
**Information technology — Object
Management Group — Common Object
Request Broker Architecture (CORBA) —
Part 2:
Interoperability**

*Technologies de l'information — OMG (Object Management Group) —
CORBA (Common Object Request Broker Architecture) —*

Partie 2: Interopérabilité



Reference number
ISO/IEC 19500-2:2012(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2012



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2012

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Table of Contents

Foreword	ix
Introduction	xi
1 Scope	1
2 Conformance and Compliance	1
2.1 Unreliable Multicast	2
3 Normative References	2
3.1 Other Specifications	3
4 Terms and definitions	3
4.1 Recommendations International Standards.....	4
4.2 Terms Defined in this Part of ISO/IEC 19500	4
4.3 Keywords for Requirement statements	5
5 Symbols (and abbreviated terms)	6
6 Interoperability Overview	7
6.1 General.....	7
6.2 Elements of Interoperability	7
6.2.1 ORB Interoperability Architecture	7
6.2.2 Inter-ORB Bridge Support	7
6.2.3 General Inter-ORB Protocol (GIOP)	8
6.2.4 Internet Inter-ORB Protocol (IIOP)®	8
6.2.5 Environment-Specific Inter-ORB Protocols (ESIOPs)	9
6.3 Relationship to Previous Versions of CORBA	9
6.4 Examples of Interoperability Solutions	10
6.4.1 Example 1	10
6.4.2 Example 2	10

6.4.3 Example 3	10
6.4.4 Interoperability Compliance	10
6.5 Motivating Factors	13
6.5.1 ORB Implementation Diversity	13
6.5.2 ORB Boundaries	13
6.5.3 ORBs Vary in Scope, Distance, and Lifetime	13
6.6 Interoperability Design Goals.....	14
6.6.1 Non-Goals	14
7 ORB Interoperability Architecture	15
7.1 Overview.....	15
7.1.1 Domains	15
7.1.2 Bridging Domains	15
7.2 ORBs and ORB Services.....	16
7.2.1 The Nature of ORB Services	16
7.2.2 ORB Services and Object Requests	16
7.2.3 Selection of ORB Services	17
7.3 Domains.....	17
7.3.1 Definition of a Domain	18
7.3.2 Mapping Between Domains: Bridging	19
7.4 Interoperability Between ORBs.....	19
7.4.1 ORB Services and Domains	19
7.4.2 ORBs and Domains	20
7.4.3 Interoperability Approaches	20
7.4.4 Policy-Mediated Bridging	22
7.4.5 Configurations of Bridges in Networks	22
7.5 Object Addressing	23
7.5.1 Domain-relative Object Referencing	24
7.5.2 Handling of Referencing Between Domains	24
7.6 An Information Model for Object References.....	25
7.6.1 What Information Do Bridges Need?	25
7.6.2 Interoperable Object References: IORs	25
7.6.3 IOR Profiles	26
7.6.4 Standard IOR Profiles	28
7.6.5 IOR Components	29
7.6.6 Standard IOR Components	29
7.6.7 Profile and Component Composition in IORs	31
7.6.8 IOR Creation and Scope	32
7.6.9 Stringified Object References	32

7.6.10 Object URLs	33
7.7 Service Context	37
7.7.1 Standard Service Contexts	38
7.7.2 Service Context Processing Rules	40
7.8 Coder/Decoder Interfaces	40
7.8.1 Codec Interface	40
7.8.2 Codec Factory	42
7.9 Feature Support and GIOP Versions.....	43
7.10 Code Set Conversion	45
7.10.1 Character Processing Terminology	45
7.10.2 Code Set Conversion Framework	48
7.10.3 Mapping to Generic Character Environments	54
7.10.4 Example of Generic Environment Mapping	56
7.10.5 Relevant OSFM Registry Interfaces	56
8 Building Inter-ORB Bridges	63
8.1 Introduction.....	63
8.2 In-Line and Request-Level Bridging	63
8.2.1 In-line Bridging	64
8.2.2 Request-level Bridging	64
8.2.3 Collocated ORBs	65
8.3 Proxy Creation and Management.....	66
8.4 Interface-specific Bridges and Generic Bridges	66
8.5 Building Generic Request-Level Bridges.....	66
8.6 Bridging Non-Referencing Domains	67
8.7 Bootstrapping Bridges	68
9 General Inter-ORB Protocol	69
9.1 Overview.....	69
9.2 Goals of the General Inter-ORB Protocol	69
9.3 GIOP Overview.....	69
9.3.1 Common Data Representation (CDR)	70
9.3.2 GIOP Message Overview	70
9.3.3 GIOP Message Transfer	71
9.4 CDR Transfer Syntax	71

9.4.1 Primitive Types	72
9.4.2 OMG IDL Constructed Types	77
9.4.3 Encapsulation	79
9.4.4 Value Types	80
9.4.5 Pseudo-Object Types	87
9.4.6 Object References	93
9.4.7 Abstract Interfaces	93
9.5 GIOP Message Formats	93
9.5.1 GIOP Message Header	94
9.5.2 Request Message	96
9.5.3 Reply Message	99
9.5.4 CancelRequest Message	102
9.5.5 LocateRequest Message	103
9.5.6 LocateReply Message	104
9.5.7 CloseConnection Message	106
9.5.8 MessageError Message	106
9.5.9 Fragment Message	106
9.6 GIOP Message Transport.....	107
9.6.1 Connection Management	108
9.6.2 Message Ordering	109
9.7 Object Location.....	110
9.8 Internet Inter-ORB Protocol (IIOP).....	111
9.8.1 TCP/IP Connection Usage	111
9.8.2 IIOP IOR Profiles	112
9.8.3 IIOP IOR Profile Components	114
9.9 Bi-Directional GIOP	115
9.9.1 Bi-directional IIOP	117
9.10 Bi-directional GIOP policy.....	118
9.11 OMG IDL.....	118
9.11.1 GIOP Module	118
9.11.2 IIOP Module	123
9.11.3 BiDirPolicy Module	124
10 Secure Interoperability	125
10.1 Overview.....	125
10.1.1 Assumptions	126
10.2 Protocol Message Definitions	127
10.2.1 The Security Attribute Service Context Element	127

10.2.2 SAS context_data Message Body Types	127
10.2.3 Authorization Token Format	132
10.2.4 Client Authentication Token Format	133
10.2.5 Identity Token Format	135
10.2.6 Principal Names and Distinguished Names	136
10.3 Security Attribute Service Protocol	137
10.3.1 Compound Mechanisms	137
10.3.2 Session Semantics	141
10.3.3 TSS State Machine	143
10.3.4 CSS State Machine	146
10.3.5 ContextError Values and Exceptions	149
10.4 Transport Security Mechanisms	150
10.4.1 Transport Layer Interoperability	150
10.4.2 Transport Mechanism Configuration	150
10.5 Interoperable Object References	151
10.5.1 Target Security Configuration	151
10.5.2 Client-side Mechanism Selection	160
10.5.3 Client-Side Requirements and Location Binding	161
10.5.4 Server Side Consideration	162
10.6 Conformance Levels	162
10.6.1 Conformance Level 0	162
10.6.2 Conformance Level 1	163
10.6.3 Conformance Level 2	163
10.6.4 Stateful Conformance	164
10.7 Sample Message Flows and Scenarios	164
10.7.1 Confidentiality, Trust in Server, and Trust in Client Established in the Connection	165
10.7.2 Confidentiality and Trust in Server Established in the Connection - Stateless Trust in Client Established in Service Context	167
10.7.3 Confidentiality, Trust in Server, and Trust in Client Established in the Connection Stateless Trust Association Established in Service Context	169
10.7.4 Confidentiality, Trust in Server, and Trust in Client Established in the Connection - Stateless Forward Trust Association Established in Service Context	172
10.8 References	173
10.9 IDL	174
10.9.1 Module GSSUP - Username/Password GSSAPI Token Formats	174
10.9.2 Module CSI - Common Secure Interoperability	175
10.9.3 Module CSIIOP - CSIV2 IOR Component Tag Definitions	179

11	Unreliable Multicast Inter-ORB Protocol	183
11.1	Introduction	183
11.1.1	Purpose	183
11.1.2	MIOP Packet	183
11.1.3	Packet Collection	183
11.1.4	PacketHeader	184
11.1.5	Joining an IP/Multicast Group	185
11.1.6	Quality Of Service	186
11.1.7	Delivery Requirements	186
11.2	MIOP Object Model	186
11.2.1	Definition	186
11.2.2	Unreliable IP/Multicast Profile Body (UIPMC_ProfileBody)	187
11.2.3	Group IOR	188
11.2.4	Extending PortableServer::POA to include Group Operations	190
11.2.5	MIOP Gateway	194
11.2.6	Multicast Group Manager	194
11.2.7	MIOP URL	210
11.3	Request Issues	211
11.3.1	GIOP Request Message Compatibility	211
11.3.2	MIOP Request Efficiency	211
11.3.3	Client Use Cases	212
11.3.4	Server Use Cases	213
11.4	Consolidated IDL	213
11.4.1	OMG IDL	213
Annex A	- Legal Information.....	221
Annex B	- Acknowledgements	225

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 19500-2 was prepared by Technical Committee ISO/IEC JTC1, Information technology, in collaboration with the Object Management Group (OMG), following the submission and processing as a Publicly Available Specification (PAS) of the OMG Common Object Request Broker Architecture (CORBA) specification Part 2 Version 3.1 CORBA Interoperability.

ISO/IEC 19500-2 is related to:

- 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 14750:1997, Information Technology - Open Distributed Processing - Interface Definition Language
- ISO/IEC 19500-2, Information Technology - Open Distributed Processing - CORBA Specification Part 1: CORBA Interfaces
- ISO/IEC 19500-3, Information Technology - Open Distributed Processing - CORBA Specification Part 3: CORBA Components

ISO/IEC 19500 consists of the following parts, under the general title *Information technology - Open distributed processing - CORBA specification*:

- Part 1: CORBA Interfaces
- Part 2: CORBA Interoperability
- Part 3: CORBA Components

ISO/IEC 19500-2:2012(E)

It is the common core of the CORBA specification. Optional parts of CORBA, such as mappings to particular programming languages, Real-time CORBA extensions, and the minimum CORBA profile for embedded systems are documented in the other specifications that together comprise the complete CORBA specification. Please visit the CORBA download page at http://www.omg.org/technology/documents/corba_spec_catalog.htm to find the complete CORBA specification set.

Apart from this Foreword, the text of this International Standard is identical with that for the OMG specification for CORBA, v3.1.1, Part 2.

Introduction

The rapid growth of distributed processing has led to a need for a coordinating framework for this standardization and 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.

RM-ODP Part 2 (ISO/IEC 10746-2) defines the foundational concepts and modeling framework for describing distributed systems. The scopes and objectives of the RM-ODP Part 2 and the UML, while related, are not the same and, in a number of cases, the RM-ODP Part 2 and the UML specification use the same term for concepts which are related but not identical (e.g., interface). Nevertheless, a specification using the Part 2 modeling concepts can be expressed using UML with appropriate extensions (using stereotypes, tags, and constraints).

RM-ODP Part 3 (ISO/IEC 10746-3) specifies a generic architecture of open distributed systems, expressed using the foundational concepts and framework defined in Part 2. Given the relation between UML as a modeling language and Part 3 of the RM-ODP standard, it is easy to show that UML is suitable as a notation for the individual viewpoint specifications defined by the RM-ODP.

This International Standard for CORBA Interfaces is a standard for the technology specification of an ODP system. It defines a technology to provide the infrastructure required to support functional distribution of an ODP system, specifying functions required to manage physical distribution, communications, processing and storage, and the roles of different technology objects in supporting those functions.

Context of CORBA

The key to understanding the structure of the CORBA architecture is the Reference Model, which consists of the following components:

- **Object Request Broker**, which enables objects to transparently make and receive requests and responses in a distributed environment. It is the foundation for building applications from distributed objects and for interoperability between applications in hetero- and homogeneous environments. The architecture and specifications of the Object Request Broker are described in this manual.
- **Object Services**, a collection of services (interfaces and objects) that support basic functions for using and implementing objects. Services are necessary to construct any distributed application and are always independent of application domains. For example, the Life Cycle Service defines conventions for creating, deleting, copying, and moving objects; it does not dictate how the objects are implemented in an application. Specifications for Object Services are contained in *CORBA services: Common Object Services Specification*.
- **Common Facilities**, a collection of services that many applications may share, but which are not as fundamental as the Object Services. For instance, a system management or electronic mail facility could be classified as a common facility. Information about Common Facilities will be contained in *CORBA facilities: Common Facilities Architecture*.
- **Application Objects**, which are products of a single vendor or in-house development group that controls their interfaces. Application Objects correspond to the traditional notion of applications, so they are not standardized by OMG. Instead, Application Objects constitute the uppermost layer of the Reference Model.

The Object Request Broker, then, is the core of the Reference Model. It is like a telephone exchange, providing the basic mechanism for making and receiving calls. Combined with the Object Services, it ensures meaningful communication between CORBA-compliant applications.

The architecture and specifications described in this standard are aimed at software designers and developers who want to produce applications that comply with OMG specifications for the Object Request Broker (ORB), or this standard (ISO/IEC 19500). The benefit of compliance is, in general, to be able to produce interoperable applications that are based on distributed, interoperating objects. The ORB provides the mechanisms by which objects transparently make requests and receive responses. Hence, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

This Part of this International Standard includes a non-normative annex.

Information technology - Object Management Group Common Object Request Broker Architecture (CORBA), Interoperability

1 Scope

This part of ISO/IEC 19500 specifies a comprehensive, flexible approach to supporting networks of objects that are distributed across and managed by multiple, heterogeneous CORBA-compliant Object Request Brokers (ORBs). The approach to inter-ORB operation is universal, because elements can be combined in many ways to satisfy a very broad range of needs.

This part of ISO/IEC 19500 covers the specification of:

- ORB interoperability architecture
- Inter-ORB bridge support
- 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.
- 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).
- The CORBA Security Attribute Service (SAS) protocol and its use within the CSIV2 architecture to address the requirements of CORBA security for interoperable authentication, delegation, and privileges.

This part of ISO/IEC 19500 provides a widely implemented and used particularization of ITU-T Rec. X.931 | ISO/IEC 14752. Open Distributed Processing - Protocol Support for Computational Interactions. It supports interoperability and location transparency in ODP systems.

2 Conformance and Compliance

An ORB is considered to be interoperability-compliant when it meets the following requirements:

- In the CORBA Core part, standard APIs are provided by an ORB to enable the construction of request-level inter-ORB bridges. APIs are defined by the Dynamic Invocation Interface, the Dynamic Skeleton Interface, and by the object identity operations described in the Interface Repository clause of this book.
- An Internet Inter-ORB Protocol (IIOP) (explained in the Building Inter-ORB Bridges clause) defines a transfer syntax and message formats (described independently as the General Inter-ORB Protocol), and defines how to transfer messages via TCP/IP connections. The IIOP can be supported natively or via a halfbridge.

Support for additional Environment Specific Inter-ORB Protocols (ESIOPs) and other proprietary protocols is optional in an interoperability-compliant system. However, any implementation that chooses to use the other protocols defined by the CORBA interoperability specifications must adhere to those specifications to be compliant with CORBA interoperability.

Figure 6.2 on page 12 shows examples of interoperable ORB domains that are CORBA-compliant. These compliance points support a range of interoperability solutions. For example, the standard APIs may be used to construct “half bridge” to the IIOP, relying on another “half bridge” to connect to another ORB. The standard APIs also support

construction of “full bridges,” without using the Internet IOP to mediate between separated bridge components. ORBs may also use the Internet IOP internally. In addition, ORBs may use GIOP messages to communicate over other network protocol families (such as Novell or OSI), and provide transport-level bridges to the IIOP.

The GIOP is described separately from the IIOP to allow future specifications to treat it as an independent compliance point.

2.1 Unreliable Multicast

Summary of Optional Verses Mandatory Interfaces

An interface to an MIOP gateway should be considered an optional interface within the MIOP specification.

Proposed Compliance Points

The MIOP specification is a single, optional compliance point within the CORBA Core specification.

Changes to Other OMG Specifications

This part of ISO/IEC 19500 contains an extension to the IOP module.

```
module IOP {  
    const ProfileId    TAG_UIPMC = 3;  
    const ComponentId TAG_GROUP = 39;  
    const ComponnetId TAG_GROUP_IIOF = 40  
};
```

3 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.

- ITU-T Recommendation X.902 (1995) | ISO/IEC 10746-2:1996, Open Distributed Processing - Reference Model: Foundations
- ITU-T Recommendation X.903 (1995) | ISO/IEC 10746-3:1996, Open Distributed Processing - Reference Model: Architecture
- ITU-T Recommendation X.920 (1999) | ISO/IEC 10750:1999, Open Distributed Processing - Interface Definition Language
- ITU-T Recommendation X.931(2000) | ISO/IEC 14752:2000, 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 10646-1:1993 Information Technology - Universal Multiple-Octect coded character set (UCS) - Part 1: Architecture and Basic Multilingual Plane

- ISO/IEC 10646-1: 1993/Amd 1:1996 Transformation Format for 16 planes of group 00 (UTF - 16)
- ISO/IEC 10646-1: 1993/Amd 2:1996 UCS Transformation Format 8 (UTF - 8)
- ISO/IEC 19500-1: 2011 Open Distributed Processing - CORBA Specification Part 1: CORBA Interfaces, pas/2011-08-07

3.1 Other Specifications

- 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.
- [JAV2I]Object Management Group, "Java to IDL," available from <http://www.omg.org/spec/JAV2I/1.4>
- [CORBASEC]Object Management Group, "Security Service," available from <http://www.omg.org/spec/SEC/>
- [ASMOTS]Object Management Group, "Additional Structuring Mechanisms for the OTS," available from <http://www.omg.org/spec/OTS/>
- [TRANS]Object Management Group, "Transaction Service," available from <http://www.omg.org/spec/TRANS/>
- [FIREWALL]Object Management Group, "CORBA Firewall Traversal Specification," available from <http://www.omg.org/members/cgi-bin/doc?ptc/04-04-05.pdf>
- [SCCP] Object Management Group, "CORBA / TC Interworking and SCCP-Inter ORB Protocol (SCCP)," available from <http://www.omg.org/spec/SCCP>
- [FTCORBA] Object Management Group, "Fault Tolerant Corba," clause 23 of CORBA 3.0.3, available from <http://www.omg.org/cgi-bin/doc?formal/2004-03-01>
- [RTCORBA] Object Management Group, "Real-Time CORBA, version 1.2," available from <http://www.omg.org/spec/RT/>
- [WATM] Object Management Group, "Wireless Access and Telecom Mobility in CORBA, Version 1.2," available from <http://www.omg.org/spec/WATM/>
- [DCOMI] Object Management Group, "Interoperability with non-CORBA Systems" clause 20 of CORBA 3.0.3, available from <http://www.omg.org/cgi-bin/doc?formal/2004-03-01>
- [TSAS] Object Management Group, "Telecommunications Service Access and Subscription Specification," available from <http://www.omg.org/spec/TSAS/>
- IETF RFC2119, "Key words for use in RFCs to Indicate Requirement Levels," S. Bradner, March 1997 (<http://ietf.org/rfc/rfc2119>)

4 Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

4.1 Recommendations | International Standards

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

- behavior
- interface
- instance
- object
- service
- state
- transparency
- type

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

- operation
- stub

4.2 Terms Defined in this Part of ISO/IEC 19500

adapter	Same as object adapter.
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.
client	The code or process that invokes an operation on an object.
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.)
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.
dynamic invocation	Constructing and issuing a request whose signature is possibly not known until run-time.
dynamic skeleton	An interface-independent kind of skeleton, used by servers to handle requests whose signatures are possibly not known until run-time.

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.
interface repository	A storage place for interface information.
interface type	A type satisfied by any object that satisfies a particular interface.
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.
language binding or mapping	The means and conventions by which a programmer writing in a specific programming language accesses ORB capabilities.
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.
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.
object implementation	Same as implementation.
object reference	A value that unambiguously identifies an object. Object references are never reused to identify another object.
objref	An abbreviation for object reference
ORB core	The ORB component which moves a request from a client to the appropriate adapter for the target object.
request	A message issued by a client to cause a service to be performed.
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.
server	A process implementing one or more operations on one or more objects.
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.
skeleton	The object-interface-specific ORB component which assists an object adapter in passing requests to particular methods.
synchronous request	A request where the client pauses to wait for completion of the request. Contrast with deferred synchronous request and one-way request.
value	Any entity that may be a possible actual parameter in a request. Values that serve to identify objects are called object references.

4.3 Keywords for Requirement statements

The keywords “must,” “must not,” “shall,” “shall not,” “should,” “should not,” and “may” in this International Standard are to be interpreted as described in IETF RFC 2119.

5 Symbols (and abbreviated terms)

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

ADT	Abstract Data Type
CSIV2	Common Secure Interoperability, version 2
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
ESIOP	Environment Specific Inter-ORB Protocol
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
SAS	Security Attribute Service
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

6 Interoperability Overview

6.1 General

ORB interoperability specifies a comprehensive, flexible approach to supporting networks of objects that are distributed across and managed by multiple, heterogeneous CORBA-compliant ORBs. The approach to “interORBability” is universal because its elements can be combined in many ways to satisfy a very broad range of needs.

6.2 Elements of Interoperability

The elements of interoperability are as follows:

- ORB interoperability architecture
- Inter-ORB bridge support
- General and Internet Inter-ORB Protocols (GIOPs and IIOPs)

In addition, the architecture accommodates Environment Specific Inter-ORB Protocols (ESIOPs) that are optimized for particular environments such as DCE.

6.2.1 ORB Interoperability Architecture

The ORB Interoperability Architecture provides a conceptual framework for defining the elements of interoperability and for identifying its compliance points. It also characterizes new mechanisms and specifies conventions necessary to achieve interoperability between independently produced ORBs.

Specifically, the architecture introduces the concepts of *immediate* and *mediated bridging* of ORB domains. The Internet Inter-ORB Protocol (IIOP) forms the common basis for broad-scope mediated bridging. The inter-ORB bridge support can be used to implement both immediate bridges and to build “half-bridges” to mediated bridge domains.

By use of bridging techniques, ORBs can interoperate without knowing any details of that ORB’s implementation, such as what particular IPC or protocols (such as ESIOPs) are used to implement the *CORBA* specification.

The IIOP may be used in bridging two or more ORBs by implementing “half bridges” that communicate using the IIOP. This approach works for both stand-alone ORBs, and networked ones that use an ESIOP.

The IIOP may also be used to implement an ORB’s internal messaging, if desired. Since ORBs are not required to use the IIOP internally, the goal of not requiring prior knowledge of each others’ implementation is fully satisfied.

6.2.2 Inter-ORB Bridge Support

The interoperability architecture clearly identifies the role of different kinds of domains for ORB-specific information. Such domains can include object reference domains, type domains, security domains (e.g., the scope of a *Principal* identifier), a transaction domain, and more.

Where two ORBs are in the same domain, they can communicate directly. In many cases, this is the preferable approach. This is not always true, however, since organizations often need to establish local control domains.

When information in an invocation must leave its domain, the invocation must traverse a bridge. The role of a bridge is to ensure that content and semantics are mapped from the form appropriate to one ORB to that of another, so that users of any given ORB only see their appropriate content and semantics.

The inter-ORB bridge support element specifies ORB APIs and conventions to enable the easy construction of interoperability bridges between ORB domains. Such bridge products could be developed by ORB vendors, Sieves, system integrators, or other third-parties.

Because the extensions required to support Inter-ORB Bridges are largely general in nature, do not impact other ORB operation, and can be used for many other purposes besides building bridges, they are appropriate for all ORBs to support. Other applications include debugging, interposing of objects, implementing objects with interpreters and scripting languages, and dynamically generating implementations.

The inter-ORB bridge support can also be used to provide interoperability with non-CORBA systems, such as Microsoft's Component Object Model (COM). The ease of doing this will depend on the extent to which those systems conform to the CORBA Object Model.

6.2.3 General Inter-ORB Protocol (GIOP)

The General Inter-ORB Protocol (GIOP) element specifies a standard transfer syntax (low-level data representation) and a set of message formats for communications between ORBs. The GIOP is specifically built for ORB to ORB interactions and is designed to work directly over any connection-oriented transport protocol that meets a minimal set of assumptions. It does not require or rely on the use of higher level RPC mechanisms. The protocol is simple, scalable, and relatively easy to implement. It is designed to allow portable implementations with small memory footprints and reasonable performance, with minimal dependencies on supporting software other than the underlying transport layer.

While versions of the GIOP running on different transports would not be directly interoperable, their commonality would allow easy and efficient bridging between such networking domains.

6.2.4 Internet Inter-ORB Protocol (IIOP)®

The Internet Inter-ORB Protocol (IIOP)® element specifies how GIOP messages are exchanged using TCP/IP connections. The IIOP specifies a standardized interoperability protocol for the Internet, providing “out of the box” interoperation with other compatible ORBs based on the most popular product- and vendor-neutral transport layer. It can also be used as the protocol between half-bridges (see below).

The protocol is designed to be suitable and appropriate for use by any ORB to interoperate in Internet Protocol domains unless an alternative protocol is necessitated by the specific design center or intended operating environment of the ORB. In that sense it represents the basic inter-ORB protocol for TCP/IP environments, a most pervasive transport layer.

The IIOP's relationship to the GIOP is similar to that of a specific language mapping to OMG IDL; the GIOP may be mapped onto a number of different transports, and specifies the protocol elements that are common to all such mappings. The GIOP by itself, however, does not provide complete interoperability, just as IDL cannot be used to build complete programs. The IIOP and other similar mappings to different transports, are concrete realizations of the abstract GIOP definitions, as shown in Figure 6.1 on page 9.

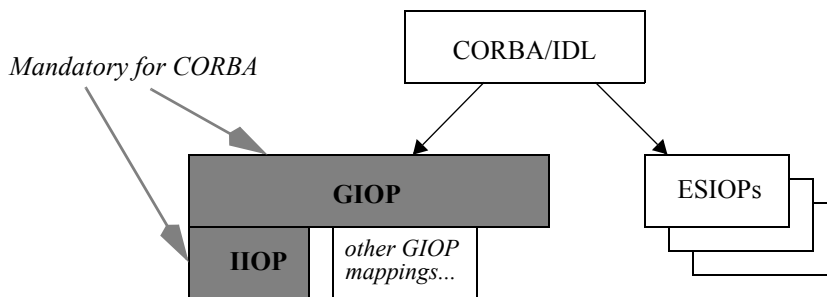


Figure 6.1 - Inter-ORB Protocol Relationships

6.2.5 Environment-Specific Inter-ORB Protocols (ESIOPs)

This part of ISO/IEC 19500 also makes provision for an open-ended set of Environment-Specific Inter-ORB Protocols (ESIOPs). Such protocols would be used for “out of the box” interoperability at user sites where a particular networking or distributing computing infrastructure is already in general use.

Because of the opportunity to leverage and build on facilities provided by the specific environment, ESIOPs might support specialized capabilities such as those relating to security and administration.

While ESIOPs may be optimized for particular environments, all ESIOP specifications will be expected to conform to the general ORB interoperability architecture conventions to enable easy bridging. The inter-ORB bridge support enables bridges to be built between ORB domains that use the IIOP and ORB domains that use a particular ESIOP.

6.3 Relationship to Previous Versions of CORBA

The ORB Interoperability Architecture builds on Common Object Request Broker Architecture by adding the notion of ORB Services and their domains. (ORB Services are described in “ORBs and ORB Services” on page 16). The architecture defines the problem of ORB interoperability in terms of bridging between those domains, and defines several ways in which those bridges can be constructed. The bridges can be internal (in-line) and external (request-level) to ORBs.

APIs included in the interoperability specifications include compatible extensions to previous versions of *CORBA* to support request-level bridging:

- A Dynamic Skeleton Interface (DSI) is the basic support needed for building request-level bridges. It is the server-side analogue of the Dynamic Invocation Interface and in the same way it has general applicability beyond bridging. For information about the Dynamic Skeleton Interface, refer to the *Dynamic Skeleton Interface* clause.
- APIs for managing object references have been defined, building on the support identified for the Relationship Service. The APIs are defined in Object Reference Operations in the *ORB Interface* clause of Part 1 of this International Standard (ISO/IEC 19500-1). The Relationship Service is described in the *Relationship Service* specification; refer to *CosObjectIdentity Module* in that specification.

6.4 Examples of Interoperability Solutions

The elements of interoperability (Inter-ORB Bridges, General and Internet Inter-ORB Protocols, Environment-Specific Inter-ORB Protocols) can be combined in a variety of ways to satisfy particular product and customer needs. This sub clause provides some examples.

6.4.1 Example 1

ORB product A is designed to support objects distributed across a network and provide “out of the box” interoperability with compatible ORBs from other vendors. In addition it allows bridges to be built between it and other ORBs that use environment-specific or proprietary protocols. To accomplish this, ORB A uses the IIOP and provides inter-ORB bridge support.

6.4.2 Example 2

ORB product B is designed to provide highly optimized, very high-speed support for objects located on a single machine. For example, to support thousands of Fresco GUI objects operated on at near function-call speeds. In addition, some of the objects will need to be accessible from other machines and objects on other machines will need to be infrequently accessed. To accomplish this, ORB A provides a half-bridge to support the Internet IOP for communication with other “distributed” ORBs.

6.4.3 Example 3

ORB product C is optimized to work in a particular operating environment. It uses a particular environment-specific protocol based on distributed computing services that are commonly available at the target customer sites. In addition, ORB C is expected to interoperate with other arbitrary ORBs from other vendors. To accomplish this, ORB C provides inter-ORB bridge support and a companion half-bridge product (supplied by the ORB vendor or some third-party) provides the connection to other ORBs. The half-bridge uses the IIOP to enable interoperability with other compatible ORBs.

6.4.4 Interoperability Compliance

An ORB is considered to be interoperability-compliant when it meets the following requirements:

- In the CORBA Core part, standard APIs are provided by an ORB to enable the construction of request-level inter-ORB bridges. APIs are defined by the Dynamic Invocation Interface, the Dynamic Skeleton Interface, and by the object identity operations described in the *Interface Repository* clause of Part 1 of this International Standard (ISO/IEC 19500-1).
- An Internet Inter-ORB Protocol (IIOP) (explained in the *Building Inter-ORB Bridges* clause) defines a transfer syntax and message formats (described independently as the General Inter-ORB Protocol), and defines how to transfer messages via TCP/IP connections. The IIOP can be supported natively or via a half-bridge.

Support for additional ESIOPs and other proprietary protocols is optional in an interoperability-compliant system. However, any implementation that chooses to use the other protocols defined by the CORBA interoperability specifications must adhere to those specifications to be compliant with CORBA interoperability.

Figure 6.2 on page 12 shows examples of interoperable ORB domains that are CORBA-compliant.

These compliance points support a range of interoperability solutions. For example, the standard APIs may be used to construct “half bridges” to the IIOP, relying on another “half bridge” to connect to another ORB. The standard APIs also support construction of “full bridges,” without using the Internet IOP to mediate between separated bridge components. ORBs may also use the Internet IOP internally. In addition, ORBs may use GIOP messages to communicate over other network protocol families (such as Novell or OSI), and provide transport-level bridges to the IIOP.

The GIOP is described separately from the IIOP to allow future specifications to treat it as an independent compliance point.

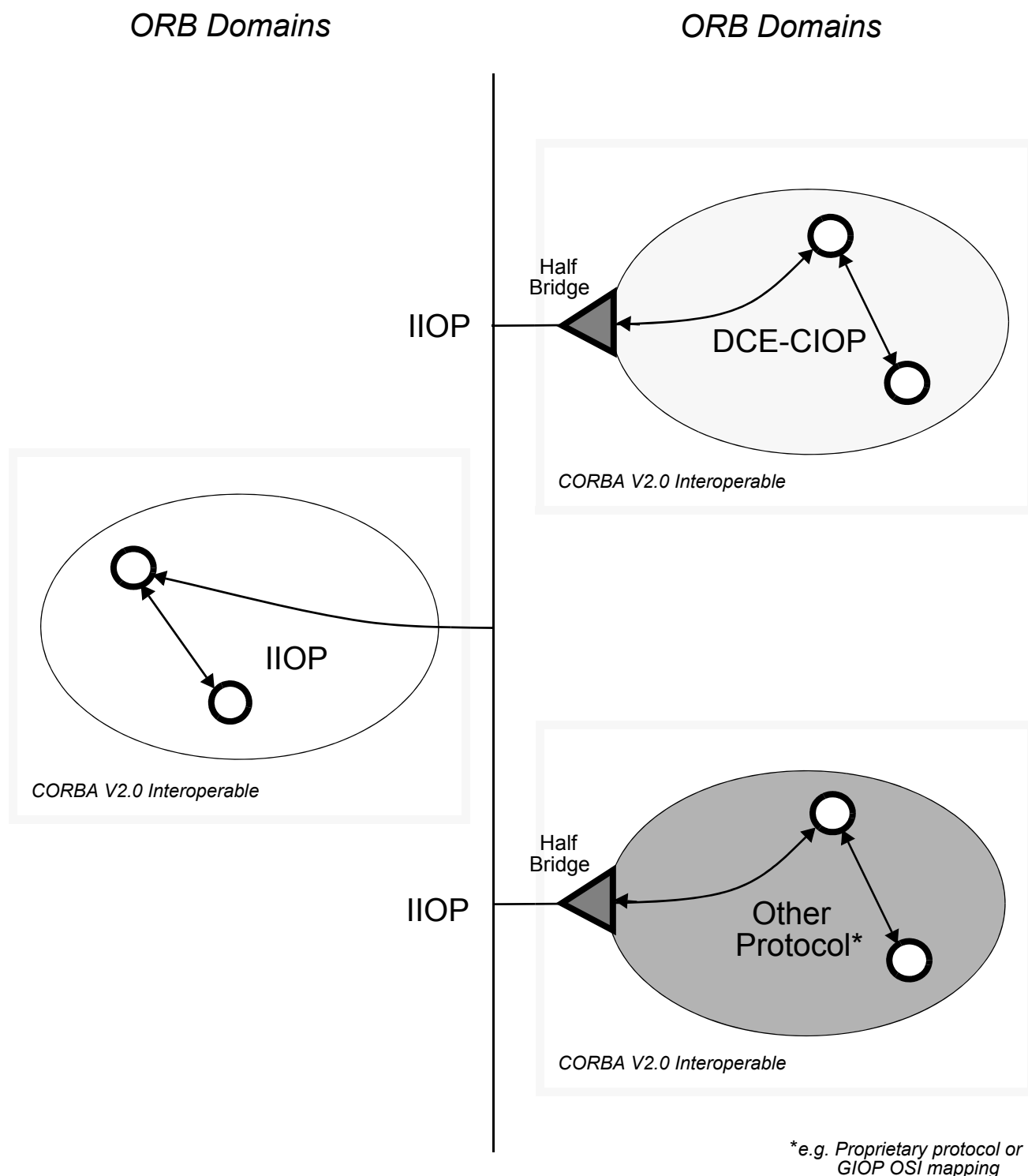


Figure 6.2 - Examples of CORBA Interoperability Compliance

6.5 Motivating Factors

This sub clause explains the factors that motivated the creation of interoperability specifications.

6.5.1 ORB Implementation Diversity

Today, there are many different ORB products that address a variety of user needs. A large diversity of implementation techniques is evident. For example, the time for a request ranges over at least 5 orders of magnitude, from a few microseconds to several seconds. The scope ranges from a single application to enterprise networks. Some ORBs have high levels of security, others are more open. Some ORBs are layered on a particular widely used protocol, others use highly optimized, proprietary protocols.

The market for object systems and applications that use them will grow as object systems are able to be applied to more kinds of computing. From application integration to process control, from loosely coupled operating systems to the information superhighway, CORBA-based object systems can be the common infrastructure.

6.5.2 ORB Boundaries

Even when it is not required by implementation differences, there are other reasons to partition an environment into different ORBs.

For security reasons, it may be important to know that it is not generally possible to access objects in one domain from another. For example, an “internet ORB” may make public information widely available, but a “company ORB” will want to restrict what information can get out. Even if they used the same ORB implementation, these two ORBs would be separate, so that the company could allow access to public objects from inside the company without allowing access to private objects from outside. Even though individual objects should protect themselves, prudent system administrators will want to avoid exposing sensitive objects to attacks from outside the company.

Supporting multiple ORBs also helps handle the difficult problem of testing and upgrading the object system. It would be unwise to test new infrastructure without limiting the set of objects that might be damaged by bugs, and it may be impractical to replace “the ORB” everywhere simultaneously. A new ORB might be tested and deployed in the same environment, interoperating with the existing ORB until either a complete switch is made or it incrementally displaces the existing one.

Management issues may also motivate partitioning an ORB. Just as networks are subdivided into domains to allow decentralized control of databases, configurations, resources, management of the state in an ORB (object reference location and translation information, interface repositories, per-object data) might also be done by creating sub-ORBs.

6.5.3 ORBs Vary in Scope, Distance, and Lifetime

Even in a single computing environment produced by a single vendor, there are reasons why some of the objects an application might use would be in one ORB, and others in another ORB. Some objects and services are accessed over long distances, with more global visibility, longer delays, and less reliable communication. Other objects are nearby, are not accessed from elsewhere, and provide higher quality service. By deciding which ORB to use, an implementor sets expectations for the clients of the objects.

One ORB might be used to retain links to information that is expected to accumulate over decades, such as library archives. Another ORB might be used to manage a distributed chess program in which the objects should all be destroyed when the game is over. Although while it is running, it makes sense for “chess ORB” objects to access the “archives ORB,” we would not expect the archives to try to keep a reference to the current board position.

6.6 Interoperability Design Goals

Because of the diversity in ORB implementations, multiple approaches to interoperability are required. Options identified in previous versions of *CORBA* include:

- *Protocol Translation*, where a gateway residing somewhere in the system maps requests from the format used by one ORB to that used by another.
- *Reference Embedding*, where invocation using a native object reference delegates to a special object whose job is to forward that invocation to another ORB.
- *Alternative ORBs*, where ORB implementations agree to coexist in the same address space so easily that a client or implementation can transparently use any of them, and pass object references created by one ORB to another ORB without losing functionality.

In general, there is no single protocol that can meet everyone's needs, and there is no single means to interoperate between two different protocols. There are many environments in which multiple protocols exist, and there are ways to bridge between environments that share no protocols.

This International Standard adopts a flexible architecture that allows a wide variety of ORB implementations to interoperate and that includes both bridging and common protocol elements.

The following goals guided the creation of interoperability specifications:

- The architecture and specifications should allow high-performance, small footprint, lightweight interoperability solutions.
- The design should scale, should not be unduly difficult to implement, and should not unnecessarily restrict implementation choices.
- Interoperability solutions should be able to work with any vendors' existing ORB implementations with respect to their CORBA-compliant core feature set; those implementations are diverse.
- All operations implied by the CORBA object model (i.e., the stringify and destringify operations defined on the **CORBA:ORB** pseudo-object and all the operations on **CORBA:Object**) as well as type management (e.g., narrowing, as needed by the C++ mapping) should be supported.

6.6.1 Non-Goals

The following were taken into account, but were not goals:

- Support for security
- Support for future ORB Services

7 ORB Interoperability Architecture

7.1 Overview

The original Interoperability RFP defines interoperability as the ability for a client on ORB A to invoke an OMG 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 implementation.
- 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.

7.1.1 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.

7.1.2 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.

7.2 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 part of ISO/IEC 19500 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.

7.2.1 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 that 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.

7.2.2 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.

7.2.3 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.

7.3 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.

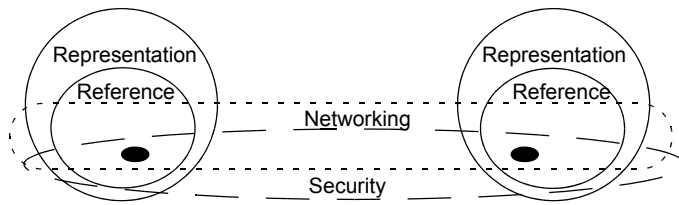


Figure 7.1 - Different Kinds of Domains can Coexist

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 modeling concept that 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).

7.3.1 Definition of a Domain

Domains allow partitioning of systems into collections of components that 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 modeled 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.

7.3.2 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 that 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 part of ISO/IEC 19500 assumes that both domains are strictly compliant with the CORBA Object Model and the *CORBA* specifications. This includes the use of OMG IDL 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.

7.4 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 text focuses on bridging in only one direction. This is purely to simplify discussions, and does not imply that unidirectional connectivity satisfies basic interoperability requirements.

7.4.1 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).

7.4.2 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.

7.4.3 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 sub clauses.

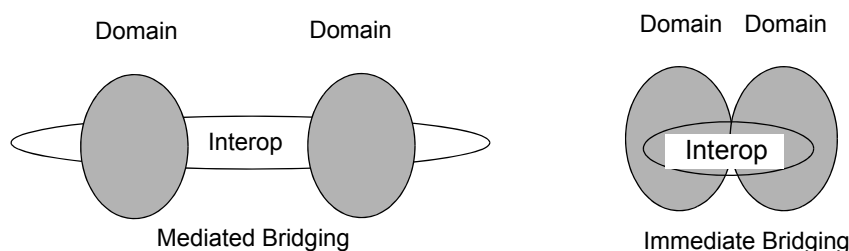


Figure 7.2 - Two bridging techniques, different uses of an intermediate form agreed on between the two domains.

7.4.3.1 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.

7.4.3.2 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 that 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.

7.4.3.3 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 that 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.

7.4.3.4 Bridging Level

As noted at the start of this sub clause, 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.

7.4.4 Policy-Mediated Bridging

An assumption made through most of this part of ISO/IEC 19500 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.

7.4.5 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.

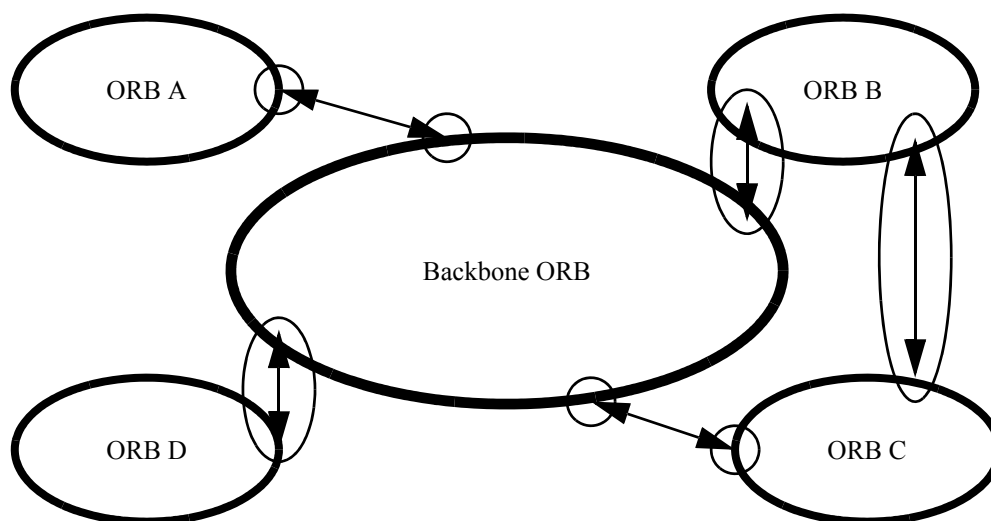


Figure 7.3 - An ORB chosen as a backbone will connect other ORBs through bridges, both full-bridges and half-bridges.

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.

7.5 Object Addressing

The Object Model in "The Object Model" clause of Part 1 of this International Standard (ISO/IEC 19500-1) 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 sub clause 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.

7.5.1 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.

7.5.2 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 $Dn+1$.

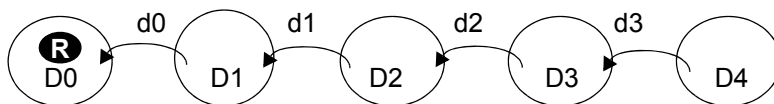


Figure 7.4 - Reference encapsulation adds domain information during bridging

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 $(dn-1, xn-1)$.

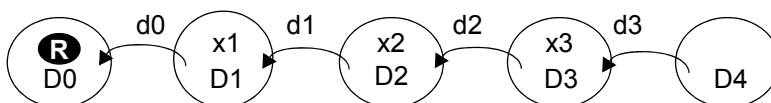


Figure 7.5 - Domain Reference Translation substitutes domain references during bridging

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.

7.6 An Information Model for Object References

This sub clause 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 *General Inter-ORB Protocol* clause of this standard, *Object References*.

7.6.1 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 Selection of ORB Services on page 17, 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.

7.6.2 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;

    typedef CORBA::OctetSeq ProfileData;
    struct TaggedProfile {
        ProfileId tag;
        ProfileData profile_data;
    };
    typedef sequence <TaggedProfile> TaggedProfileSeq ;

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

    struct IOR {
        string type_id;
        TaggedProfileSeq profiles;
    };

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

    typedef unsigned long ComponentId;
    typedef CORBA::OctetSeq ComponentData;

    struct TaggedComponent {
        ComponentId tag;
        ComponentData component_data;
    };

    typedef sequence<TaggedComponent> TaggedComponentSeq;
    typedef sequence <TaggedComponent> MultipleComponentProfile;
    typedef CORBA::OctetSeq ObjectKey;
};

```

7.6.3 IOR Profiles

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.

When a specific protocol is used to convey an object reference passed as a parameter in an IDL operation invocation (or reply), an IOR which reflects, in its contained profiles, the full protocol understanding of the operation client (or server in case of reply) may be sent. A receiving ORB which operates (based on topology and policy information available to it) on

profiles rather than the received IOR as a whole, to create a derived reference for use in its own domain of reference, is placing itself as a bridge between reference domains. Interoperability inhibiting situations can arise when an orb sends an IOR with multiple profiles (using one of its supported protocols) to a receiving orb, and that receiving orb later returns a derived reference to that object, which has had profiles or profile component data removed or transformed from the original IOR contents.

To assist in classifying behavior of ORBS in such bridging roles, two classes of IOR conformance may be associated with the conformance requirements for a given ORB interoperability protocol:

- Full IOR conformance requires that an orb which receives an IOR for an object passed to it through that ORB interoperability protocol, shall recover the original IOR, in its entirety, for passing as a reference to that object from that orb through that same protocol.
- Limited-Profile IOR conformance requires that an orb which receives an IOR passed to it through a given ORB interoperability protocol, shall recover all of the standard information contained in the IOR profile for that protocol, whenever passing a reference to that object, using that same protocol, to another ORB.

NOTE: Conformance to IIOP versions 1.0, 1.1, and 1.2 only requires support of limited-Profile IOR conformance, specifically for the IIOP IOR profile. However, due to interoperability problems induced by Limited-Profile IOR conformance, it is now deprecated by the CORBA 2.4 specification for an orb to not support Full IOR conformance. Some future IIOP versions could require Full IOR conformance.

An ORB may be unable to use any of the profiles provided in an IOR for various reasons which may be broadly categorized as transient ones like temporary network outage, and non-transient ones like unavailability of appropriate protocol software in the ORB. The decision about the category of outage that causes an ORB to be unable to use any profile from an IOR is left up to the ORB. At an appropriate point, when an ORB discovers that it is unable to use any profile in an IOR, depending on whether it considers the reason transient or non-transient, it should raise the standard system exception **TRANSIENT** with standard minor code 2, or **IMP_LIMIT** with the standard minor code 1.

Each profile has a unique numeric tag, assigned by the OMG. The ones defined here are for the IIOP (see clause 9.8.3, IIOP IOR Profile Components, on page 114) 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 that contains only a single terminating character). 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.

ORBs claiming to support the Full-IOR conformance are required to preserve all the semantic content of any IOR (including the ordering of each profile and its components), and may only apply transformations which preserve semantics (e.g., changing Byte order for encapsulation).

For example, consider an echo operation for object references:

```
interface Echoer {Object echo(in Object o);};
```

Assume that the method body implementing this “echo” operation simply returns its argument. When a client application invokes the echo operation and passes an arbitrary object reference, if both the client and server ORBs claim support to Full IOR conformance, the reference returned by the operation is guaranteed to have not been semantically altered by either client or server ORB. That is, all its profiles will remain intact and in the same order as they were present when the reference was sent. This requirement for ORBs which claim support for Full-IOR conformance, ensures that, for example, a client can safely store an object reference in a naming service and get that reference back again later without losing information inside the reference.

7.6.4 Standard IOR Profiles

```
module IOP {
    const ProfileId TAG_INTERNET_IOP = 0;
    const ProfileId TAG_MULTIPLE_COMPONENTS = 1;
    const ProfileId TAG_SCCP_IOP = 2;
    const ProfileId TAG_UIPMC = 3;
};
```

7.6.4.1 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 IOP IOR Profiles on page 112, contains a CDR encapsulation of a structure containing addressing and object identification information used by IOP. Version 1.1 of the **TAG_INTERNET_IOP** profile also includes a **sequence<TaggedComponent>** that can contain additional information supporting optional IOP features, ORB services such as security, and future protocol extensions.

Protocols other than IOP (such as ESIOPs and other GIOPs) can share profile information (such as object identity or security information) with IOP by encoding their additional profile information as components in the **TAG_INTERNET_IOP** profile. All **TAG_INTERNET_IOP** profiles support IOP, 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 IOP.

The **profile_data** for the **TAG_INTERNET_IOP** profile is a CDR encapsulation of the **IOP::ProfileBody_1_1** type, described in IOP IOR Profiles on page 112.

7.6.4.2 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, the use of which must be specified by the protocol using this profile. This profile may be used to carry IOR components, as specified in IOR Components on page 29.

The **profile_data** for the **TAG_MULTIPLE_COMPONENTS** profile is a CDR encapsulation of the **MultipleComponentProfile** type shown above.

7.6.4.3 The TAG_SCCP_IOP Profile

See the OMG specification [SCCP] and Annex A of Part 2 of this International Standard for additional information.

7.6.4.4 The TAG_UIPMC Profile

The **TAG_UIPMC** tag is used by MIOP. See clause 11 of this Part of this International standard and Annex A of Part 2 of this International Standard for additional information.

7.6.5 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.

Specifications of protocols must describe how the components affect the protocol. In addition, a protocol definition must specify, for each TaggedComponent, whether inclusion of the component in profiles supporting the protocol is required (MANDATORY PRESENCE) or not required (OPTIONAL PRESENCE). An ORB claiming to support Full-IOR conformance shall not drop optional components, once they have been added to a profile.

7.6.6 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_FIREWALL_TRANS = 23;
const ComponentId TAG_SCCP_CONTACT_INFO = 24;
const ComponentId TAG_JAVA_CODEBASE = 25;
const ComponentId TAG_TRANSACTION_POLICY = 26;
const ComponentId TAG_MESSAGE_ROUTERS = 30;
const ComponentId TAG_OTS_POLICY = 31;
const ComponentId TAG_INV_POLICY = 32;
const ComponentId TAG_CSI_SEC_MECH_LIST = 33;
const ComponentId TAG_NULL_TAG = 34;
const ComponentId TAG_SECIOP_SEC_TRANS = 35;
const ComponentId TAG_TLS_SEC_TRANS = 36;
const ComponentId TAG_ACTIVITY_POLICY = 37;
const ComponentId TAG_RMI_CUSTOM_MAX_STREAM_FORMAT = 38;
const ComponentId TAG_GROUP = 39;
const ComponentId TAG_GROUP_IOP = 40;
const ComponentId TAG_PASSTHRU_TRANS = 41;
const ComponentId TAG_FIREWALL_PATH = 42;
const ComponentId TAG_IOP_SEC_TRANS = 43;

const ComponentId TAG_INET_SEC_TRANS = 123;
};

```

7.6.6.1 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 IOP 1.1 or greater, it is optionally present.

7.6.6.2 TAG_ALTERNATE_IOP_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 IOP version 1.2.

The **TAG_ALTERNATE_IOP_ADDRESS** component has an associated value of type:

```

struct {
    string HostID,
    unsigned 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.

7.6.6.3 Other Components

The following standard components are specified in various OMG specifications:

- **TAG_CODE_SETS** - See CodeSet Component of IOR Multi-Component Profile in clause 7.10.2.4 in this part of this International Standard.
- **TAG_POLICIES** - See the “CORBA Messaging” clause of ISO/IEC 19500-1.
- **TAG_SEC_NAME** - See the Mechanism Tags sub clause of [CORBASEC].
- **TAG_ASSOCIATION_OPTIONS** - See the Tag Association Options sub clause of [CORBASEC].
- **TAG_SSL_SEC_TRANS** - See the Mechanism Tags sub clause of [CORBASEC].
- **TAG_GENERIC_SEC_MECH** and all other tags with names in the form TAG_*_SEC_MECH - See the “Mechanism Tags” sub clause of [CORBASEC].
- **TAG_FIREWALL_SEC** - See [FIREWALL].
- **TAG_SCCP_CONTACT_INFO** - See [SCCP].
- **TAG_JAVA_CODEBASE** - See [JAV2I].
- **TAG_TRANSACTION_POLICY** - See [TRANS].
- **TAG_MESSAGE_ROUTERS** - See the “CORBA Messaging” clause of ISO/IEC 19500-1.
- **TAG_OTS_POLICY** - See [TRANS].
- **TAG_INV_POLICY** - See [TRANS].
- **TAG_INET_SEC_TRANS** - See [CORBASEC].
- **TAG_CSI_SEC_MECH_LIST**, **TAG_NULL_TAG**, **TAG_SECIOP_SEC_TRANS**, **TAG_TLS_SEC_TRANS** - See the “Secure Interoperability” clause 10 in this part of this International Standard.
- **TAG_ACTIVITY_POLICY** - See [ASMOTS].
- **TAG_RMI_CUSTOM_MAX_STREAM_FORMAT** - See [JAV2I].
- **TAG_GROUP** and **TAG_GROUP_IOP** - See the “Unreliable Multicast Inter-ORB Protocol” clause 11 in this part of this International Standard.
- **TAG_IOP_SEC_TRANS** - See the “Secure Interoperability” clause 10 in this part of this International Standard.

7.6.7 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.

7.6.8 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.

7.6.9 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 destrringify the object reference.
- The ORBs in question might not share a network protocol, or be connected.
- Security constraints may be placed on object reference destrringification.

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 destrringified 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> ::= <i><o><r>“.”
- (3) <hex_Octets> ::= <hex_Octet> {<hex_Octet>}*
- (4) <hex_Octet> ::= <hexDigit> <hexDigit>
- (5) <hexDigit> ::= <digit> | <a> | | <c> | <d> | <e> | <f>
- (6) <digit> ::= “0” | “1” | “2” | “3” | “4” | “5” |
(7) “6” | “7” | “8” | “9”
- (8) <a> ::= “a” | “A”
- (9) ::= “b” | “B”
- (10) <c> ::= “c” | “C”
- (11) <d> ::= “d” | “D”
- (12) <e> ::= “e” | “E”
- (13) <f> ::= “f” | “F”
- (14) <i> ::= “i” | “I”
- (15) <o> ::= “o” | “O”
- (16) <r> ::= “r” | “R”

NOTE: The case for characters in a stringified IOR is not significant.

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 clause 9.4, CDR Transfer Syntax, on page 71 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.

7.6.10 Object URLs

To address the problem of bootstrapping and allow for more convenient exchange of human-readable object references, **ORB::string_to_object** allows URLs in the **corbaloc** and **corbaname** formats to be converted into object references.

If conversion fails, **string_to_object** raises a **BAD_PARAM** exception with one of following standard minor codes, as appropriate:

Minor Code	Description
7	string_to_object conversion failed due to bad scheme name
8	string_to_object conversion failed due to bad address
9	string_to_object conversion failed due to bad schema specific part
10	string_to_object conversion failed due to non-specific reason

7.6.10.1 corbaloc URL

The **corbaloc** URL scheme provides stringified object references that are more easily manipulated by users than **IOR** URLs. Currently, **corbaloc** URLs denote objects that can be contacted by IIOP or **resolve_initial_references**. Other transport protocols can be explicitly specified when they become available. Examples of IIOP and **resolve_initial_references** (**rir:**) based **corbaloc** URLs are:

corbaloc::555xyz.com/Prod/TradingService
corbaloc:iiop:1.1@555xyz.com/Prod/TradingService
corbaloc::555xyz.com,:556xyz.com:80/Dev/NameService
corbaloc:rir:/TradingService
corbaloc:rir:/NameService
corbaloc:iiop:192.168.14.25:555/NameService
corbaloc::[1080::8:800:200C:417A]:88/DefaultEventChannel

A **corbaloc** URL contains one or more:

- protocol identifiers
- protocol specific components such as address and protocol version information

When the **rir** protocol is used, no other protocols are allowed. After the addressing information, a **corbaloc** URL ends with a single object key. The full syntax is:

<corbaloc>	= "corbaloc:"<obj_addr_list>["/"<key_string>]
<obj_addr_list>	= [<obj_addr> ","]* <obj_addr>
<obj_addr>	= <prot_addr> <future_prot_addr>
<prot_addr>	= <rir_prot_addr> <iiop_prot_addr>
<rir_prot_addr>	= <rir_prot_token>":"
<rir_prot_token>	= "rir"
<iiop_prot_addr>	= <iiop_id><iiop_addr>
<iiop_id>	= ":" <iiop_prot_token>":"
<iiop_prot_token>	= "iiop"
<iiop_addr>	= [<version> <host> [":" <port>]]
<host>	= DNS_style_host_name ip_address
<host>	= DNS_style_host_name IPv4_address "[" IPv6_address "]"
<version>	= <major> "." <minor> "@" empty_string
<port>	= number
<major>	= number
<minor>	= number
<future_prot_addr>	= <future_prot_id><future_prot_addr>
<future_prot_id>	= <future_prot_token>":"
<future_prot_token>	= possible examples: "atm" "dce"
<future_prot_addr>	= protocol specific address
<key_string>	= <string> empty_string

Where:

obj_addr_list: comma-separated list of protocol id, version, and address information. This list is used in an implementation-defined manner to address the object. An object may be contacted by any of the addresses and protocols.

NOTE: If the **rir** protocol is used, no other protocols are allowed.

obj_addr: A protocol identifier, version tag, and a protocol specific address. The comma ',' and '/' characters are specifically prohibited in this component of the URL.

rir_prot_addr: **resolve_initial_references** protocol identifier. This protocol does not have a version tag or address. See 7.6.10.2, 'corbaloc:rir URL'.

iiop_prot_addr: **iiop** protocol identifier, version tag, and address containing a DNS-style host name or IP address. See corbaloc:iiop URL on page 35 for the iiop specific definitions.

future_prot_addr: a placeholder for future **corbaloc** protocols.

future_prot_id: token representing a protocol terminated with a ":".

future_prot_token: token representing a protocol. Currently only "iiop" and "rir" are defined.

future_prot_addr: a protocol specific address and possibly protocol version information. An example of this for **iiop** is "1.1@555xyz.com."

key_string: a stringified object key.

The **key_string** corresponds to the octet sequence in the **object_key** member of a GIOP **Request** or **LocateRequest** header as defined in 15.4 of CORBA 2.3. The **key_string** uses the escape conventions described in RFC 2396 to map away from octet values that cannot directly be part of a URL. US-ASCII alphanumeric characters are not escaped. Characters outside this range are escaped, except for the following:

```

";" | "/" | "." | "?" | ":" | "@" | "&" | "=" | "+" | "$" |
"," | "-" | "_" | "!" | "~" | "*" | "'" | "(" | ")"

```

The **key_string** is not NUL-terminated.

7.6.10.2 corbaloc:rir URL

The **corbaloc:rir** URL is defined to allow access to the ORB's configured initial references through a URL. The protocol address syntax is:

```

<rir_prot_addr>      = <rir_prot_token>":"
<rir_prot_token>     = "rir"

```

Where:

rir_prot_addr: **resolve_initial_references** protocol identifier. There is no version or address information when **rir** is used.

rir_prot_token: The token "rir" identifies this protocol.

For a **corbaloc:rir** URL, the **<key_string>** is used as the argument to **resolve_initial_references**. An empty **<key_string>** is interpreted as the default "NameService."

The **rir** protocol cannot be used with any other protocol in a URL.

7.6.10.3 corbaloc:iiop URL

The **corbaloc:iiop** URL is defined for use in TCP/IP- and DNS-centric environments. The full protocol address syntax is:

```

<iiop_prot_addr>     = <iiop_id><iiop_addr>
<iiop_id>            = <iiop_default> | <iiop_prot_token>":"
<iiop_default>       = "."
<iiop_prot_token>    = "iiop"

```

<iiop_addr>	= [<version> <host> [“:” <port>]]
<host>	= DNS_style_host_name IPv4_address “[” IPv6_address “]”
<version>	= <major> “.” <minor> “@” empty_string
<port>	= number
<major>	= number
<minor>	= number

Where:

iiop_prot_addr: iiop protocol identifier, version tag, and address containing a DNS-style host name or IP address.

iiop_id: tokens recognized to indicate an iiop protocol corbaloc.

iiop_default: default token indicating iiop protocol, “:”.

iiop_prot_token: iiop protocol token, “iiop.”

iiop_address: a single address.

host: DNS-style host name or IP address. If not present, the local host is assumed.

version: a major and minor version number, separated by ‘.’ and followed by ‘@’. If the version is absent, 1.0 is assumed.

IPv4_address: numeric IPv4 address (dotted decimal notation).

IPv6_address: numeric IPv6 address (colon separated hexadecimal or mixed hexadecimal/decimal notation as described in RFC 2373).

port: port number the agent is listening on (see below). Default is 2809.

7.6.10.4 corbaloc Server Implementation

The only requirements on an object advertised by a **corbaloc** URL are that there must be a software agent listening on the host and port specified by the URL. This agent must be capable of handling **GIOP Request** and **LocateRequest** messages targeted at the object key specified in the URL.

A normal CORBA server meets these criteria. It is also possible to implement lightweight object location forwarding agents that respond to **GIOP Request** messages with **Reply** messages with a **LOCATION_FORWARD** status, and respond to **GIOP LocateRequest** messages with **LocateReply** messages.

7.6.10.5 corbaname URL

The **corbaname** URL scheme is described in the Naming Service specification. It extends the capabilities of the **corbaloc** scheme to allow URLs to denote entries in a Naming Service. Resolving **corbaname** URLs does not require a Naming Service implementation in the ORB core. Some examples are:

corbaname::555objs.com#a/string/path/to/obj

This URL specifies that at host **555objs.com**, an object of type **NamingContext** (with an object key of **NameService**) can be found, or alternatively, that an agent is running at that location which will return a reference to a **NamingContext**. The (stringified) name **a/string/path/to/obj** is then used as the argument to a **resolve** operation on that **NamingContext**. The URL denotes the object reference that results from that lookup.

corbaname:rir:#a/local/obj

This URL specifies that the stringified name **a/local/obj** is to be resolved relative to the naming context returned by **resolve_initial_references("NameService")**.

7.6.10.6 Future corbaloc URL Protocols

This part of ISO/IEC 19500 only defines use of iiop with corbaloc. New protocols can be added to **corbaloc** as required. Each new protocol must implement the **<future_prot_addr>** component of the URL and define described in **corbaloc** URL on page 33.

A possible example of a future corbaloc URL that incorporates an ATM address is:

corbaloc:iiop:xyz.com,atm:E.164:358.400.1234567/dev/test/objectX

7.6.10.7 Future URL Schemes

Several currently defined non-CORBA URL scheme names are reserved. Implementations may choose to provide these or other URL schemes to support additional ways of denoting objects with URLs.

Table 7.1 lists the required and some optional formats.

Table 7.1

Scheme	Description	Status
IOR:	Standard stringified IOR format	Required
corbaloc:	Simple object reference. rir: must be supported.	Required
corbaname:	CosName URL	Required
file://	Specifies a file containing a URL/IOR	Optional
ftp://	Specifies a file containing a URL/IOR that is accessible via ftp protocol.	Optional
http://	Specifies an HTTP URL that returns an object URL/IOR.	Optional

7.7 Service Context

Emerging specifications for Object Services occasionally require service-specific context information to be passed implicitly with requests and replies. 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;
    typedef CORBA::OctetSeq ContextData;

    struct ServiceContext {
        ServiceId      context_id;
        ContextData    context_data;
    };
    typedef sequence <ServiceContext>ServiceContextList;
};
```

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 Encapsulation on page 79). 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 24 bits of a service context ID contain a 24-bit vendor service context codeset ID (VSCID); the low-order 8 bits contain the rest of the service context ID. A vendor (or group of vendors) who wishes to define a specific set of service context IDs 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 VSCIDs of zero to 15 inclusive (0x000000 to 0x00000f) are 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).

7.7.1 Standard Service Contexts

```
module IOP {                                     // IDL
    const ServiceId      TransactionService = 0;
```