1 Heterogeneous Distributed Computing

Since interoperability must span the enterprise and even go beyond it to customers’ and suppliers’ systems, the interfaces must be independent of platform, operating system, programming language, even network protocol; anything less will create “islands of interoperability” insulated from each other by arbitrary technological barriers that keep us from realizing the benefits we’re entitled from our investment in computing.

2 Object interoperability

In CORBA, the services that an object provides are expressed in a contract that serves as the interface between and the rest of our system. It serves two distinct purposes:

- it informs potential clients of the services the objects provides, and tells them how to construct a message to invoke the service; and

- it lets the communication infrastructure know the format all messages the objects will receive and send, allowing the infrastructure to translate data formats where necessary to provide transparent connection between sender and receiver.

Each object also needs a unique handle that a client can pass to the infrastructure to route a message to it.

The interfaces are expressed in OMG Interface Definition Language (OMG IDL) making them accessible to objects written in virtually any programming language.

The cross-platform communications architecture is the Common Object Request Broker Architecture (CORBA).

3 The OMA

Application-level Integration: CORBA connects only objects, not applications. OMG provides it in the Object Management Architecture or OMA. It is divided into two major components: lower-level CORBA services and
intermediate-level CORBA facilities. While CORBA services provides services for objects, CORBA facilities provides services for applications.

4 What is a CORBA OBJECT?

- Encapsulation: interface and implementation: Encapsulation enables-
  - plug-and-play software
  - location transparency: Clients send the invocation to their local ORB, not to the local object itself;
  - the ORB routes the message to its destination.
  - Inheritance—to create a new object template more easily by adapting an existing one than by creating it from scratch.
  - Polymorphism

5 OMG IDL

This is used to define CORBA’s object interface. For every major P/L, an OMG standard language mapping specifies how IDL types, method invocation, and other constructs convert into language function calls. The IDL compiler uses the mapping specs to generate a set of function calls from IDL operations.

5.1 Connecting to the ORB

The connections to the ORB, one for each interface in the IDL file, are generated automatically by the IDL compiler. Since ORB’s and proprietary ORBs and IDL compilers come in matched sets. You must use the IDL compiler with its companion ORB; the skeleton from vendor A will not match an ORB from vendor B. However, you can simply recompile the IDL code with the new ORB to get a new skeleton.
5.2 Summary

a. Start with an IDL interface definition

b. Select a P/L and using the OMG standard language mapping, determined the the corresponding calls in that language.

c. Using the IDL compiler that came with our ORB, we compiled IDL to generate a skeleton that joins our ORB. The skeleton contains function calls.

d. The IDL compiler also generates a client stub. It contains function declarations.

5.3 Foundations for Interoperability

Our objective is to use a web of ORB-to-ORB communications pathways to enable interoperability among all of the CORBA objects on a network. The two problems we have to overcome are:

- location — where do we find the object’s implementation: SOLUTION: object reference

- translation — how does our invocation get translated to a foreign ORB’s data format on the way over and the response get translated on the way back? SOLUTION: it’s IDL.

6 The Object Request Broker

6.1 The OBJECT REFERENCE

Every CORBA object in a system has its own object reference assigned by its ORB at its creation. This remains valid until the object is deleted.

It is required that any ORB understand any object reference at any time. (More on this later.)
6.2 IDL and ORB

Besides using IDL to create the client stub and implementation skeleton, CORBA requires that the ORB store the IDL definition for all of its objects in an interface repository (IR). THIS COLLECTION OF INTERCAE DEFINITIONS IS A KEY RESOURCE IN THE DISTRIBUTED SYSTEM.

Using IDL interfaces, interface definitions can be added to the IR, modified, deleted, or retrieved; the contents of the IR may be searched; and inheritance trees in it may be traced to determine the exact type of an object.

Uses of IRs:

- Interoperability—knowing the types and order of the arguments in a message enables communicating ORBs to translate byte order and data format wherever necessary.

- Interface browsing and debugging.

6.3 THE DYNAMIC INVOCATION INTERFACE (DII)

Sophisticated users of the dynamic distributed object systems demand to use new objects as soon as they are added to any ORB on their network, without having to wait for its installation on their workstation. DII enables a properly written client at RUN TIME to:

i. Discover new objects;

ii. Discover their interfaces;

iii. Retrieve their interface definitions;

iv. Construct and dispatch invocations; and

v. Receive the resulting response or exception information

to and from objects whose client stubs whose are not linked in to its module—e.g., objects that were added to the system after the client was written.

How does client figure out which object or interface it wants to retrieve from the IR? E.g. Naming and Trader Services.
7 CORBA-based Interoperability

Interoperability in CORBA is based on ORB-to-ORB communication. The client does nothing different for a remote invocation compared to the local case. It passes its usual IDL-based invocation to its local ORB. The ORB decides whether to send to a local object or to a remote ORB to be forwarded to the remote object.

Each ORB is required to maintain at least two databases or directory systems: the IR with its collection of interface definitions and an implementation repository with information about available object implementations.

7.1 Integrating a Purchased Object Component

When you purchase a new software written by someone else, you will receive both an executable object implementation and an IDL file from the vendor. Install the object on the ORB on the server node where it will reside. The installation process will produce an object reference, which may be automatically placed in a directory or naming service for you, or may be written to a file where you can access it.

Now when you are ready to write the client. Move to your development platform (may be different from the server’s ORB). Load the IDL file and compile it with the local IDL compiler. It does not matter which language the vendor used to write the object you purchased. You can choose your own language. Discard the skeleton. Use the client stub to access the ORB from your client. In your client code, retrieve the object reference from the NAMING SERVICE or FILE where it was stored it to invoke operations on purchased object, wherever it might reside.

8 The OBJECT MANAGEMENT ARCHITECTURE

8.1 CORBA Services in OMA

- Lifecycle Service
- Relationship Service
• Naming Service
• Persistent Object Service
• Externalization Service
• Event Service
• Object Query Service
• Object Properties Service
• Object Transaction Service
• Concurrency Service
• Licensing Service

Other specifications in progress:
• Trader Service
• Security Service
• Secure Time Service
• Object Collection Service
• Object Startup Service

9 Understanding the ORB

There are four steps to a dynamic invocation:

1. Identify the object you want to invoke
2. Retrieve its interface
3. Construct the invocation
4. Invoke the request, and receive the results.
9.1 Identifying a target object:

In a dynamic distributed object environment, users will probably consult an Object Trading Service to locate object implementations.

The Trader service is more like a combination Yellow Pages and mail-order catalog which lists every available service with ancillary information such as features, location, or cost. Most users will probably access Trader via browsers, clicking through services until they find one they like. Additional use will come from programs written to access Trader directly.

In this context, the desktop client becomes a generic command center, which activates with few (if any) interfaces enabled via static stubs, and proceeds to configure itself to invoke all of the actions its users wants via the DII.

9.2 Retrieving the Target Interface

First, using the object reference obtained from the Naming or Trading service, the client invokes the ORB operation get_interface. This returns not the interface itself, but an object reference that returns the top-level components of the interface when passed to the IR.

Additional calls return all of the interface’s operations (by name), their parameters, and their types.

We will need more information in order to construct an invocation: what each operation does, the function of each parameter, allowed parameter ranges, allowable sequence of operations, and anything else that might be helpful (or necessary). They won’t come from IR; the logical place to look is the Trader.

9.3 Constructing the Invocation

DII provides standard interfaces for constructing a request.

10 The Interface Repository

The interface repository is crucial to the operation of CORBA. CORBA requires that each ORB bear and implement the IR interface, allowing the IDL definitions for all of the objects it knows about to be stored, modified,
and retrieved. These definitions can be used for many different purposes; the CORBA specification manual points out three ways the ORB can use them directly:

- To provide interoperability between different ORB implementations.
- To provide type-checking of request signatures, whether a request was issued through the DII or through a stub.
- To check the correctness of inheritance graphs.

10.1 Using the IR

There are two basic ways you can access the interface repository:

- You can use utilities provided by your ORB vendor
- You can write code that invokes the standard IR IDL interface mandated by OMG.

10.2 Identifying an IR

It is not necessary to have a one-to-one correspondence between IRs and ORBs.

- A particular IR may be shared by more than one ORB; and
- An ORB may access more than one IR.

How does an ORB keep track of IRs? Your company can maintain a number of IRs simultaneously, for different purposes. ORBs will assume that two IRs with the same RepositoryID contain identical interface definitions; i.e., replicated IRs.
10.3 How do IRs work?

An implementation of IR requires some form of persistent object store. However, there is no specification as how to implement them. For example, an IR may be implemented on top of a database. If the database is replicated, remote ORBs will retrieve interfaces quickly from local copies, but changes will propagate with a latency characteristic of the underlying database implementation.