Building Inter-ORB Bridges 12

This chapter provides an implementation-oriented conceptual framework for the construction of bridges to provide interoperability between ORBs. It focuses on the layered *request level bridges* that the CORBA Core specifications facilitate, although ORBs may always be internally modified to support bridges.

Key feature of the specifications for inter-ORB bridges are as follows:

- Enables requests from one ORB to be translated to requests on another
- Provides support for managing tables keyed by object references

The OMG IDL specification for interoperable object references, which are important to inter-ORB bridging, is shown in “Interoperable Object References: IORs” on page 11-14.

Contents

This chapter contains the following sections.

Section Title	Page
“In-Line and Request-Level Bridging”	12-2
“Proxy Creation and Management”	12-5
“Interface-specific Bridges and Generic Bridges”	12-6
“Building Generic Request-Level Bridges”	12-6
“Bridging Non-Referencing Domains”	12-7
“Bootstrapping Bridges”	12-7

12.1 In-Line and Request-Level Bridging

Bridging of an invocation between a client in one domain and a server object in another domain can be mediated through a standardized mechanism, or done immediately using nonstandard ones.

The question of how this bridging is constructed is broadly independent of whether the bridging uses a standardized mechanism. There are two possible options for where the bridge components are located:

- Code inside the ORB may perform the necessary translation or mappings; this is termed *in-line bridging*.
- Application style code outside the ORB can perform the translation or mappings; this is termed *request level bridging*.

Request level bridges which mediate through a common protocol (using networking, shared memory, or some other IPC provided by the host operating system) between distinct execution environments will involve components, one in each ORB, known as "half bridges."

When that mediation is purely internal to one execution environment, using a shared programming environment's binary interfaces to CORBA- and OMG-IDL-defined data types, this is known as a "full bridge"[1]. From outside the execution environment this will appear identical to some kinds of in-line bridging, since only that environment knows the construction techniques used. However, full bridges more easily support portable policy mediation components, because of their use of only standard CORBA programming interfaces.

Network protocols may be used immediately "in-line," or to mediate between request-level half bridges. The General Inter-ORB Protocol can be used in either manner. In addition, this specification provides for Environment Specific Inter-ORB Protocols (ESIOP), allowing for alternative mediation mechanisms.

Note that mediated, request level half-bridges can be built by anyone who as access to an ORB, without needing information about the internal construction of that ORB. Immediate-mode request level half-bridges (i.e., ones using nonstandard mediation mechanisms) can similarly be built without needing information about ORB internals. Only in-line bridges (using either standard or nonstandard mediation mechanisms) need potentially proprietary information about ORB internals.

---

1. Special initialization supporting object referencing domains (e.g. two protocols) to be exposed to application programmers to support construction of this style bridge.

12.1.1 In-line Bridging

In-line bridging is in general the most direct method of bridging between ORBs. It is structurally similar to the engineering commonly used to bridge between systems within a single ORB (e.g., mediating using some common inter-process communications scheme, such as a network protocol). This means that implementing in-line bridges involves as fundamental a set of changes to an ORB as adding a new inter-process communications scheme. (Some ORBs may be designed to facilitate such modifications, though.)

In this approach, the required bridging functionality can be provided by a combination of software components at various levels:

- As additional or alternative services provided by the underlying ORBs
- As additional or alternative stub and skeleton code.

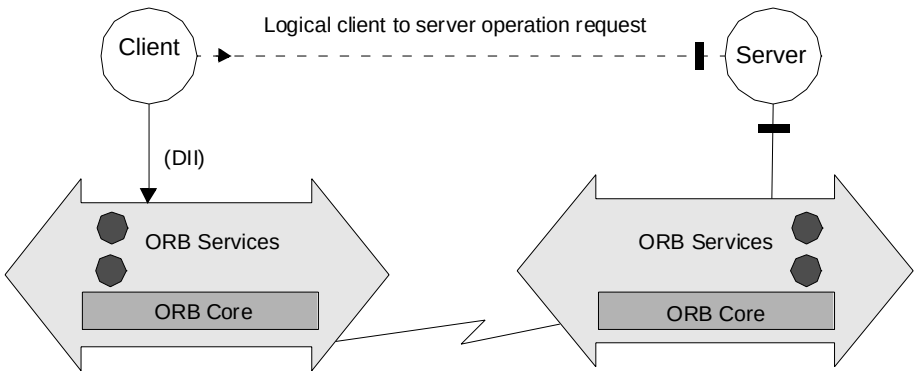

Figure 12-1 In-Line bridges are built using *ORB internal APIs*.

12.1.2 Request-level Bridging

The general principle of request-level bridging is as follows:

1. The original request is passed to a proxy object in the client ORB.
2. The proxy object translates the request contents (including the target object reference) to a form that will be understood by the server ORB.
3. The proxy invokes the required operation on the apparent server object.
4. Any operation result is passed back to the client via a complementary route.

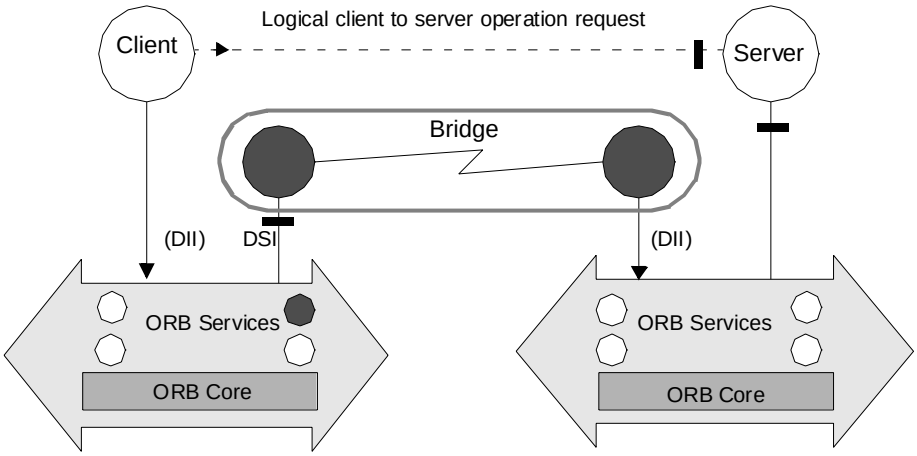

Figure 12-2 Request-Level bridges are built using *public ORB APIs.*

The request translation involves performing object reference mapping for all object references involved in the request (the target, explicit parameters, and perhaps implicit ones such as transaction context). As elaborated later, this translation may also involve mappings for other domains: the security domain of **CORBA::Principal** parameters, type identifiers, and so on.

It is a language mapping requirement of the CORBA Core specification that all dynamic typing APIs (e.g., **Any, NamedValue**) support such manipulation of parameters even when the bridge was not created with compile-time knowledge of the data types involved.

12.1.3 Collocated ORBs

In the case of immediate bridging (i.e. not via a standardized, external protocol) the means of communication between the client-side bridge component and that on the server-side is an entirely private matter. One possible engineering technique optimizes this communication by coalescing the two components into the same system or even the same address space. In the latter case, accommodations must be made by both ORBs to allow them to share the same execution environment.

Similar observations apply to request level bridges, which in the case of collocated ORBs use a common binary interface to all OMG IDL-defined data as their mediating data format.

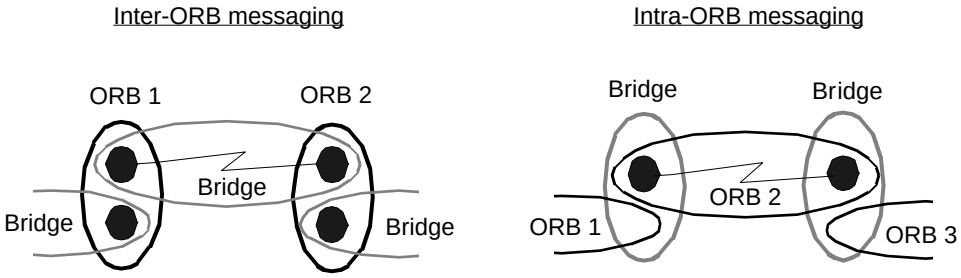

Figure 12-3 When the two ORBs are collocated in a bridge execution environment, network communications will be purely intra-ORB. If the ORBs are not collocated, such communications must go between ORBs.

An advantage of using bridges spanning collocated ORBs is that all external messaging can be arranged to be intra-ORB, using whatever message passing mechanisms each ORB uses to achieve distribution within a single ORB, multiple machine system. That is, for bridges between networked ORBs such a bridge would add only a single “hop,” a cost analogous to normal routing.

12.2 Proxy Creation and Management

Bridges need to support arbitrary numbers of proxy objects, because of the (bidirectional) object reference mappings required. The key schemes for creating and managing proxies are *reference translation* and *reference encapsulation*, as discussed in “Handling of Referencing Between Domains” on page 11-12.

- Reference translation approaches are possible with CORBA V2.0 Core APIs. Proxies themselves can be created as normal objects using the Basic Object Adapter (BOA) and the Dynamic Skeleton Interface (DSI).
- Reference Encapsulation is not supported by the BOA, since it would call for knowledge of more than one ORB. Some ORBs could provide other object adapters which support such encapsulation.

Note that from the perspective of clients, they only ever deal with local objects; clients do not need to distinguish between proxies and other objects. Accordingly, all CORBA operations supported by the local ORB are also supported through a bridge. The ORB used by the client might, however, be able to recognize that encapsulation is in use, depending on how the ORB is implemented.

Also, note that the **CORBA::InterfaceDef** used when creating proxies (e.g, the one passed to **CORBA::BOA::create**) could be either a proxy to one in the target ORB, or could be an equivalent local one. When the domains being bridged include a type domain, then the **InterfaceDef** objects cannot be proxies since type descriptions will not have the same information. When bridging CORBA compliant ORBs, type domains by definition do not need to be bridged.

12.3 Interface-specific Bridges and Generic Bridges

Request-level bridges may be:

- *Interface-specific*: they support predetermined IDL interfaces only, and are built using IDL-compiler generated stub and skeleton interfaces.
- *Generic*: capable of bridging requests to server objects of arbitrary IDL interfaces, using the interface repository and other dynamic invocation support (DII and DSI).

Interface-specific bridges may be more efficient in some cases (a generic bridge could conceivably create the same stubs and skeletons using the interface repository), but the requirement for prior compilation means that this approach offers less flexibility than use of generic bridges.

12.4 Building Generic Request-Level Bridges

The CORBA Core specifications define the following interfaces. These interfaces are of particular significance when building a generic request-level bridge:

- ***Dynamic Invocation Interface (DII)*** lets the bridge make arbitrary invocations on object references whose types may not have been known when the bridge was developed or deployed.
- ***Dynamic Skeleton Interface (DSI)*** lets the bridge handle invocations on proxy object references which it implements, even when their types may not have been known when the bridge was developed or deployed.
- ***Interface Repositories*** are consulted by the bridge to acquire the information used to drive DII and DSI, such as the type codes for operation parameters, return values, and exceptions.
- ***Object Adapters*** (such as the Basic Object Adapter) are used to create proxy object references both when bootstrapping the bridge and when mapping object references which are dynamically passed from one ORB to the other.
- ***CORBA Object References*** support operations to fully describe their interfaces and to create tables mapping object references to their proxies (and vice versa).

Interface repositories accessed on either side of a half bridge need not have the same information, though of course the information associated with any given repository ID (e.g, an interface type ID, exception ID) or operation ID must be the same.

Using these interfaces and an interface to some common transport mechanism such as TCP, portable request-level half bridges connected to an ORB can:

- Use DSI to translate all CORBA invocations on proxy objects to the form used by some mediating protocol such as IIOP (see the General Inter-ORB Protocol chapter).
- Translate requests made using such a mediating protocol into DII requests on objects in the ORB.

As noted in “In-Line and Request-Level Bridging” on page 12-2, translating requests and responses (including exceptional responses) involves mapping object references (and other explicit and implicit parameter data) from the form used by the ORB to the form used by the mediating protocol, and vice versa. Explicit parameters, which are defined by an operation’s OMG-IDL definition, are presented through DII or DSI and are listed in the Interface Repository entry for any particular operation.

Operations on object references such as **hash()** and **is_equivalent()** may be used to maintain tables that support such mappings. When such a mapping does not exist, an object adapter is used to create a ORB-specific proxy object references, and bridge-internal interfaces are used to create the analogous data structure for the mediating protocol.

12.5 Bridging Non-Referencing Domains

In the simplest form of request-level bridging, the bridge operates only on IDL-defined data, and bridges only object reference domains. In this case, a proxy object in the client ORB acts as a representative of the target object and is, in almost any practical sense, indistinguishable from the target server object - indeed, even the client ORB will not be aware of the distinction.

However, as alluded to above, there may be multiple domains that need simultaneous bridging. The transformation and encapsulation schemes described above may not apply in the same way to Principal or type identifiers. Request level bridges may need to translate such identifiers, in addition to object references, as they are passed as explicit operation parameters.

Moreover, there is an emerging class of “implicit context” information that ORBs may need to convey with any particular request, such as transaction and security context information. Such parameters are not defined as part of an operation’s OMG-IDL signature, hence are “implicit” in the invocation context. Bridging the domains of such implicit parameters could involve additional kinds of work, needing to mediate more policies, than bridging the object reference, Principal, and type domains directly addressed by CORBA.

CORBA does not yet have a generic way (including support for both static and dynamic invocations) to expose such implicit context information.

12.6 Bootstrapping Bridges

A particularly useful policy for setting up bridges is to create a pair of proxies for two Naming Service naming contexts (one in each ORB) and then install those proxies as naming contexts in the other ORB’s naming service. (The Naming Service is described in *CORBAservices*.) This will allow clients in either ORB to transparently perform naming context lookup operations on the other ORB, retrieving (proxy) object references for other objects in that ORB. In this way, users can access facilities that have been selectively exported from another ORB, through a naming context, with no administrative action beyond exporting those initial contexts. (See “Obtaining Initial Object References” on page 4-10 for additional information).

This same approach may be taken with other discovery services, such as a trading service or any kind of object that could provide object references as operation results (and in “out” parameters). While bridges can be established which only pass a predefined set of object references, this kind of minimal connectivity policy is not always desirable.