IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 841 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 842 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 843 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 844 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 845 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 846 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 847 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 848 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 849 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 850 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 851 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 852 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 853 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 854 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 855 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 856 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 857 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 858 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 859 of 1442

Contents

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 860 of 1442

Preface

About This Document

Under the terms of the collaboration between OMG and X/Open Co Ltd., this document is a candidate for endorsement by X/Open, initially as a Preliminary Specification and later as a full CAE Specification. The collaboration between OMG and X/Open Co Ltd. ensures joint review and cohesive support for emerging object-based specifications.

X/Open Preliminary Specifications undergo close scrutiny through a review process at X/Open before publication and are inherently stable specifications. Upgrade to full CAE Specification, after a reasonable interval, takes place following further review by X/Open. This further review considers the implementation experience of members and the full implications of conformance and branding.

Object Management Group

The Object Management Group, Inc. (OMG) is an international organization supported by over 800 members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development. The organization's charter includes the establishment of industry guidelines and object management specifications to provide a common framework for application development. Primary goals are the reusability, portability, and interoperability of object-based software in distributed, heterogeneous environments. Conformance to these specifications will make it possible to develop a heterogeneous applications environment across all major hardware platforms and operating systems.

OMG's objectives are to foster the growth of object technology and influence its direction by establishing the Object Management Architecture (OMA). The OMA provides the conceptual infrastructure upon which all OMG specifications are based.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 861 of 1442

X/Open

X/Open is an independent, worldwide, open systems organization supported by most of the world's largest information system suppliers, user organizations and software companies. Its mission is to bring to users greater value from computing, through the practical implementation of open systems. X/Open's strategy for achieving its mission is to combine existing and emerging standards into a comprehensive, integrated systems environment called the Common Applications Environment (CAE).

The components of the CAE are defined in X/Open CAE specifications. These contain, among other things, an evolving portfolio of practical application programming interfaces (APIs), which significantly enhance portability of application programs at the source code level. The APIs also enhance the interoperability of applications by providing definitions of, and references to, protocols and protocol profiles.

The X/Open specifications are also supported by an extensive set of conformance tests and by the X/Open trademark (XPG brand), which is licensed by X/Open and is carried only on products that comply with the CAE specifications.

Intended Audience

The architecture and specifications described in this manual are aimed at software designers and developers who want to produce applications that comply with OMG standards for the Object Request Broker (ORB). The benefit of compliance is, in general, to be able to produce interoperable applications that are based on distributed, interoperating objects. As defined by the Object Management Group (OMG) in the *Object Management Architecture Guide,* 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.

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 *CORBAservices: Common Object Services Specification.*

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 862 of 1442

- **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 *CORBAfacilities: Common Facilities Architecture*.
- **Application Objects,** which are products of a single vendor on 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.

Associated Documents

The CORBA documentation set includes the following books:

- *Object Management Architecture Guide* defines the OMG's technical objectives and terminology and describes the conceptual models upon which OMG standards are based. It also provides information about the policies and procedures of OMG, such as how standards are proposed, evaluated, and accepted.
- *CORBA: Common Object Request Broker Architecture and Specification* contains the architecture and specifications for the Object Request Broker.
- *CORBAservices: Common Object Services Specification* contains specifications for the Object Services.
- *CORBAfacilities: Common Facilities Architecture* contains the architecture for Common Facilities.

OMG collects information for each book in the documentation set by issuing Requests for Information, Requests for Proposals, and Requests for Comment and, with its membership, evaluating the responses. Specifications are adopted as standards only when representatives of the OMG membership accept them as such by vote.

To obtain books in the documentation set, or other OMG publications, refer to the enclosed subscription card or contact the Object Management Group, Inc. at:

OMG Headquarters
250 First Avenue, Suite 201
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
pubs@omg.org
http://www.omg.org

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 863 of 1442

Definition of CORBA Compliance

The minimum required for a CORBA-compliant system is adherence to the specifications in CORBA Core and one mapping. Each additional language mapping is a separate, optional compliance point. Optional means users aren't required to implement these points if they are unnecessary at their site, but if implemented, they must adhere to the *CORBA* specifications to be called CORBA-compliant. For instance, if a vendor supports C++, their ORB must comply with the OMG IDL to C++ binding specified in the *C++ Language Mapping Specification*.

Interoperability and Interworking are separate compliance points. For detailed information about Interworking compliance, refer to "Compliance to COM/CORBA Interworking" on page 17-34.

As described in the *OMA Guide*, the OMG's Core Object Model consists of a core and components. Likewise, the body of *CORBA* specifications is divided into core and component-like specifications. The structure of this manual reflects that division.

The *CORBA* core specifications are categorized as follows:

CORBA Core, as specified in Chapters 1-11

CORBA Interoperability, as specified in Chapters 12-16

CORBA Interworking, as specified in Chapters 17-21

CORBA Quality of Service, as specified in Chapters 22-26

Note – The CORBA Language Mappings have been separated from the CORBA Core and each language mapping is its own separate book. Refer to CORBA Language Mappings at the OMG Formal Document web area for this information.

Structure of This Manual

This manual is divided into the categories of Core, Interoperability, and Interworking. These divisions reflect the compliance points of CORBA. In addition to this preface, CORBA: Common Object Request Broker Architecture and Specification contains the following chapters:

Core

Chapter 1 - The Object Model describes the computation model that underlies the CORBA architecture.

Chapter 2 - CORBA Overview contains the overall structure of the ORB architecture and includes information about CORBA interfaces and implementations.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 864 of 1442

Chapter 3 - OMG IDL Syntax and Semantics details the OMG interface definition language (OMG IDL), which is the language used to describe the interfaces that client objects call and object implementations provide.

Chapter 4 - ORB Interface defines the interface to the ORB functions that do not depend on object adapters: these operations are the same for all ORBs and object implementations.

Chapter 5 - Value Type Semantics describes the semantics of passing an object by value, which is similar to that of standard programming languages.

Chapter 6 - Abstract Interface Semantics explains an IDL abstract interface, which provides the capability to defer the determination of whether an object is passed by reference or by value until runtime.

Chapter 7 - The Dynamic Invocation Interface details the DII, the client's side of the interface that allows dynamic creation and invocation of request to objects.

Chapter 8 -- The Dynamic Skeleton Interface describes the DSI, the server's-side interface that can deliver requests from an ORB to an object implementation that does not have compile-time knowledge of the type of the object it is implementing. DSI is the server's analogue of the client's Dynamic Invocation Interface (DII).

Chapter 9 - Dynamic Management of Any Values details the interface for the Dynamic Any type. This interface allows statically-typed programming languages such as C and Java to create or receive values of type Any without compile-time knowledge that the typer contained in the Any.

Chapter 10 - Interface Repository explains the component of the ORB that manages and provides access to a collection of object definitions.

Chapter 11 - Portable Object Adapter defines a group of IDL interfaces than an implementation uses to access ORB functions.

Interoperability

Chapter 12 - Interoperability Overview describes the interoperability architecture and introduces the subjects pertaining to interoperability: inter-ORB bridges; general and Internet inter-ORB protocols (GIOP and IIOP); and environment-specific, inter-ORB protocols (ESIOPs).

Chapter 13 - ORB Interoperability Architecture introduces the framework of ORB interoperability, including information about domains; approaches to inter-ORB bridges; what it means to be compliant with ORB interoperability; and ORB Services and Requests.

Chapter 14 - Building Inter-ORB Bridges explains how to build bridges for an implementation of interoperating ORBs.

Chapter 15 - General Inter-ORB Protocol describes the general inter-ORB protocol (GIOP) and includes information about the GIOP's goals, syntax, format, transport, and object location. This chapter also includes information about the Internet inter-ORB protocol (IIOP).

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 865 of 1442

Chapter 16 - DCE ESIOP - Environment-Specific Inter-ORB Protocol (ESIOP) details a protocol for the OSF DCE environment. The protocol is called the DCE Environment Inter-ORB Protocol (DCE ESIOP).

Interworking

Chapter 17 - Interworking Architecture describes the architecture for communication between two object management systems: Microsoft's COM (including OLE) and the OMG's CORBA.

Chapter 18 - Mapping: COM and CORBA explains the data type and interface mapping between COM and CORBA. The mappings are described in the context of both Win16 and Win32 COM.

Chapter 19 - Mapping: OLE Automation and CORBA details the two-way mapping between OLE Automation (in ODL) and CORBA (in OMG IDL).

Note: Chapter 19 also includes an appendix describing solutions that vendors might implement to support existing and older OLE Automation controllers and an appendix that provides an example of how the Naming Service could be mapped to an OLE Automation interface according to the Interworking specification.

Chapter 20 - Interoperability with non-CORBA Systems describes the effective access to CORBA servers through DCOM and the reverse.

Chapter 21 - Portable Interceptors defines ORB operations that allow services such as security to be inserted in the invocation path.

Quality of Service (QoS)

Chapter 22 - CORBA Messaging includes three general topics: Quality of Service, Asynchronous Method Invocations (to include Time-Independent or "Persistent" Requests), and the specification of interoperable Routing interfaces to support the transport of requests asynchronously from the handling of their replies.

Chapter 23 - Minimum CORBA describes minimumCORBA, a subset of CORBA designed for systems with limited resources.

Chapter 24 - Real-Time CORBA defines an optional set of extensions to CORBA tailored to equip ORBs to be used as a component of a Real-Time system.

Chapter 25 - Fault Tolerant CORBA describes Fault Tolerant systems, basic fault tolerance mechanisms, replication management, and logging and recovery management.

Chapter 26 - Common Secure Interoperability defines 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 866 of 1442

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Helvetica bold - OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier bold` - Programming language elements.

Helvetica - Exceptions

Terms that appear in *italics* are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Acknowledgements

The following companies submitted and/or supported parts of the specifications that were approved by the Object Management Group to become *CORBA:*

- Adiron, LLC
- Alcatel
- BEA Systems, Inc.
- BNR Europe Ltd.
- Borland International, Inc.
- Compaq Computer Corporation
- Concept Five Technologies
- Cooperative Research Centre for Distributed Systems Technology (DSTC)
- Defense Information Systems Agency
- Digital Equipment Corporation
- Ericsson
- Eternal Systems, Inc.
- Expersoft Corporation
- France Telecom
- FUJITSU LIMITED
- Genesis Development Corporation
- Gensym Corporation
- Hewlett-Packard Company
- HighComm
- Highlander Communications, L.C.
- Humboldt-University
- HyperDesk Corporation
- ICL, Plc.
- Inprise Corporation
- International Business Machines Corporation
- International Computers, Inc.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 867 of 1442

- IONA Technologies, Plc.
- Lockheed Martin Federal Systems, Inc.
- Lucent Technologies, Inc.
- Micro Focus Limited
- MITRE Corporation
- Motorola, Inc.
- NCR Corporation
- NEC Corporation
- Netscape Communications Corporation
- Nortel Networks
- Northern Telecom Corporation
- Novell, Inc.
- Object Design, Inc.
- Objective Interface Systems, Inc.
- Object-Oriented Concepts, Inc.
- OC Systems, Inc.
- Open Group - Open Software Foundation
- Oracle Corporation
- PeerLogic, Inc.
- Persistence Software, Inc.
- Promia, Inc.
- Siemens Nixdorf Informationssysteme AG
- SPAWAR Systems Center
- Sun Microsystems, Inc.
- SunSoft, Inc.
- Sybase, Inc.
- Telefónica Investigación y Desarrollo S.A. Unipersonal
- TIBCO, Inc.
- Tivoli Systems, Inc.
- Tri-Pacific Software, Inc.
- University of California, Santa Barbara
- University of Rhode Island
- Visual Edge Software, Ltd.
- Washington University

In addition to the preceding contributors, the OMG would like to acknowledge Mark Linton at Silicon Graphics and Doug Lea at the State University of New York at Oswego for their work on the C++ mapping.

References

IDL Type Extensions RFP, March 1995. OMG TC Document 95-1-35.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 868 of 1442

The Common Object Request Broker: Architecture and Specification, Revision 2.2, February 1998.

CORBAservices: Common Object Services Specification, Revised Edition, OMG TC Document 95-3-31.

COBOL Language Mapping RFP, December 1995. OMG TC document 95-12-10.

COBOL 85 ANSI X3.23-1985 / ISO 1989-1985.

IEEE Standard for Binary Floating-Point Arithmetic, ANIS/IEEE Std 754-1985.

XDR: External Data Representation Standard, RFC1832, R. Srinivasan, Sun Microsystems, August 1995.

OSF Character and Code Set Registry, OSF DCE SIG RFC 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.

X/Open System Interface Definitions, Issue 4 Version 2, 1995.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 869 of 1442

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 870 of 1442

Interoperability Overview 12

Contents

This chapter contains the following sections.

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.

12.1 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 871 of 1442

12.1.1 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.

12.1.2 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 872 of 1442

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.

12.1.3 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.

12.1.4 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 12-1 on page 12-4.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 873 of 1442

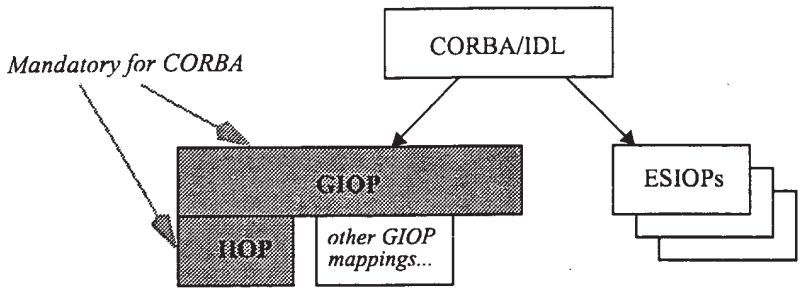

Figure 12-1 Inter-ORB Protocol Relationships.

12.1.5 Environment-Specific Inter-ORB Protocols (ESIOPs)

This specification 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" interoperation 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.

12.2 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 Section 13.2, "ORBs and ORB Services," on page 13-3). 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 chapter.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 874 of 1442

- 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 chapter of this book. The Relationship Service is described in the Relationship Service specification; refer to the *CosObjectIdentity Module* section of that specification.

12.3 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 section provides some examples.

12.3.1 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.

12.3.2 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.

12.3.3 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.

12.3.4 Interoperability Compliance

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

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 875 of 1442

- In the CORBA Core part of this specification, 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 chapter of this book.
- An Internet Inter-ORB Protocol (IIOP) (explained in the Building Inter-ORB Brdiges chapter) 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 12-2 on page 12-7 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 876 of 1442

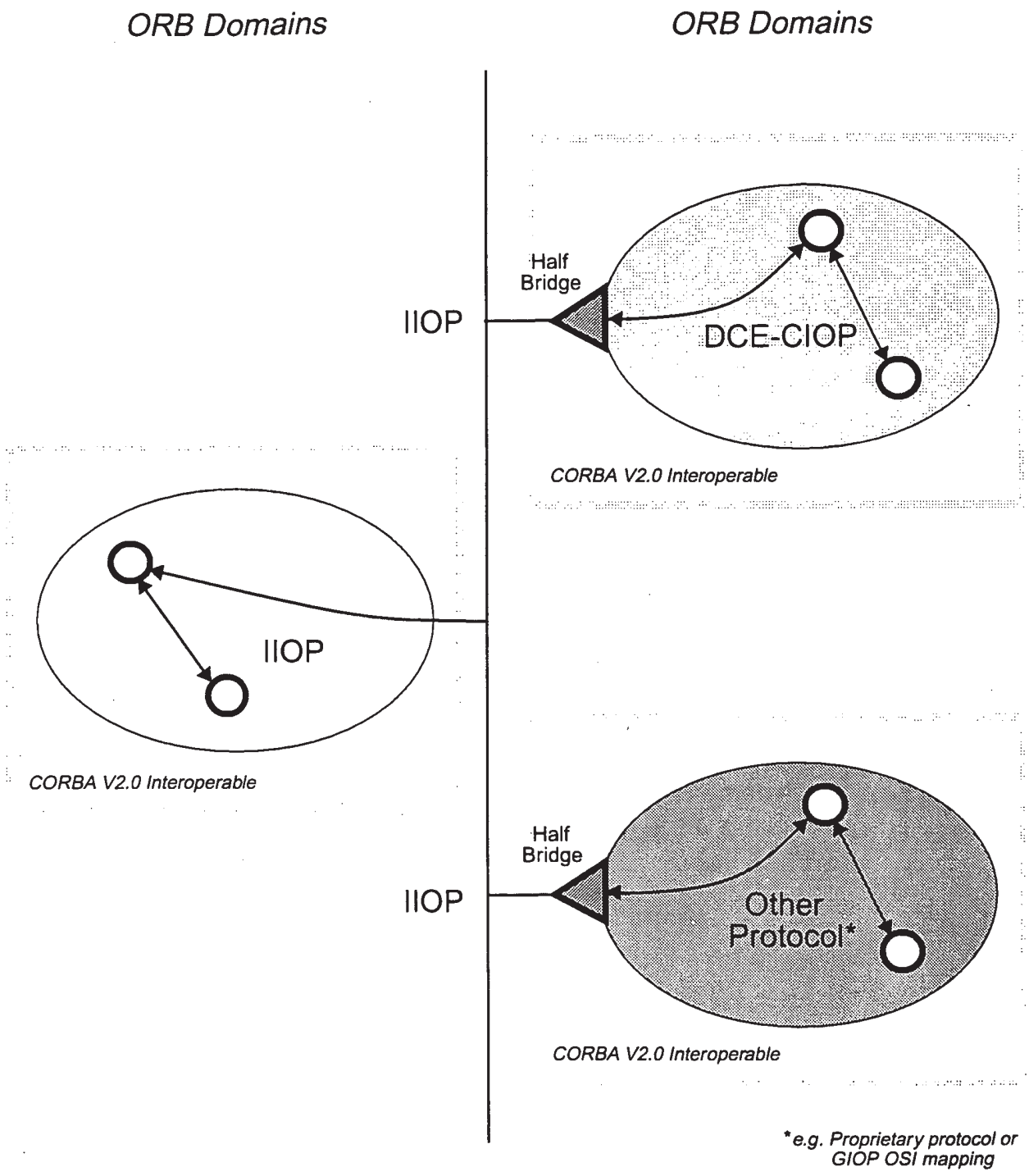

Figure 12-2 Examples of CORBA Interoperability Compliance

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 877 of 1442

12.4 Motivating Factors

This section explains the factors that motivated the creation of interoperability specifications.

12.4.1 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.

12.4.2 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 878 of 1442

12.4.3 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 implementer 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.

12.5 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 specification 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 879 of 1442

- 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.

12.5.1 Non-Goals

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

- Support for security
- Support for future ORB Services

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 880 of 1442

ORB Interoperability Architecture 13

Contents

This chapter contains the following sections.

13.1 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 881 of 1442

- 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.

13.1.1 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.

13.1.2 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 882 of 1442

13.2 ORBs and ORB Services

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

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

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

13.2.1 The Nature of ORB Services

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

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

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

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

13.2.2 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 883 of 1442

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.

13.2.3 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

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 884 of 1442

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.

13.3 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.

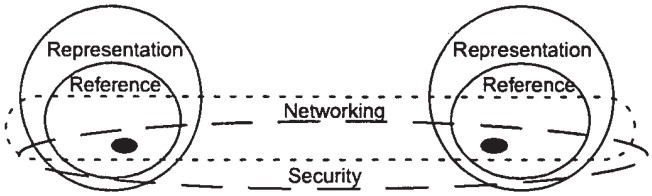

Figure 13-1 Different Kinds of Domains can Coexist.

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

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

13.3.1 Definition of a Domain

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

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 885 of 1442

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.

13.3.2 Mapping Between Domains: Bridging

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

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

A special case of this requirement is that the object models of the two domains need to be compatible. This specification assumes that both domains are strictly compliant with the CORBA Object Model and the *CORBA* specifications. This includes the use of OMG IDL 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 886 of 1442

13.4 Interoperability Between ORBs

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

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

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

13.4.1 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).

13.4.2 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).

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 887 of 1442

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.

13.4.3 Interoperability Approaches

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

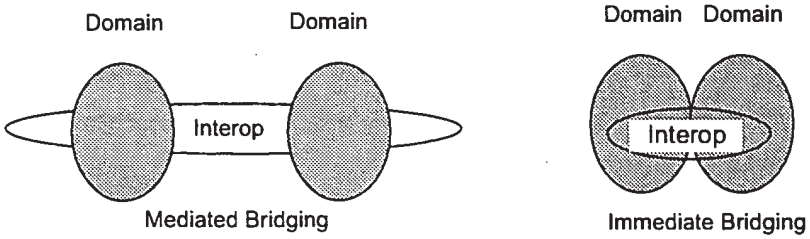

Figure 13-2 Two bridging techniques, different uses of an intermediate form agreed on between the two domains.

13.4.3.1 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 888 of 1442

- 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.

13.4.3.2 Immediate Bridging

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

Observations on immediate bridging are as follows:

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

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

13.4.3.3 Location of Inter-Domain Functionality

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

For example, if a single ORB encompasses two security domains, the inter-domain bridge could be implemented wholly within the ORB and thus be invisible as far as ORB interoperability is concerned. A similar situation arises when a bridge between two ORBs or domains is implemented wholly within a process or system which has access to both. In such cases, the engineering issues of inter-domain bridging are

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 889 of 1442

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.

13.4.3.4 Bridging Level

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

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

13.4.4 Policy-Mediated Bridging

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

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

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

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

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

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 890 of 1442

13.4.5 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.

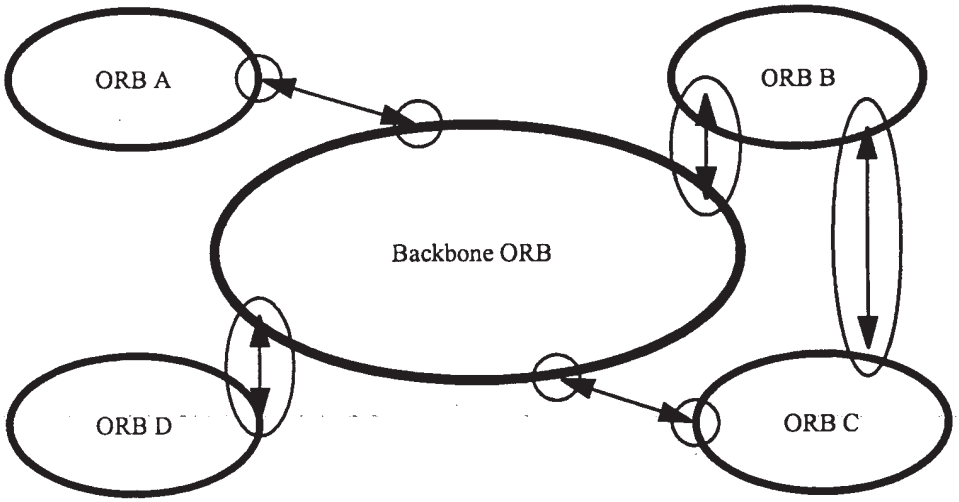

Figure 13-3 An ORB chosen as a backbone will connect other ORBs through bridges, both full-bridges and half-bridges.

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.

13.5 Object Addressing

The Object Model (see Chapter 1, Requests) 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 891 of 1442

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

This section 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.

13.5.1 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.

13.5.2 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 892 of 1442

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*.

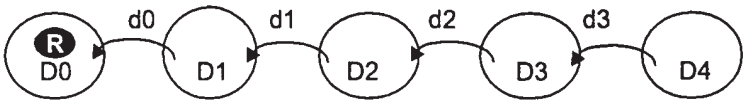

Figure 13-4 Reference encapsulation adds domain information during bridging.

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)*.

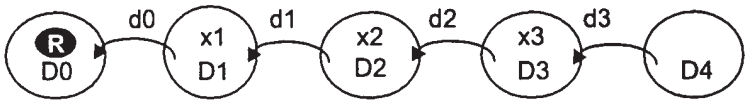

Figure 13-5 Domain Reference Translation *substitutes domain references* during bridging.

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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 893 of 1442

- 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.

13.6 An Information Model for Object References

This section provides a simple, powerful information model for the information found in an object reference. That model is intended to be used directly by developers of bridging technology, and is used in that role by the IIOP, described in the *General Inter-ORB Protocol* chapter, *Object References* section.

13.6.1 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 Section 13.2.3, "Selection of ORB Services" on page 13-4, 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.

13.6.2 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 894 of 1442

```
module IOP {                                        // IDL

    // Standard Protocol Profile tag values

    typedef unsigned long           ProfileId;

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

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

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

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

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

13.6.3 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)

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 895 of 1442

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 Section 15.7.3, "IIOP IOR Profile Components" on page 15-54) and for use in "multiple component profiles." Profile tags in the range **0x80000000** through **0xffffffff** are reserved for future use, and are not currently available for assignment.

Null object references are indicated by an empty set of profiles, and by a "Null" type ID (a string which contains only a single terminating character). 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 896 of 1442

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.

13.6.4 Standard IOR Profiles

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

    typedef sequence <TaggedComponent> MultipleComponentProfile;
};
```

13.6.4.1 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 Section 15.7.2, "IIOP IOR Profiles" on page 15-51, contains a CDR encapsulation of a structure containing addressing and object identification information used by IIOP. Version 1.1 of the **TAG_INTERNET_IOP** profile also includes a **sequence<TaggedComponent>** that can contain additional information supporting optional IIOP features, ORB services such as security, and future protocol extensions.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 897 of 1442

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

The **profile_data** for the **TAG_INTERNET_IOP** profile is a CDR encapsulation of the **IIOP::ProfileBody_1_1** type, described in Section 15.7.2, "IIOP IOR Profiles" on page 15-51.

13.6.4.2 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 Section 13.6.5, "IOR Components" on page 13-18.

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

13.6.4.3 The TAG_SCCP_IOP Profile

See the CORBA/IN Interworking specification (dtc/2000-02-02).

13.6.5 IOR Components

TaggedComponents contained in **TAG_INTERNET_IOP** and **TAG_MULTIPLE_COMPONENTS** profiles are identified by unique numeric tags using a namespace distinct form 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 898 of 1442

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.

13.6.6 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_IIOP_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_INET_SEC_TRANS = 123;
};
```

The following additional components that can be used by other protocols are specified in the DCE ESIOP chapter of this document and *CORBAServices*, Security Service, in the Security Service for DCE ESIOP section:

```
const ComponentId   TAG_COMPLETE_OBJECT_KEY = 5;
const ComponentId   TAG_ENDPOINT_ID_POSITION = 6;
const ComponentId   TAG_LOCATION_POLICY = 12;
const ComponentId   TAG_DCE_STRING_BINDING = 100;
const ComponentId   TAG_DCE_BINDING_NAME = 101;
const ComponentId   TAG_DCE_NO_PIPES = 102;
```


IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 899 of 1442

```
const ComponentId    TAG_DCE_SEC_MECH = 103; // Security Service
```

13.6.6.1 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 IIOP 1.1 or greater, it is optionally present.

13.6.6.2 TAG_ALTERNATE_IIOP_ADDRESS Component

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

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

```
struct {
    string HostID,
    unsigned short Port
};
```

encoded as a CDR encapsulation.

Zero or more instances of the **TAG_ALTERNATE_IIOP_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_IIOP_ADDRESS** components are present in a **TAG_INTERNET_IOP** Profile, no order of use is prescribed by Version 1.2 of IIOP.

13.6.6.3 Other Components

The following standard components are specified in various OMG specifications:

- **TAG_CODE_SETS** - See Section 13.10.2.4, "CodeSet Component of IOR Multi-Component Profile" on page 13-42.
- **TAG_POLICIES** - See CORBA Messaging - chapter 22.
- **TAG_SEC_NAME** - See the Security Service specification, Mechanism Tags section.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 900 of 1442

- **TAG_ASSOCIATION_OPTIONS** - See the Security Service specification, Tag Association Options section.
- **TAG_SSL_SEC_TRANS** - See the Security Service specification, Mechanism Tags section.
- **TAG_GENERIC_SEC_MECH** and all other tags with names in the form **TAG_*_SEC_MECH** - See the Security Service specification, Mechanism Tags section.
- **TAG_FIREWALL_SEC** - See the Firewall specification (orbos/98-05-04).
- **TAG_SCCP_CONTACT_INFO** - See the CORBA/IN Interworking specification (telecom/98-10-03).
- **TAG_JAVA_CODEBASE** - See the Java to IDL Language Mapping specification (formal/99-07-59), Codebase Transmission section.
- **TAG_TRANSACTION_POLICY** - See the Object Transaction Service specification (formal/00-06-28).
- **TAG_MESSAGE_ROUTERS** - See CORBA Messaging (chapter 22).
- **TAG_OTS_POLICY** - See the Object Transaction Service specification (formal/00-06-28).
- **TAG_INV_POLICY** - See the Object Transaction Service specification (formal/00-06-28).
- **TAG_INET_SEC_TRANS** - See the Security Service specification (formal/00-06-25).
- **TAG_COMPLETE_OBJECT_KEY** (See Section 16.5.4, "Complete Object Key Component" on page 16-19).
- **TAG_ENDPOINT_ID_POSITION** (See Section 16.5.5, "Endpoint ID Position Component" on page 16-20).
- **TAG_LOCATION_POLICY** (See Section 16.5.6, "Location Policy Component" on page 16-20).
- **TAG_DCE_STRING_BINDING** (See Section 16.5.1, "DCE-CIOP String Binding Component" on page 16-17).
- **TAG_DCE_BINDING_NAME** (See Section 16.5.2, "DCE-CIOP Binding Name Component" on page 16-18).
- **TAG_DCE_NO_PIPES** (See Section 16.5.3, "DCE-CIOP No Pipes Component" on page 16-19).

13.6.7 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 901 of 1442

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*.

13.6.8 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.

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

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 902 of 1442

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

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

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

```
(1)    <oref>     ::= <prefix> <hex_Octets>
(2)    <prefix>   ::= <i><o><r>":"
(3)    <hex_Octets> ::= <hex_Octet> {<hex_Octet>}*
(4)    <hex_Octet> ::= <hexDigit> <hexDigit>
(5)    <hexDigit> ::= <digit> | <a> | <b> | <c> | <d> | <e> | <f>
(6)    <digit>    ::= "0" | "1" | "2" | "3" | "4" | "5" |
                     |  "6" | "7" | "8" | "9"
(7)    <a>        ::= "a" | "A"
(8)    <b>        ::= "b" | "B"
(9)    <c>        ::= "c" | "C"
(10)   <d>        ::= "d" | "D"
(11)   <e>        ::= "e" | "E"
(12)   <f>        ::= "f" | "F"
(13)   <i>        :: = "i" | "I"
(14)   <o>        :: = "o" | "O"
(15)   <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 Section 15.3, "CDR Transfer Syntax" on page 15-4 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.

13.6.10 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:

- 7 - string_to_object conversion failed due to bad scheme name

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 903 of 1442

- 8 - string_to_object conversion failed due to bad address
- 9 - string_to_object conversion failed due to bad bad schema specific part
- 10 - string_to_object conversion failed due to non specific reason

13.6.10.1 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
```

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
<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
```


IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 904 of 1442

<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 Section 13.6.10.2, "`corbaloc:rir` URL".

iiop_prot_addr: iiop protocol identifier, version tag, and address containing a DNS-style host name or IP address. See Section 13.6.10.3, "`corbaloc:iiop` URL"" 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 section 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.

13.6.10.2 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"**

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 905 of 1442

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 can not be used with any other protocol in a URL.

13.6.10.3 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 | ip_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.

ip_address: numeric IP address (dotted decimal notation)

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

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 906 of 1442

13.6.10.4 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.

13.6.10.5 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**, a 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")**.

13.6.10.6 Future corbaloc URL Protocols

This specification 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 a described in Section 13.6.10.1, "`corbaloc` URL" on page 13-24."

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

13.6.10.7 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.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 907 of 1442

Table 13-1 lists the required and some optional formats.

Table 13-1 URL formats

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

13.7 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:

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 908 of 1442

```
module IOP {        // IDL

    typedef unsigned long              ServiceId;

    struct ServiceContext {
        ServiceId           context_id;
        sequence <octet>    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 Section 15.3.3, "Encapsulation" on page 15-14). The **context_id** member contains the service ID value identifying the service and data format. Context data is encapsulated in octet sequences to permit ORBs to handle context data without unmarshaling, and to handle unknown context data types.

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

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

The high-order 20 bits of service-context ID contain a 20-bit vendor service context codeset ID (VSCID); the low-order 12 bits contain the rest of the service context ID. A vendor (or group of vendors) who wish to define a specific set of 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 VSCID of zero is reserved for use for OMG-defined standard service context IDs (i.e., service context IDs in the range 0-4095 are reserved as OMG standard service contexts).

13.7.1 Standard Service Contexts

```
module IOP {        // IDL
    const ServiceId         TransactionService = 0;
    const ServiceId         CodeSets = 1;
    const ServiceId         ChainBypassCheck = 2;
    const ServiceId         ChainBypassInfo = 3;
    const ServiceId         LogicalThreadId = 4;
    const ServiceId         BI_DIR_IIOP = 5;
```


IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 909 of 1442

```
    const ServiceId     SendingContextRunTime = 6;
    const ServiceId     INVOCATION_POLICIES = 7;
    const ServiceId     FORWARDED_IDENTITY = 8;
    const ServiceId     UnknownExceptionInfo = 9;
    const ServiceId     RTCorbaPriority = 10;
    const ServiceId     RTCorbaPriorityRange = 11;
    const ServiceId     ExceptionDetailMessage = 14;
};
```

The standard ServiceIds currently defined are:

- **TransactionService** identifies a CDR encapsulation of the **CosTransactions::PropogationContext** defined in the Object Transaction Service specification (formal/00-06-28).
- **CodeSets** identifies a CDR encapsulation of the **CONV_FRAME::CodeSetContext** defined in Section 13.10.2.5, "GIOP Code Set Service Context" on page 13-43.
- DCOM-CORBA Interworking uses three service contexts as defined in "DCOM-CORBA Interworking" in the "Interoperability with non-CORBA Systems"chapter. They are:
  - **ChainBypassCheck**, which carries a CDR encapsulation of the **struct CosBridging::ChainBypassCheck**. This is carried only in a **Request** message as described in Section 20.9.1, "CORBA Chain Bypass" on page 20-19.
  - **ChainBypassInfo**, which carries a CDR encapsulation of the **struct CosBridging::ChainBypassInfo**. This is carried only in a **Reply** message as described in Section 20.9.1, "CORBA Chain Bypass" on page 20-19.
  - **LogicalThreadId**, which carries a CDR encapsulation of the **struct CosBridging::LogicalThreadId** as described in Section 20.10, "Thread Identification" on page 20-21.
- **BI_DIR_IIOP** identifies a CDR encapsulation of the **IIOP::BiDirIIOPServiceContext** defined in Section 15.8, "Bi-Directional GIOP" on page 15-55.
- **SendingContextRunTime** identifies a CDR encapsulation of the IOR of the **SendingContext::RunTime** object (see Section 5.6, "Access to the Sending Context Run Time" on page 5-18).
- For information on **INVOCATION_POLICIES** refer to CORBA Messaging (chapter 22).
- For information on **FORWARDED_IDENTITY** refer to the Firewall specification (orbos/98-05-04).
- **UnknownExceptionInfo** identifies a CDR encapsulation of a marshaled instance of a **java.lang.throwable** or one of its subclasses as described in Java to IDL Language Mapping, "Mapping of UnknownExceptionInfo Service Context," section.
- For information on **RTCorbaPriority** refer to the Real-time CORBA (chapter 24).

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 910 of 1442

- For information on **RTCorbaPriorityRange** refer to the Real-time CORBA (chapter 24).
- **ExceptionDetailMessage** identifies a CDR encapsulation of a wstring, encoded using GIOP 1.2 with a TCS-W of UTF-16. This service context may be sent on Reply messages with a reply_status of SYSTEM_EXCEPTION or USER_EXCEPTION. The usage of this service context is defined by language mappings.

13.7.2 Service Context Processing Rules

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

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

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

13.8 Coder/Decoder Interfaces

The formats of IOR components and service context data used by ORB services are often defined as CDR encapsulations encoding instances of IDL defined data types. The **Codec** provides a mechanism to transfer these components between their IDL data types and their CDR encapsulation representations.

A **Codec** is obtained from the **CodecFactory**. The **CodecFactory** is obtained through a call to **ORB::resolve_initial_references ("CodecFactory")**.

13.8.1 Codec Interface

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

        CORBA::OctetSeq encode (in any data)
```


IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 911 of 1442

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

13.8.1.1 Exceptions

InvalidTypeForEncoding

This exception is raised by **encode** or **encode_value** when the type is invalid for the encoding. For example, this exception is raised if the encoding is **ENCODING_CDR_ENCAPS** version 1.0 and a type that does not exist in that version, such as wstring, is passed to the operation.

FormatMismatch

This exception is raised by **decode** or **decode_value** when the data in the octet sequence cannot be decoded into an any.

TypeMismatch

This exception is raised by **decode_value** when the given **TypeCode** does not match the given octet sequence.

13.8.1.2 Operations

encode

Convert the given **any** into an octet sequence based on the encoding format effective for this **Codec**.

This operation may raise **InvalidTypeForEncoding**.

Parameter

data	The data, in the form of an **any**, to be encoded into an octet sequence.

Return Value

An octet sequence containing the encoded **any**. This octet sequence contains both the **TypeCode** and the data of the type.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 912 of 1442

decode

Decode the given octet sequence into an **any** based on the encoding format effective for this **Codec**.

This operation raises **FormatMismatch** if the octet sequence cannot be decoded into an **any**.

Parameter

data	The data, in the form of an octet sequence, to be decoded into an **any**.

Return Value

An **any** containing the data from the decoded octet sequence.

encode_value

Convert the given any into an octet sequence based on the encoding format effective for this **Codec**. Only the data from the any is encoded, not the **TypeCode**.

This operation may raise **InvalidTypeForEncoding**.

Parameter

data	The data, in the form of an **any**, to be encoded into an octet sequence.

Return Value

An octet sequence containing the data from the encoded **any**.

decode_value

Decode the given octet sequence into an any based on the given **TypeCode** and the encoding format effective for this Codec.

This operation raises **FormatMismatch** if the octet sequence cannot be decoded into an any.

Parameter

data	The data, in the form of an octet sequence, to be decoded into an **any**.
tc	The **TypeCode** to be used to decode the data.

Return Value

An **any** containing the data from the decoded octet sequence.

13.8.2 Codec Factory

```
module IOP {
    typedef short EncodingFormat;
    const EncodingFormat ENCODING_CDR_ENCAPS = 0;
```


IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 913 of 1442

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

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

13.8.2.1 Encoding Structure

The **Encoding** structure defines the encoding format of a **Codec**. It details the encoding format, such as CDR Encapsulation encoding, and the major and minor versions of that format.

The encodings which shall be supported are:

- **ENCODING_CDR_ENCAPS**, version 1.0;
- **ENCODING_CDR_ENCAPS**, version 1.1;
- **ENCODING_CDR_ENCAPS**, version 1.2;
- **ENCODING_CDR_ENCAPS** for all future versions of GIOP as they arise.

Vendors are free to support additional encodings.

13.8.2.2 CodecFactory Interface

create_codec

Create a **Codec** of the given encoding.

This operation raises **UnknownEncoding** if this factory cannot create a **Codec** of the given encoding.

Parameter

enc The **Encoding** for which to create a **Codec**.

Return Value

A **Codec** obtained with the given encoding.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 914 of 1442

13.9 Feature Support and GIOP Versions

The association of service contexts with GIOP versions, (along with some other supported features tied to GIOP minor version), is shown in Table 13-2..

Table 13-2 Feature Support Tied to Minor GIOP Version Number

Feature	Version 1.0	Version 1.1	Version 1.2
TransactionService Service Context	yes	yes	yes
CodeSets Service Context		yes	yes
DCOM Bridging Service Contexts: ChainBypassCheck ChainBypassInfo LogicalThreadId			yes
Object by Value Service Context: SendingContextRunTime			yes
Bi-Directional IIOP Service Context: BI_DIR_IIOP			yes
Asynch Messaging Service Context INVOCATION_POLICIES			optional[$]
Firewall Service Context FORWARDED_IDENTITY			optional[$]
Java Language Throwable Service Context: UnknownExceptionInfo			yes
Realtime CORBA Service Contexts RTCorbaPriority RTCorbaPriorityRange			optional (Realtime CORBA only)
ExceptionDetailMessage Service Context			optional
IOR components in IIOP profile		yes	yes
TAG_ORB_TYPE		yes	yes
TAG_CODE_SETS		yes	yes
TAG_ALTERNATE_IIOP_ADDRESS			yes
TAG_ASSOCIATION_OPTION		yes	yes
TAG_SEC_NAME		yes	yes
TAG_SSL_SEC_TRANS		yes	yes
TAG_GENERIC_SEC_MECH		yes	yes
TAG_*_SEC_MECH		yes	yes
TAG_JAVA_CODEBASE			yes

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 915 of 1442

Table 13-2 Feature Support Tied to Minor GIOP Version Number *(Continued)*

Feature	Version 1.0	Version 1.1	Version 1.2
TAG_FIREWALL_TRANS			optional[$]
TAG_SCCP_CONTACT_INFO			optional[$]
TAG_TRANSACTION_POLICY			optional[$]
TAG_MESSAGE_ROUTERS			optional[$]
TAG_OTS_POLICY			optional[$]
TAG_INV_POLICY			optional[$]
TAG_INET_SEC_TRANS			optional[$]
Extended IDL data types		yes	yes
Bi-Directional GIOP Features			yes
Value types and Abstract Interfaces			yes

Note – [$] All features that have been added after CORBA 2.3 have been marked as optional in GIOP 1.2. These features cannot be compulsory in GIOP 1.2 since there is no way to incorporate them in deployed implementations of 1.2. However, in order to have the additional features of CORBA 2.4 work properly these optional features must be supported by the GIOP 1.2 implementation connecting CORBA 2.4 ORBs.

13.10 Code Set Conversion

13.10.1 Character Processing Terminology

This section introduces a few terms and explains a few concepts to help understand the character processing portions of this document.

13.10.1.1 Character Set

A finite set of different characters used for the representation, organization, or control of data. In this specification, the term "character set" is used without any relationship to code representation or associated encoding. Examples of character sets are the English alphabet, Kanji or sets of ideographic characters, corporate character sets (commonly used in Japan), and the characters needed to write certain European languages.

13.10.1.2 Coded Character Set, or Code Set

A set of unambiguous rules that establishes a character set and the one-to-one relationship between each character of the set and its bit representation or numeric value. In this specification, the term "code set" is used as an abbreviation for the term

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 916 of 1442

"coded character set." Examples include ASCII, ISO 8859-1, JIS X0208 (which includes Roman characters, Japanese hiragana, Greek characters, Japanese kanji, etc.) and Unicode.

13.10.1.3 Code Set Classifications

Some language environments distinguish between byte-oriented and "wide characters." The byte-oriented characters are encoded in one or more 8-bit bytes. A typical single-byte encoding is ASCII as used for western European languages like English. A typical multi-byte encoding which uses from one to three 8-bit bytes for each character is eucJP (Extended UNIX Code - Japan, packed format) as used for Japanese workstations.

Wide characters are a fixed 16 or 32 bits long, and are used for languages like Chinese, Japanese, etc., where the number of combinations offered by 8 bits is insufficient and a fixed-width encoding is needed. A typical example is Unicode (a "universal" character set defined by the The Unicode Consortium, which uses an encoding scheme identical to ISO 10646 UCS-2, or 2-byte Universal Character Set encoding). An extended encoding scheme for Unicode characters is UTF-16 (UCS Transformation Format, 16-bit representations).

The C language has data types **`char`** for byte-oriented characters and **`wchar_t`** for wide characters. The language definition for C states that the sizes for these characters are implementation-dependent. Some environments do not distinguish between byte-oriented and wide characters (e.g., Ada and Smalltalk). Here again, the size of a character is implementation-dependent. The following table illustrates code set classifications as used in this document.

Table 13-3 Code Set Classification

Orientation	Code Element Encoding	Code Set Examples	C Data Type
byte-oriented	single-byte	ASCII, ISO 8859-1 (Latin-1), EBCDIC, ...	char
	multi-byte	UTF-8, eucJP, Shift-JIS, JIS, Big5, ...	char[]
non-byte-oriented	fixed-length	ISO 10646 UCS-2 (Unicode), ISO 10646 UCS-4, UTF-16, ...	wchar_t

13.10.1.4 Narrow and Wide Characters

Some language environments distinguish between "narrow" and "wide" characters. Typically the narrow characters are considered to be 8-bit long and are used for western European languages like English, while the wide characters are 16-bit or 32-bit long and are used for languages like Chinese, Japanese, etc., where the number of combinations offered by 8 bits are insufficient. However, as noted above there are common encoding schemes in which Asian characters are encoded using multi-byte code sets and it is incorrect to assume that Asian characters are always encoded as "wide" characters.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 917 of 1442

Within this specification, the general terms "narrow character" and "wide character" are only used in discussing OMG IDL.

13.10.1.5 Char Data and Wchar Data

The phrase "**char** data" in this specification refers to data whose IDL types have been specified as **char** or **string**. Likewise "**wchar** data" refers to data whose IDL types have been specified as **wchar** or **wstring**.

13.10.1.6 Byte-Oriented Code Set

An encoding of characters where the numeric code corresponding to a character code element can occupy one or more bytes. A byte as used in this specification is synonymous with octet, which occupies 8 bits.

13.10.1.7 Multi-Byte Character Strings

A character string represented in a byte-oriented encoding where each character can occupy one or more bytes is called a multi-byte character string. Typically, wide characters are converted to this form from a (fixed-width) process code set before transmitting the characters outside the process (see below about process code sets). Care must be taken to correctly process the component bytes of a character's multi-byte representation.

13.10.1.8 Non-Byte-Oriented Code Set

An encoding of characters where the numeric code corresponding to a character code element can occupy fixed 16 or 32 bits.

13.10.1.9 Char and Wchar Transmission Code Set (TCS-C and TCS-W)

These two terms refer to code sets that are used for transmission between ORBs after negotiation is completed. As the names imply, the first one is used for **char** data and the second one for **wchar** data. Each TCS can be byte-oriented or non-byte oriented.

13.10.1.10 Process Code Set and File Code Set

Processes generally represent international characters in an internal fixed-width format which allows for efficient representation and manipulation. This internal format is called a "process code set." The process code set is irrelevant outside the process, and hence to the interoperation between CORBA clients and servers through their respective ORBs.

When a process needs to write international character information out to a file, or communicate with another process (possibly over a network), it typically uses a different encoding called a "file code set." In this specification, unless otherwise

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 918 of 1442

indicated, all references to a program's code set refer to the file code set, not the process code set. Even when a client and server are located physically on the same machine, it is possible for them to use different file code sets.

13.10.1.11 Native Code Set

A native code set is the code set which a client or a server uses to communicate with its ORB. There might be separate native code sets for **char** and **wchar** data.

13.10.1.12 Transmission Code Set

A transmission code set is the commonly agreed upon encoding used for character data transfer between a client's ORB and a server's ORB. There are two transmission code sets established per session between a client and its server, one for **char** data (TCS-C) and the other for **wchar** data (TCS-W). Figure 13-6 illustrates these relationships:

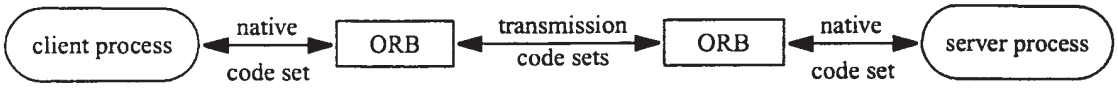

Figure 13-6 Transmission Code Sets

The intent is for TCS-C to be byte-oriented and TCS-W to be non-byte-oriented. However, this specification does allow both types of characters to be transmitted using the same transmission code set. That is, the selection of a transmission code set is orthogonal to the wideness or narrowness of the characters, although a given code set may be better suited for either narrow or wide characters.

13.10.1.13 Conversion Code Set (CCS)

With respect to a particular ORB's native code set, the set of other or target code sets for which an ORB can convert all code points or character encodings between the native code set and that target code set. For each code set in this CCS, the ORB maintains appropriate translation or conversion procedures and advertises the ability to use that code set for transmitted data in addition to the native code set.

13.10.2 Code Set Conversion Framework

13.10.2.1 Requirements

The file code set that an application uses is often determined by the platform on which it runs. In Japan, for example, Japanese EUC is used on Unix systems, while Shift-JIS is used on PCs. Code set conversion is therefore required to enable interoperability across these platforms. This proposal defines a framework for the automatic conversion of code sets in such situations. The requirements of this framework are:

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 919 of 1442

1. Backward compatibility. In previous CORBA specifications, IDL type **char** was limited to ISO 8859-1. The conversion framework should be compatible with existing clients and servers that use ISO 8859-1 as the code set for **char**.

2. Automatic code set conversion. To facilitate development of CORBA clients and servers, the ORB should perform any necessary code set conversions automatically and efficiently. The IDL type **octet** can be used if necessary to prevent conversions.

3. Locale support. An internationalized application determines the code set in use by examining the LOCALE string (usually found in the LANG environment variable), which may be changed dynamically at run time by the user. Example LOCALE strings are fr_FR.ISO8859-1 (French, used in France with the ISO 8859-1 code set) and ja_JP.ujis (Japanese, used in Japan with the EUC code set and X11R5 conventions for LOCALE). The conversion framework should allow applications to use the LOCALE mechanism to indicate supported code sets, and thus select the correct code set from the registry.

4. CMIR and SMIR support. The conversion framework should be flexible enough to allow conversion to be performed either on the client or server side. For example, if a client is running in a memory-constrained environment, then it is desirable for code set converters to reside in the server and for a Server Makes It Right (SMIR) conversion method to be used. On the other hand, if many servers are executed on one server machine, then converters should be placed in each client to reduce the load on the server machine. In this case, the conversion method used is Client Makes It Right (CMIR).

13.10.2.2 Overview of the Conversion Framework

Both the client and server indicate a native code set indirectly by specifying a locale. The exact method for doing this is language-specific, such as the XPG4 C/C++ function `setlocale`. The client and server use their native code set to communicate with their ORB. (Note that these native code sets are in general different from process code sets and hence conversions may be required at the client and server ends.)

The conversion framework is illustrated in Figure 13-7. The server-side ORB stores a server's code set information in a component of the IOR multiple-component profile structure (see Section 13.6.2, "Interoperable Object References: IORs" on page 13-14)[1]. The code sets actually used for transmission are carried in the service context field of an IOP (Inter-ORB Protocol) request header (see Section 13.7, "Service Context" on page 13-28 and Section 13.10.2.5, "GIOP Code Set Service Context" on page 13-43). Recall that there are two code sets (TCS-C and TCS-W) negotiated for each session.

1. Version 1.1 of the IIOP profile body can also be used to specify the server's code set information, as this version introduces an extra field that is a sequence of tagged components.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 920 of 1442

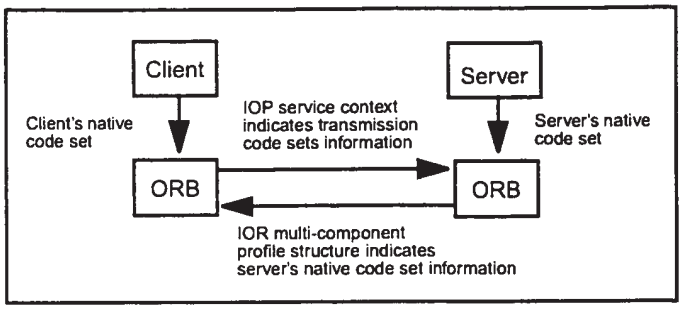

Figure 13-7 Code Set Conversion Framework Overview

If the native code sets used by a client and server are the same, then no conversion is performed. If the native code sets are different and the client-side ORB has an appropriate converter, then the CMIR conversion method is used. In this case, the server's native code set is used as the transmission code set. If the native code sets are different and the client-side ORB does not have an appropriate converter but the server-side ORB does have one, then the SMIR conversion method is used. In this case, the client's native code set is used as the transmission code set.

The conversion framework allows clients and servers to specify a native **char** code set and a native **wchar** code set, which determine the local encodings of IDL types **char** and **wchar**, respectively. The conversion process outlined above is executed independently for the **char** code set and the **wchar** code set. In other words, the algorithm that is used to select a transmission code set is run twice, once for **char** data and once for **wchar** data.

The rationale for selecting two transmission code sets rather than one (which is typically inferred from the locale of a process) is to allow efficient data transmission without any conversions when the client and server have identical representations for **char** and/or **wchar** data. For example, when a Windows NT client talks to a Windows NT server and they both use Unicode for wide character data, it becomes possible to transmit wide character data from one to the other without any conversions. Of course, this becomes possible only for those wide character representations that are well-defined, not for any proprietary ones. If a single transmission code set was mandated, it might require unnecessary conversions. (For example, choosing Unicode as the transmission code set would force conversion of all byte-oriented character data to Unicode.)

13.10.2.3 ORB Databases and Code Set Converters

The conversion framework requires an ORB to be able to determine the native code set for a locale and to convert between code sets as necessary. While the details of exactly how these tasks are accomplished are implementation-dependent, the following databases and code set converters might be used:

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 921 of 1442

- Locale database. This database defines a native code set for a process. This code set could be byte-oriented or non-byte-oriented and could be changed programmatically while the process is running. However, for a given session between a client and a server, it is fixed once the code set information is negotiated at the session's setup time.
- Environment variables or configuration files. Since the locale database can only indicate one code set while the ORB needs to know two code sets, one for **char** data and one for **wchar** data, an implementation can use environment variables or configuration files to contain this information on native code sets.
- Converter database. This database defines, for each code set, the code sets to which it can be converted. From this database, a set of "conversion code sets" (CCS) can be determined for a client and server. For example, if a server's native code set is eucJP, and if the server-side ORB has eucJP-to-JIS and eucJP-to-SJIS bilateral converters, then the server's conversion code sets are JIS and SJIS.
- Code set converters. The ORB has converters which are registered in the converter database.

13.10.2.4 CodeSet Component of IOR Multi-Component Profile

The code set component of the IOR multi-component profile structure contains:

- server's native **char** code set and conversion code sets, and
- server's native **wchar** code set and conversion code sets.

Both **char** and **wchar** conversion code sets are listed in order of preference. The code set component is identified by the following tag:

```
const IOP::ComponentID TAG_CODE_SETS = 1;
```

This tag has been assigned by OMG (See Section 13.6.6, "Standard IOR Components" on page 13-19.). The following IDL structure defines the representation of code set information within the component:

```
module CONV_FRAME {                          // IDL
    typedef unsigned long CodeSetId;
    struct CodeSetComponent {
        CodeSetId               native_code_set;
        sequence<CodeSetId>     conversion_code_sets;
    };
    struct CodeSetComponentInfo {

        CodeSetComponent        ForCharData;
        CodeSetComponent        ForWcharData;
    };
};
```

Code sets are identified by a 32-bit integer id from the OSF Character and Code Set Registry (See Section 13.10.5.1, "Character and Code Set Registry" on page 13-49 for further information). Data within the code set component is represented as a structure

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 922 of 1442

of type **CodeSetComponentInfo**, and is encoded as a CDR encapsulation. In other words, the **char** code set information comes first, then the **wchar** information, represented as structures of type **CodeSetComponent**.

A null value should be used in the **native_code_set** field if the server desires to indicate no native code set (possibly with the identification of suitable conversion code sets).

If the code set component is not present in a multi-component profile structure, then the default **char** code set is ISO 8859-1 for backward compatibility. However, there is no default **wchar** code set. If a server supports interfaces that use wide character data but does not specify the **wchar** code sets that it supports, client-side ORBs will raise exception INV_OBJREF, with standard minor code 1.

If a client application invokes an operation which results in an attempt by the client ORB to marshall wchar or wstring data for an in parameter (or to unmarshal wchar or wstring data for an in/out parameter, out parameter or the return value), and the associated Object Reference does not include a codeset component, then the client ORB shall raise the INV_OBJREF standard system exception with standard minor code 2 as a response to the operation invocation.

13.10.2.5 GIOP Code Set Service Context

The code set GIOP service context contains:

- **char** transmission code set, and
- **wchar** transmission code set

in the form of a code set service. This service is identified by:

```
const IOP::ServiceID  CodeSets = 1;
```

The following IDL structure defines the representation of code set service information:

```
module CONV_FRAME {                          // IDL
    typedef unsigned long CodeSetId;
    struct CodeSetContext {
        CodeSetId           char_data;
        CodeSetId           wchar_data;
    };
};
```

Code sets are identified by a 32-bit integer id from the OSF Character and Code Set Registry (See Section 13.10.5.1, "Character and Code Set Registry" on page 13-49 for further information).

Note – A server's **char** and **wchar** Code set components are usually different, but under some special circumstances they can be the same. That is, one could use the same code set for both **char** data and **wchar** data. Likewise the **CodesetIds** in the service context don't have to be different.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 923 of 1442

13.10.2.6 Code Set Negotiation

The client-side ORB determines a server's native and conversion code sets from the code set component in an IOR multi-component profile structure, and it determines a client's native and conversion code sets from the locale setting (and/or environment variables/configuration files) and the converters that are available on the client. From this information, the client-side ORB chooses **char** and **wchar** transmission code sets (TCS-C and TCS-W). For both requests and replies, the **char** TCS-C determines the encoding of **char** and **string** data, and the **wchar** TCS-W determines the encoding of **wchar** and **wstring** data.

Code set negotiation is not performed on a per-request basis, but only when a client initially connects to a server. All text data communicated on a connection are encoded as defined by the TCSs selected when the connection is established.

Figure 13-8 illustrates, there are two channels for character data flowing between the client and the server. The first, TCS-C, is used for **char** data and the second, TCS-W, is used for **wchar** data. Also note that two native code sets, one for each type of data, could be used by the client and server to talk to their respective ORBs (as noted earlier, the selection of the particular native code set used at any particular point is done via `setlocale` or some other implementation-dependent method).

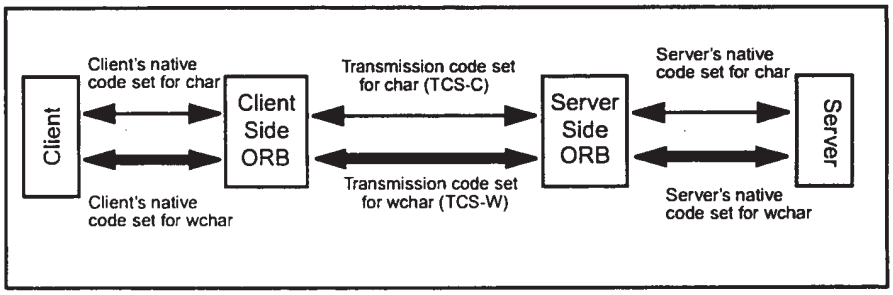

Figure 13-8 Transmission Code Set Use

Let us look at an example. Assume that the code set information for a client and server is as shown in the table below. (Note that this example concerns only **char** code sets and is applicable only for data described as **char**s in the IDL.)

	Client	**Server**
Native code set:	SJIS	eucJP
Conversion code sets:	eucJP, JIS	SJIS, JIS

The client-side ORB first compares the native code sets of the client and server. If they are identical, then the transmission and native code sets are the same and no conversion is required. In this example, they are different, so code set conversion is necessary. Next, the client-side ORB checks to see if the server's native code set, eucJP, is one of the conversion code sets supported by the client. It is, so eucJP is selected as the

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 924 of 1442

transmission code set, with the client (i.e., its ORB) performing conversion to and from its native code set, SJIS, to eucJP. Note that the client may first have to convert all its data described as **char**s (and possibly **wchar_t**s) from process codes to SJIS first.

Now let us look at the general algorithm for determining a transmission code set and where conversions are performed. First, we introduce the following abbreviations:

- CNCS - Client Native Code Set;
- CCCS - Client Conversion Code Sets;
- SNCS - Server Native Code Set;
- SCCS - Server Conversion Code Sets; and
- TCS - Transmission Code Set.

The algorithm is as follows:

```
if (CNCS==SNCS)
        TCS = CNCS;                 // no conversion required
else {
        if (elementOf(SNCS,CCCS))
            TCS = SNCS;   // client converts to server's native code set
        else if (elementOf(CNCS,SCCS))
            TCS = CNCS;   // server converts from client's native code set
        else if (intersection(CCCS,SCCS) != emptySet) {
            TCS = oneOf(intersection(CCCS,SCCS));
            // client chooses TCS, from intersection(CCCS,SCCS), that is
            // most preferable to server;
            // client converts from CNCS to TCS and server
            // from TCS to SNCS
        else if (compatible(CNCS,SNCS))
            TCS = fallbackCS;       // fallbacks are UTF-8 (for char data) and
                                    // UTF-16 (for wchar data)
        else
            raise CODESET_INCOMPATIBLE exception;
}
```

The algorithm first checks to see if the client and server native code sets are the same. If they are, then the native code set is used for transmission and no conversion is required. If the native code sets are not the same, then the conversion code sets are examined to see if

1. the client can convert from its native code set to the server's native code set,
2. the server can convert from the client's native code set to its native code set, or
3. transmission through an intermediate conversion code set is possible.

If the third option is selected and there is more than one possible intermediate conversion code set (i.e., the intersection of CCCS and SCCS contains more than one code set), then the one most preferable to the server is selected.[2]

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 925 of 1442

If none of these conversions is possible, then the fallback code set (UTF-8 for **char** data and UTF-16 for **wchar** data— see below) is used. However, before selecting the fallback code set, a compatibility test is performed. This test looks at the character sets encoded by the client and server native code sets. If they are different (e.g., Korean and French), then meaningful communication between the client and server is not possible and a CODESET_INCOMPATIBLE exception is raised. This test is similar to the DCE compatibility test and is intended to catch those cases where conversion from the client native code set to the fallback, and the fallback to the server native code set would result in massive data loss. (See Section 13.10.5, "Relevant OSFM Registry Interfaces" on page 13-49 for the relevant OSF registry interfaces that could be used for determining compatibility.)

A DATA_CONVERSION exception is raised when a client or server attempts to transmit a character that does not map into the negotiated transmission code set. For example, not all characters in Taiwan Chinese map into Unicode. When an attempt is made to transmit one of these characters via Unicode, an ORB is required to raise a DATA_CONVERSION exception, with standard minor code 1.

In summary, the fallback code set is UTF-8 for **char** data (identified in the Registry as 0x05010001, "X/Open UTF-8; UCS Transformation Format 8 (UTF-8)"), and UTF-16 for **wchar** data (identified in the Registry as 0x00010109, "ISO/IEC 10646-1:1993; UTF-16, UCS Transformation Format 16-bit form"). As mentioned above the fallback code set is meaningful only when the client and server character sets are compatible, and the fallback code set is distinguished from a default code set used for backward compatibility.

If a server's native **char** code set is not specified in the IOR multi-component profile, then it is considered to be ISO 8859-1 for backward compatibility. However, a server that supports interfaces that use wide character data is required to specify its native **wchar** code set; if one is not specified, then the client-side ORB raises exception INV_OBJREF, with standard minor code set to 1.

Similarly, if no **char** transmission code set is specified in the code set service context, then the **char** transmission code set is considered to be ISO 8859-1 for backward compatibility. If a client transmits wide character data and does not specify its **wchar** transmission code set in the service context, then the server-side ORB raises exception BAD_PARAM, with standard minor code set to 23.

To guarantee "out-of-the-box" interoperability, clients and servers must be able to convert between their native **char** code set and UTF-8 and their native **wchar** code set (if specified) and Unicode. Note that this does not require that all server native code sets be mappable to Unicode, but only those that are exported as native in the IOR. The server may have other native code sets that aren't mappable to Unicode and those can

---

2. Recall that server conversion code sets are listed in order of preference.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 926 of 1442

be exported as SCCSs (but not SNCSs). This is done to guarantee out-of-the-box interoperability and to reduce the number of code set converters that a CORBA-compliant ORB must provide.

ORB implementations are strongly encouraged to use widely-used code sets for each regional market. For example, in the Japanese marketplace, all ORB implementations should support Japanese EUC, JIS and Shift JIS to be compatible with existing business practices.

13.10.3 Mapping to Generic Character Environments

Certain language environments do not distinguish between byte-oriented and wide characters. In such environments both **char** and **wchar** are mapped to the same "generic" character representation of the language. **String** and **wstring** are likewise mapped to generic strings in such environments. Examples of language environments that provide generic character support are Smalltalk and Ada.

Even while using languages that do distinguish between wide and byte-oriented characters (e.g., C and C++), it is possible to mimic some generic behavior by the use of suitable macros and support libraries. For example, developers of Windows NT and Windows 95 applications can write portable code between NT (which uses Unicode strings) and Windows 95 (which uses byte-oriented character strings) by using a set of macros for declaring and manipulating characters and character strings. Appendix A in this chapter shows how to map wide and byte-oriented characters to these generic macros.

Another way to achieve generic manipulation of characters and strings is by treating them as abstract data types (ADTs). For example, if strings were treated as abstract data types and the programmers are required to create, destroy, and manipulate strings only through the operations in the ADT interface, then it becomes possible to write code that is representation-independent. This approach has an advantage over the macro-based approach in that it provides portability between byte-oriented and wide character environments even without recompilation (at runtime the string function calls are bound to the appropriate byte-oriented/wide library). Another way of looking at it is that the macro-based genericity gives compile-time flexibility, while ADT-based genericity gives runtime flexibility.

Yet another way to achieve generic manipulation of character data is through the ANSI C++ Strings library defined as a template that can be parameterized by `char`, `wchar_t`, or other integer types.

Given that there can be several ways of treating characters and character strings in a generic way, this standard cannot, and therefore does not, specify the mapping of **char**, **wchar**, **string**, and **wstring** to all of them. It only specifies the following normative requirements which are applicable to generic character environments:

- **wchar** must be mapped to the generic character type in a generic character environment.
- **wstring** must be mapped to a string of such generic characters in a generic character environment.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 927 of 1442

- The language binding files (i.e., stubs) generated for these generic environments must ensure that the generic type representation is converted to the appropriate code sets (i.e., CNCS on the client side and SNCS on the server side) before character data is given to the ORB runtime for transmission.

13.10.3.1 Describing Generic Interfaces

To describe generic interfaces in IDL we recommend using **wchar** and **wstring**. These can be mapped to generic character types in environments where they do exist and to wide characters where they do not. Either way interoperation between generic and non-generic character type environments is achieved because of the code set conversion framework.

13.10.3.2 Interoperation

Let us consider an example to see how a generic environment can interoperate with a non-generic environment. Let us say there is an IDL interface with both **char** and **wchar** parameters on the operations, and let us say the client of the interface is in a generic environment while the server is in a non-generic environment (for example the client is written in Smalltalk and the server is written in C++).

Assume that the server's (byte-oriented) native **char** code set (SNCS) is eucJP and the client's native **char** code set (CNCS) is SJIS. Further assume that the code set negotiation led to the decision to use eucJP as the **char** TCS-C and Unicode as the **wchar** TCS-W.

As per the above normative requirements for mapping to a generic environment, the client's Smalltalk stubs are responsible for converting all **char** data (however they are represented inside Smalltalk) to SJIS and all **wchar** data to the client's **wchar** code set before passing the data to the client-side ORB. Note that this conversion could be an identity mapping if the internal representation of narrow and wide characters is the same as that of the native code set(s). The client-side ORB now converts all **char** data from SJIS to eucJP and all **wchar** data from the client's **wchar** code set to Unicode, and then transmits to the server side.

The server side ORB and stubs convert the eucJP data and Unicode data into C++'s internal representation for **char**s and **wchar**s as dictated by the IDL operation signatures. Notice that when the data arrives at the server side it does not look any different from data arriving from a non-generic environment (e.g., that is just like the server itself). In other words, the mappings to generic character environments do not affect the code set conversion framework.

13.10.4 Example of Generic Environment Mapping

This section shows how **char**, **wchar**, **string**, and **wchar** can be mapped to the generic C/C++ macros of the Windows environment. This is merely to illustrate one possibility. This section is not normative and is applicable only in generic environments. See Section 13.10.3, "Mapping to Generic Character Environments" on page 13-47.

IPR2016-00726 -ACTIVISION, EA, TAKE-TWO, 2K, ROCKSTAR,
Ex. 1102, p. 928 of 1442