-domain bridge may be dispersed or co-located, with respect to ORBs or systems. It also means that the distinction between an ORB and a bridge can be a matter of perspective: there is a duality between viewing inter-system messaging as belonging to ORBs, or to bridges.

For example, if a single ORB encompasses two security domains, the inter-domain bridge could be implemented wholly within the ORB and thus be invisible as far as ORB interoperability is concerned. A similar situation arises when a bridge between two ORBs or domains is implemented wholly within a process or system which has access to both. In such cases, the engineering issues of inter-domain bridging are confined, possibly to a single system or process. If it were practical to implement all bridging in this way, then interactions between systems or processes would be solely within a single domain or ORB.

5.4.3.4 Bridging Level

As noted at the start of this section, bridges may be implemented both internally to an ORB and as layers above it. These are called respectively “in-line” and “request-level” bridges.

Request-level bridges use the CORBA APIs, including the Dynamic Skeleton Interface, to receive and issue requests. However, there is an emerging class of “implicit context” which may be associated with some invocations, holding ORB Service information such as transaction and security context information, which is not at this time exposed through general purpose public APIs. (Those APIs expose only OMG IDL-defined operation parameters, not implicit ones.) Rather, the precedent set with the Transaction Service is that special purpose APIs are defined to allow bridging of each kind of context. This means that request-level bridges must be built to specifically understand the implications of bridging such ORB Service domains, and to make the appropriate API calls.

5.4.4 Policy-Mediated Bridging

An assumption made through most of this specification is that the existence of domain boundaries should be transparent to requests: that the goal of interoperability is to hide such boundaries. However, if this were always the goal, then there would be no real need for those boundaries in the first place.

Realistically, administrative domain boundaries exist because they reflect ongoing differences in organizational policies or goals. Bridging the domains will in such cases require *policy mediation*. That is, inter-domain traffic will need to be constrained, controlled, or monitored; fully transparent bridging may be highly undesirable. Resource management policies may even need to be applied, restricting some kinds of traffic during certain periods.

Security policies are a particularly rich source of examples: a domain may need to audit external access, or to provide domain-based access control. Only a very few objects, types of objects, or classifications of data might be externally accessible through a “firewall.”

Such policy-mediated bridging requires a bridge that knows something about the traffic being bridged. It could in general be an application-specific policy, and many policy-mediated bridges could be parts of applications. Those might be organization-specific, off-the-shelf, or anywhere in between.

Request-level bridges, which use only public ORB APIs, easily support the addition of policy mediation components, without loss of access to any other system infrastructure that may be needed to identify or enforce the appropriate policies.

5.4.5 Configurations of Bridges in Networks

In the case of network-aware ORBs, we anticipate that some ORB protocols will be more frequently bridged to than others, and so will begin to serve the role of “backbone ORBs.” (This is a role that the IIOP is specifically expected to serve.) This use of “backbone topology” is true both on a large scale and a small scale. While a large scale public data

network provider could define its own backbone ORB, on a smaller scale, any given institution will probably designate one commercially available ORB as its backbone.

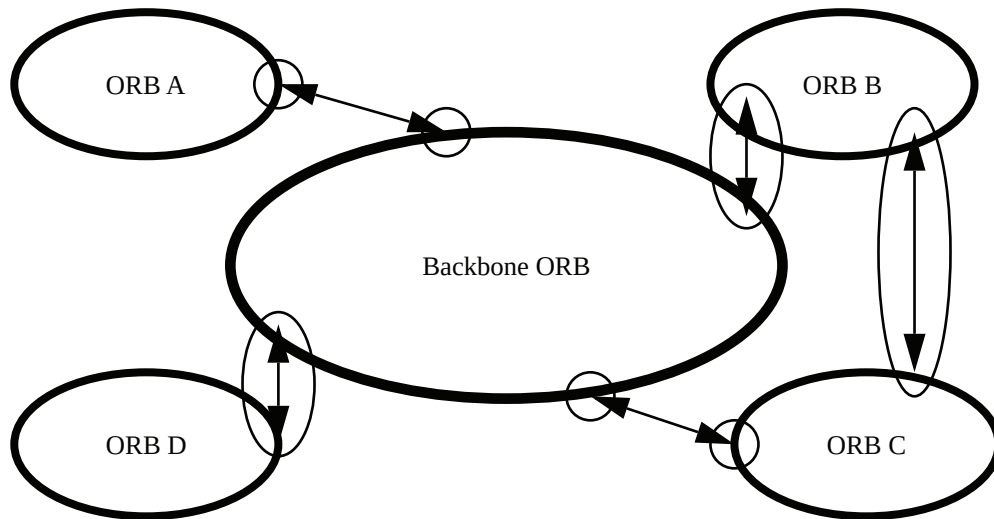


Figure 3 - An ORB chosen as a backbone will connect other ORBs through bridges, both full-bridges and half-bridges.

Adopting a backbone style architecture is a standard administrative technique for managing networks. It has the consequence of minimizing the number of bridges needed, while at the same time making the ORB topology match typical network organizations. (That is, it allows the number of bridges to be proportional to the number of protocols, rather than combinatorial.)

In large configurations, it will be common to notice that adding ORB bridges doesn't even add any new "hops" to network routes, because the bridges naturally fit in locations where connectivity was already indirect, and augment or supplant the existing network firewalls.

5.5 Object Addressing

The CORBA Object Model defines an object reference as an object name that reliably denotes a particular object. An object reference identifies the same object each time the reference is used in a request, and an object may be denoted by multiple, distinct references.

The fundamental ORB interoperability requirement is to allow clients to use such object names to invoke operations on objects in other ORBs. Clients do not need to distinguish between references to objects in a local ORB or in a remote one. Providing this transparency can be quite involved, and naming models are fundamental to it.

This section of this specification discusses models for naming entities in multiple domains, and transformations of such names as they cross the domain boundaries. That is, it presents transformations of object reference information as it passes through networks of inter-ORB bridges. It uses the word "ORB" as synonymous with referencing domain; this is purely to simplify the discussion. In other contexts, "ORB" can usefully denote other kinds of domain.

5.5.1 Domain-relative Object Referencing

Since CORBA does not require ORBs to understand object references from other ORBs, when discussing object references from multiple ORBs one must always associate the object reference's domain (ORB) with the object

reference. We use the notation $D0.R0$ to denote an object reference $R0$ from domain $D0$; this is itself an object reference. This is called “domain-relative” referencing (or addressing) and need not reflect the implementation of object references within any ORB.

At an implementation level, associating an object reference with an ORB is only important at an inter-ORB boundary; that is, inside a bridge. This is simple, since the bridge knows from which ORB each request (or response) came, including any object references embedded in it.

5.5.2 Handling of Referencing Between Domains

When a bridge hands an object reference to an ORB, it must do so in a form understood by that ORB: the object reference must be in the recipient ORB’s native format. Also, in cases where that object originated from some other ORB, the bridge must associate each newly created “proxy” object reference with (what it sees as) the original object reference.

Several basic schemes to solve these two problems exist. These all have advantages in some circumstances; all can be used, and in arbitrary combination with each other, since CORBA object references are opaque to applications. The ramifications of each scheme merits attention, with respect to scaling and administration. The schemes include:

1. *Object Reference Translation Reference Embedding*: The bridge can store the original object reference itself, and pass an entirely different proxy reference into the new domain. The bridge must then manage state on behalf of each bridged object reference, map these references from one ORB’s format to the other’s, and vice versa.
2. *Reference Encapsulation*: The bridge can avoid holding any state at all by conceptually concatenating a domain identifier to the object name. Thus if a reference $D0.R$, originating in domain $D0$, traversed domains $D1...D4$ it could be identified in $D4$ as proxy reference $d3.d2.d1.d0.R$, where dn is the address of Dn relative to $Dn+1$.

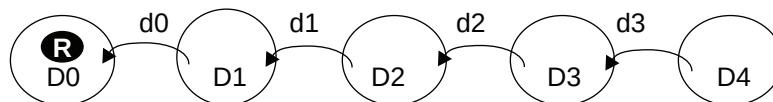


Figure 4 - Reference encapsulation adds domain information during bridging

3. *Domain Reference Translation*: Like object reference translation, this scheme holds some state in the bridge. However, it supports sharing that state between multiple object references by adding a domain-based route identifier to the proxy (which still holds the original reference, as in the reference encapsulation scheme). It achieves this by providing encoded domain route information each time a domain boundary is traversed; thus if a reference $D0.R$, originating in domain $D0$, traversed domains $D1...D4$ it would be identified in $D4$ as $(d3, x3).R$, and in $D2$ as $(d1, x1).R$, and so on, where dn is the address of Dn relative to $Dn+1$, and xn identifies the pair $(dn-1, xn-1)$.

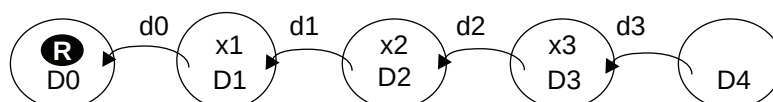


Figure 5 - Domain Reference Translation substitutes domain references during bridging

4. *Reference Canonicalization*: This scheme is like domain reference translation, except that the proxy uses a “well-known” (e.g., global) domain identifier rather than an encoded path. Thus a reference R , originating in domain $D0$ would be identified in other domains as $D0.R$.

Observations about these approaches to inter-domain reference handling are as follows:

- Naive application of reference encapsulation could lead to arbitrarily large references. A “topology service” could optimize cycles within any given encapsulated reference and eliminate the appearance of references to local objects as alien references.
- A topology service could also optimize the chains of routes used in the domain reference translation scheme. Since the links in such chains are re-used by any path traversing the same sequence of domains, such optimization has particularly high leverage.
- With the general purpose APIs defined in *CORBA*, object reference translation can be supported even by ORBs not specifically intended to support efficient bridging, but this approach involves the most state in intermediate bridges. As with reference encapsulation, a topology service could optimize individual object references. (APIs are defined by the Dynamic Skeleton Interface and Dynamic Invocation Interface)
- The chain of addressing links established with both object and domain reference translation schemes must be represented as state within the network of bridges. There are issues associated with managing this state.
- Reference canonicalization can also be performed with managed hierarchical name spaces such as those now in use on the Internet and X.500 naming.

5.6 An Information Model for Object References

This section provides a simple, powerful information model for the information found in an object reference. That model is intended to be used directly by developers of bridging technology, and is used in that role by the IIOP, described in the 6.2.5, ‘Object References’.

5.6.1 What Information Do Bridges Need?

The following potential information about object references has been identified as critical for use in bridging technologies:

- *Is it null?* Nulls only need to be transmitted and never support operation invocation.
- *What type is it?* Many ORBs require knowledge of an object’s type in order to efficiently preserve the integrity of their type systems.
- *What protocols are supported?* Some ORBs support objrefs that in effect live in multiple referencing domains, to allow clients the choice of the most efficient communications facilities available.
- *What ORB Services are available?* As noted in 5.2.3, ‘Selection of ORB Services’, several different ORB Services might be involved in an invocation. Providing information about those services in a standardized way could in many cases reduce or eliminate negotiation overhead in selecting them.

5.6.2 Interoperable Object References: IORs

To provide the information above, an “Interoperable Object Reference,” (IOR) data structure has been provided. This data structure need not be used internally to any given ORB, and is not intended to be visible to application-level ORB programmers. It should be used only when crossing object reference domain boundaries, within bridges.

This data structure is designed to be efficient in typical single-protocol configurations, while not penalizing multiprotocol ones.

```

module IOP {                                     // IDL

    // Standard Protocol Profile tag values

    typedef unsigned long      ProfileId;
    const ProfileId            TAG_INTERNET_IOP = 0;
    const ProfileId            TAG_MULTIPLE_COMPONENTS = 1;

    struct TaggedProfile {
        ProfileId                tag;
        sequence <octet>          profile_data;
    };

    // an Interoperable Object Reference is a sequence of
    // object-specific protocol profiles, plus a type ID.

    struct IOR {
        string                    type_id;
        sequence <TaggedProfile> profiles;
    };

    // Standard way of representing multicomponent profiles.
    // This would be encapsulated in a TaggedProfile.

    typedef unsigned long ComponentId;
    struct TaggedComponent {
        ComponentId              tag;
        sequence <octet>          component_data;
    };
    typedef sequence <TaggedComponent> MultipleComponentProfile;
};

```

Object references have at least one *tagged profile*. Each profile supports one or more protocols and encapsulates all the basic information the protocols it supports need to identify an object. Any single profile holds enough information

to drive a complete invocation using any of the protocols it supports; the content and structure of those profile entries are wholly specified by these protocols. A bridge between two domains may need to know the detailed content of the profile for those domains' profiles, depending on the technique it uses to bridge the domains¹.

Each profile has a unique numeric tag, assigned by the OMG. The ones defined here are for the IIOP (see 6.6.3, 'IIOP IOR Profile Components') and for use in "multiple component profiles." Profile tags in the range **0x80000000** through **0xffffffff** are reserved for future use, and are not currently available for assignment.

Null object references are indicated by an empty set of profiles, and by a "Null" type ID (a string which contains only a single terminating character). A Null TypeID is the only mechanism that can be used to represent the type **CORBA::Object**. Type IDs may only be "Null" in any message, requiring the client to use existing knowledge or to consult the object, to determine interface types supported. The type ID is a Repository ID identifying the interface type, and is provided to allow ORBs to preserve strong typing. This identifier is agreed on within the bridge and, for reasons outside the scope of this interoperability specification, needs to have a much broader scope to address various problems in system evolution and maintenance. Type IDs support detection of type equivalence, and in conjunction with an Interface Repository, allow processes to reason about the relationship of the type of the object referred to and any other type.

The type ID, if provided by the server, indicates the most derived type that the server wishes to publish, at the time the reference is generated. The object's actual most derived type may later change to a more derived type. Therefore, the type ID in the IOR can only be interpreted by the client as a hint that the object supports at least the indicated interface. The client can succeed in narrowing the reference to the indicated interface, or to one of its base interfaces,

based solely on the type ID in the IOR, but must not fail to narrow the reference without consulting the object via the "_is_a" or "_get_interface" pseudo-operations.

5.6.2.1 The TAG_INTERNET_IOP Profile

The **TAG_INTERNET_IOP** tag identifies profiles that support the Internet Inter-ORB Protocol. The **ProfileBody** of this profile, described in detail in 6.6.2, 'IIOP IOR Profiles', contains a CDR encapsulation of a structure containing addressing and object identification information used by IIOP. Version 1.1 of the **TAG_INTERNET_IOP** profile also includes a **sequence<TaggedComponent>** that can contain additional information supporting optional IIOP features, ORB services such as security, and future protocol extensions.

Protocols other than IIOP (such as ESIOPs and other GIOPs) can share profile information (such as object identity or security information) with IIOP by encoding their additional profile information as components in the **TAG_INTERNET_IOP** profile. All **TAG_INTERNET_IOP** profiles support IIOP, regardless of whether they also support additional protocols. Interoperable ORBs are not required to create or understand any other profile, nor are they required to create or understand any of the components defined for other protocols that might share the **TAG_INTERNET_IOP** profile with IIOP.

5.6.2.2 The TAG_MULTIPLE_COMPONENTS Profile

The **TAG_MULTIPLE_COMPONENTS** tag indicates that the value encapsulated is of type **MultipleComponentProfile**. In this case, the profile consists of a list of protocol components, indicating ORB services accessible using that protocol. ORB services are assigned component identifiers in a namespace that is distinct from the profile identifiers. Note that protocols may use the **MultipleComponentProfile** data structure to hold

1. Based on topology and policy information available to it, a bridge may find it prudent to add or remove some profiles as it forwards an object reference. For example, a bridge acting as a firewall might remove all profiles except ones that make such profiles, letting clients that understand the profiles make routing choices.

profile components even without using **TAG_MULTIPLE_COMPONENTS** to indicate that particular protocol profile, and need not use a **MultipleComponentProfile** to hold sets of profile components.

5.6.2.3 IOR Components

TaggedComponents contained in **TAG_INTERNET_IOP** and **TAG_MULTIPLE_COMPONENTS** profiles are identified by unique numeric tags using a namespace distinct from that is used for profile tags. Component tags are assigned by the OMG.

Specifications of components must include the following information:

- *Component ID*: The compound tag that is obtained from OMG.
- *Structure and encoding*: The syntax of the component data and the encoding rules. If the component value is encoded as a CDR encapsulation, the IDL type that is encapsulated and the GIOP version which is used for encoding the value, if different than GIOP 1.0, must be specified as part of the component definition.
- *Semantics*: How the component data is intended to be used.
- *Protocols*: The protocol for which the component is defined, and whether it is intended that the component be usable by other protocols.
- *At most once*: whether more than one instance of this component can be included in a profile.

Specification of protocols must describe how the components affect the protocol. The following should be specified in any protocol definition for each **TaggedComponent** that the protocol uses:

- *Mandatory presence*: Whether inclusion of the component in profiles supporting the protocol is required (MANDATORY PRESENCE) or not required (OPTIONAL PRESENCE).
- *Droppable*: For optional presence component, whether component, if present, must be retained or may be dropped.

5.6.3 Standard IOR Components

The following are standard IOR components that can be included in **TAG_INTERNET_IOP** and **TAG_MULTIPLE_COMPONENTS** profiles, and may apply to IIOP, other GIOPs, ESIOPs, or other protocols. An ORB must not drop these components from an existing IOR.

module IOP {

const ComponentId	TAG_ORB_TYPE = 0;
const ComponentId	TAG_CODE_SETS = 1;
const ComponentId	TAG_POLICIES = 2;
const ComponentId	TAG_ALTERNATE_IOP_ADDRESS = 3;
const ComponentId	TAG_ASSOCIATION_OPTIONS = 13;
const ComponentId	TAG_SEC_NAME = 14;
const ComponentId	TAG_SPKM_1_SEC_MECH = 15;
const ComponentId	TAG_SPKM_2_SEC_MECH = 16;
const ComponentId	TAG_KerberosV5_SEC_MECH = 17;

```

const ComponentId    TAG_CSI_ECMA_Secret_SEC_MECH = 18;
const ComponentId    TAG_CSI_ECMA_Hybrid_SEC_MECH = 19;
const ComponentId    TAG_SSL_SEC_TRANS = 20;
const ComponentId    TAG_CSI_ECMA_Public_SEC_MECH = 21;
const ComponentId    TAG_GENERIC_SEC_MECH = 22;
const ComponentId    TAG_JAVA_CODEBASE = 25;
};

```

The following additional components that can be used by other protocols are specified in the DCE ESIOP chapter of *CORBA 2.3* and others in the section on Security Service for DCE ESIOP in the Security Service chapter of *CORBAServices*:

```

const ComponentId    TAG_COMPLETE_OBJECT_KEY = 5;
const ComponentId    TAG_ENDPOINT_ID_POSITION = 6;
const ComponentId    TAG_LOCATION_POLICY = 12;
const ComponentId    TAG_DCE_STRING_BINDING = 100;
const ComponentId    TAG_DCE_BINDING_NAME = 101;
const ComponentId    TAG_DCE_NO_PIPES = 102;
const ComponentId    TAG_DCE_SEC_MECH = 103; // Security Service

```

5.6.3.1 TAG_ORB_TYPE Component

It is often useful in the real world to be able to identify the particular kind of ORB an object reference is coming from, to work around problems with that particular ORB, or exploit shared efficiencies.

The **TAG_ORB_TYPE** component has an associated value of type **unsigned long**, encoded as a CDR encapsulation, designating an ORB type ID allocated by the OMG for the ORB type of the originating ORB. Anyone may register any ORB types by submitting a short (one-paragraph) description of the ORB type to the OMG, and will receive a new ORB type ID in return. A list of ORB type descriptions and values will be made available on the OMG web server.

The **TAG_ORB_TYPE** component can appear at most once in any IOR profile. For profiles supporting IIOP 1.1 or greater, it is optionally present and may not be dropped.

5.6.3.2 TAG_ALTERNATE_Iiop_ADDRESS Component

In cases where the same object key is used for more than one internet location, the following standard IOR Component is defined for support in IIOP version 1.2.

The **TAG_ALTERNATE_Iiop_ADDRESS** component has an associated value of type

```

struct {
    string HostID,
    short Port

```

};

encoded as a CDR encapsulation.

Zero or more instances of the **TAG_ALTERNATE_IOP_ADDRESS** component type may be included in a version 1.2 **TAG_INTERNET_IOP** Profile. Each of these alternative addresses may be used by the client orb, in addition to the host and port address expressed in the body of the Profile. In cases where one or more **TAG_ALTERNATE_IOP_ADDRESS** components are present in a **TAG_INTERNET_IOP** Profile, no order of use is prescribed by Version 1.2 of IIOP.

5.6.3.3 Other Components

The following standard components are specified in various OMG specifications. Refer to 2.2, 'Other Specifications' for additional information.

- **TAG_CODE_SETS** (see 5.7.2.4, 'CodeSet Component of IOR Multi-Component Profile').
- **TAG_POLICIES** (reserved for CORBA Messaging specification, which will be incorporated into CORBA 3.0).
- **TAG_SEC_NAME** (see 15.10.2 Mechanism Tags, Security chapter - CORBAServices).
- **TAG_ASSOCIATION_OPTIONS** (see 15.10.3 Tag Association Options, Security chapter - CORBAServices).
- **TAG_SSL_SEC_TRANS** (see 15.10.2 Mechanism Tags, Security chapter - CORBAServices).
- **TAG_GENERIC_SEC_MECH** and all other tags with names in the form **TAG_*_SEC_MECH** (see 15.10.2 Mechanism Tags, Security chapter - CORBAServices).
- **TAG_JAVA_CODEBASE** (see 1.4.9.3, "Codebase Transmission" - Java to IDL Language Mapping).
- **TAG_COMPLETE_OBJECT_KEY** (see 16.5.4, "Complete Object Key Component" - CORBA 2.3).
- **TAG_ENDPOINT_ID_POSITION** (see 16.5.5, "Endpoint ID Position Component" - CORBA 2.3).
- **TAG_LOCATION_POLICY** (see 16.5.6, "Location Policy Component" - CORBA 2.3).
- **TAG_DCE_STRING_BINDING** (see 16.5.1, "DCE-CIOP String Binding Component" - CORBA 2.3).
- **TAG_DCE_BINDING_NAME** (see 16.5.2, "DCE-CIOP Binding Name Component" - CORBA 2.3).
- **TAG_DCE_NO_PIPES** (see 16.5.3, "DCE-CIOP No pipes Component" - CORBA 2.3).

5.6.4 Profile and Component Composition in IORs

The following rules augment the preceding discussion:

1. Profiles must be independent, complete, and self-contained. Their use shall not depend on information contained in another profile.
2. Any invocation uses information from exactly one profile.
3. Information used to drive multiple inter-ORB protocols may coexist within a single profile, possibly with some information (e.g., components) shared between the protocols, as specified by the specific protocols.
4. Unless otherwise specified in the definition of a particular profile, multiple profiles with the same profile tag may be included in an IOR.

5. Unless otherwise specified in the definition of a particular component, multiple components with the same component tag may be part of a given profile within an IOR.
6. A **TAG_MULTIPLE_COMPONENTS** profile may hold components shared between multiple protocols. Multiple such profiles may exist in an IOR.
7. The definition of each protocol using a **TAG_MULTIPLE_COMPONENTS** profile must specify which components it uses, and how it uses them.
8. Profile and component definitions can be either public or private. Public definitions are those whose tag and data format is specified in OMG documents. For private definitions, only the tag is registered with OMG.
9. Public component definitions shall state whether or not they are intended for use by protocols other than the one(s) for which they were originally defined, and dependencies on other components.

The OMG is responsible for allocating and registering protocol and component tags. Neither allocation nor registration indicates any “standard” status, only that the tag will not be confused with other tags. Requests to allocate tags should be sent to tag_request@omg.org.

5.6.5 IOR Creation and Scope

IORs are created from object references when required to cross some kind of referencing domain boundary. ORBs will implement object references in whatever form they find appropriate, including possibly using the IOR structure. Bridges will normally use IORs to mediate transfers where that standard is appropriate.

5.6.6 Stringified Object References

Object references can be “stringified” (turned into an external string form) by the **ORB::object_to_string** operation, and then “destringified” (turned back into a programming environment’s object reference representation) using the **ORB::string_to_object** operation.

There can be a variety of reasons why being able to parse this string form might *not* help make an invocation on the original object reference:

- Identifiers embedded in the string form can belong to a different domain than the ORB attempting to destringify the object reference.
- The ORBs in question might not share a network protocol, or be connected.
- Security constraints may be placed on object reference destringification.

Nonetheless, there is utility in having a defined way for ORBs to generate and parse stringified IORs, so that in some cases an object reference stringified by one ORB could be destringified by another.

To allow a stringified object reference to be internalized by what may be a different ORB, a stringified IOR representation is specified. This representation instead establishes that ORBs could parse stringified object references using that format. This helps address the problem of bootstrapping, allowing programs to obtain and use object references, even from different ORBs.

The following is the representation of the stringified (externalized) IOR:

- (1) **<oref> ::= <prefix> <hex_Octets>**
- (2) **<prefix> ::= “IOR:”**

- (3) `<hex_Octets> ::= <hex_Octet> {<hex_Octet>}`*
- (4) `<hex_Octet> ::= <hexDigit> <hexDigit>`
- (5) `<hexDigit> ::= <digit> | <a> | <b> | <c> | <d> | <e> | <f>`
- (6) `<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" |`
`| "6" | "7" | "8" | "9"`
- (7) `<a> ::= "a" | "A"`
- (8) `<b> ::= "b" | "B"`
- (9) `<c> ::= "c" | "C"`
- (10) `<d> ::= "d" | "D"`
- (11) `<e> ::= "e" | "E"`
- (12) `<f> ::= "f" | "F"`

The hexadecimal strings are generated by first turning an object reference into an IOR, and then encapsulating the IOR using the encoding rules of CDR, as specified in GIOP 1.0. (See 6.2, 'CDR Transfer Syntax' for more information.) The content of the encapsulated IOR is then turned into hexadecimal digit pairs, starting with the first octet in the encapsulation and going until the end. The high four bits of each octet are encoded as a hexadecimal digit, then the low four bits.

5.6.7 Object Service Context

Emerging specifications for Object Services occasionally require service-specific context information to be passed implicitly with requests and replies. (Specifications for OMG's Object Services are contained in *CORBA Services: Common Object Service Specifications*). The Interoperability specifications define a mechanism for identifying and passing this service-specific context information as "hidden" parameters. The specification makes the following assumptions:

- Object Service specifications that need additional context passed will completely specify that context as an OMG IDL data type.
- ORB APIs will be provided that will allow services to supply and consume context information at appropriate points in the process of sending and receiving requests and replies.
- It is an ORB's responsibility to determine when to send service-specific context information, and what to do with such information in incoming messages. It may be possible, for example, for a server receiving a request to be unable to de-encapsulate and use a certain element of service-specific context, but nevertheless still be able to successfully reply to the message.

As shown in the following OMG IDL specification, the IOP module provides the mechanism for passing Object Service-specific information. It does not describe any service-specific information. It only describes a mechanism for transmitting it in the most general way possible. The mechanism is currently used by the DCE ESIOP and could also be used by the Internet Inter-ORB protocol (IIOP) General Inter_ORB Protocol (GIOP).

Each Object Service requiring implicit service-specific context to be passed through GIOP will be allocated a unique service context ID value by OMG. Service context ID values are of type **unsigned long**. Object service specifications are responsible for describing their context information as single OMG IDL data types, one data type associated with each service context ID.

The marshaling of Object Service data is described by the following OMG IDL:

```

module IOP {      // IDL

    typedef unsigned long      ServiceId;

    struct ServiceContext {
        ServiceId      context_id;
        sequence <octet>      context_data;
    };

    typedef sequence <ServiceContext>      ServiceContextList;

    const ServiceId      TransactionService = 0;
    const ServiceId      CodeSets = 1;
    const ServiceId      ChainBypassCheck = 2;
    const ServiceId      ChainBypassInfo = 3;
    const ServiceId      LogicalThreadId = 4;
    const ServiceId      BI_DIR_IOP = 5;
    const ServiceId      SendingContextRunTime = 6;
    const ServiceId      INVOCATION_POLICIES = 7;
    const ServiceId      FORWARDED_IDENTITY = 8;
    const ServiceId      UnknownExceptionInfo = 9;
};

```

The context data for a particular service will be encoded as specified for its service-specific OMG IDL definition, and that encoded representation will be encapsulated in the **context_data** member of **IOP::ServiceContext**. (See 6.2.2.9, 'Encapsulation'). The **context_id** member contains the service ID value identifying the service and data format. Context data is encapsulated in octet sequences to permit ORBs to handle context data without unmarshaling, and to handle unknown context data types.

During request and reply marshaling, ORBs will collect all service context data associated with the *Request* or *Reply* in a **ServiceContextList**, and include it in the generated messages. No ordering is specified for service context data within the list. The list is placed at the beginning of those messages to support security policies that may need to apply to the majority of the data in a request (including the message headers).

Each Object Service requiring implicit service-specific context to be passed through GIOP will be allocated a unique service context ID value by the OMG. Service context ID values are of type unsigned long. Object service

specifications are responsible for describing their context information as single OMG IDL data types, one data type associated with each service context ID.

The high-order 20 bits of service-context ID contain a 20-bit vendor service context codeset ID (VSCID); the low-order 12 bits contain the rest of the service context ID. A vendor (or group of vendors) who wish to define a specific set of system exception minor codes should obtain a unique VSCID from the OMG, and then define a specific set of service context IDs using the VSCID for the high-order bits.

The VSCID of zero is reserved for use for OMG-defined standard service context IDs (i.e., service context IDs in the range 0-4095 are reserved as OMG standard service contexts).

The **ServiceIds** currently defined are:

- **TransactionService** identifies a CDR encapsulation of the **CosTSInteroperation::PropogationContext** defined in *CORBAServices: Common Object Services Specifications*. (See 2.2, 'Other Specifications' for additional information).
- **CodeSets** identifies a CDR encapsulation of the **CONV_FRAME::CodeSetContext** defined in 5.7.2.5, 'GIOP Code Set Service Context'.
- COM-CORBA Interworking uses three service contexts as defined in "DCOM-CORBA Interworking" in the "Interoperability with non-CORBA Systems" chapter as described in CORBA 2.3 (see 2.2, 'Other Specifications' for additional information). They are:
 - **ChainBypassCheck**, which carries a CDR encapsulation of the **struct CosBridging::ChainBypassCheck**. This is carried only in a **Request** message as described in 20.9.1, "CORBA Chain Bypass" - CORBA 2.3.
 - **ChainBypassInfo**, which carries a CDR encapsulation of the **struct CosBridging::ChainBypassInfo**. This is carried only in a **Reply** message as described in 20.9.1, "CORBA Chain Bypass" - CORBA 2.3.
 - **LogicalThreadId**, which carries a CDR encapsulation of the **struct CosBridging::LogicalThreadId** as described in 20.10, "Thread Identification" - CORBA 2.3.
- **BI_DIR_IIOp** identifies a CDR encapsulation of the **IIOp::BiDirIIOpServiceContext** defined in 6.7, 'Bi-Directional GIOP'.
- **SendingContextRunTime** identifies a CDR encapsulation of the IOR of the **SendingContext::RunTime** object in 5.6, "Access to the Sending Context Run Time" - CORBA 2.3.
- **UnknownExceptionInfo** identifies a CDR encapsulation of a marshaled instance of a **java.lang.throwable** or one of its subclasses as described in Java to IDL Language Mapping.
- The **profile_data** for the **TAG_INTERNET_IOP** profile is a CDR encapsulation of the **IIOp::ProfileBody_1_1** type, described in 6.6.2, 'IIOp IOR Profiles'.
- The **profile_data** for the **TAG_MULTIPLE_COMPONENTS** profile is a CDR encapsulation of the **MultipleComponentProfile** type, which is a sequence of **TaggedComponent** structures, described in 5.6, 'An Information Model for Object References'.
- The **component_data member** identifies a CDR encapsulation of a **BindingNameComponent** structure, described in 16.5.2.1, "BindingNameComponent" - CORBA 2.3.

Service context IDs are associated with a specific version of GIOP, but will always be allocated in the OMG service context range. This allows any ORB to recognize when it is receiving a standard service context, even if it has been defined in a version of GIOP that it does not support.

The following are the rules for processing a received service context:

- The service context is in the OMG defined range:
 - If it is valid for the supported GIOP version, then it must be processed correctly according to the rules associated with it for that GIOP version level.
 - If it is not valid for the GIOP version, then it may be ignored by the receiving ORB, however it must be passed on through a bridge. No exception shall be raised.