CORBA – Common Object Request Broker Architecture

CORBA (Common Object Request Broker Architecture) is a distributed object standard defined by the Object Management Group (OMG) in the early 1990s. Its goal was to enable software components written in different programming languages and running on different platforms to communicate seamlessly.

CORBA introduced the concept of an Object Request Broker (ORB), which acts as middleware between clients and distributed objects. In essence, CORBA extended the object-oriented programming paradigm into distributed systems by allowing so-called “distant objects” (remote objects) to interact as if they were local.

This was a major step in software integration before REST APIs and microservices became dominant.

Object Request Broker (ORB)

The **ORB** is the central component of CORBA. It is responsible for:

- locating remote objects,
- forwarding client requests,
- handling communication between distributed components,
- managing object references.

From the programmer’s perspective, calling a remote object method in CORBA looks very similar to calling a local method. The ORB hides the complexity of network communication.

Conceptually:

Client → ORB → Remote Object

Remote Object → ORB → Client

The ORB handles all low-level details such as transport protocols and data encoding.

Distributed Objects Concept

CORBA extends object-oriented principles to distributed environments.

Remote objects:

- expose methods,
- encapsulate data,
- can be invoked across networks.

Unlike REST (which is resource-based) or JSON-RPC (which is procedure-based), CORBA is object-based. Objects can share both state (data) and behavior (methods).

This design reflects classic OOP thinking in distributed systems.

IDL – Interface Definition Language

CORBA introduced a language-independent interface definition mechanism called IDL (Interface Definition Language).

IDL allows developers to define object interfaces independently of programming languages.

Example IDL:

```
interface Calculator {
   long add(in long a, in long b);
};
```

From this IDL definition, language-specific stubs and skeletons are generated automatically (e.g., for C++, Java, Python).

This ensures:

- language interoperability
- strict interface contracts
- platform independence

IDL was one of the early solutions to cross-language service integration.

CORBA IDL supports interface inheritance, allowing one interface to extend another. This enables the reuse of method definitions and supports polymorphism in distributed systems.

Below is an example from a simplified employee management system.

```
module Company {

interface Person {
   string getName();
   long getId();
};

interface Employee : Person {
   double getSalary();
   void setSalary(in double newSalary);
};

interface Manager : Employee {
   void addTeamMember(in Person p);
};
```

```
};
```

Marshalling and Unmarshalling

CORBA introduced standardized marshalling and unmarshalling mechanisms.

This mechanism is essential in distributed systems because memory representations differ across:

- programming languages
- operating systems
- CPU architectures (32-bit vs 64-bit, little-endian vs big-endian)
- Without marshalling, raw in-memory data could not be safely transmitted between heterogeneous systems.

Marshalling: converting objects or parameters into a platform-independent serialized format for transmission.

Unmarshalling: reconstructing the original objects from the serialized data on the receiving side.

This ensures that data types remain consistent across systems, different hardware architectures can communicate, and programming language differences are abstracted.

Service Registry Concept

CORBA introduced early ideas similar to a service registry.

Objects could register themselves in a naming service, allowing clients to discover them dynamically.

This resembles:

- modern service discovery systems (e.g., Consul, Eureka),
- DNS-based service resolution,
- microservice registries.

Services were browsable in a uniform manner, which was advanced for its time.

Known Implementations

Well-known CORBA implementations include:

- ORBix
- VisiBroker
- TAO

Java-based CORBA implementations

- Java RMI (conceptually related, though not CORBA-based)

CORBA support has historically been included in enterprise application servers such as:

- JBoss
- IBM WebSphere

Although less common today, CORBA is still present in:

- banking systems,
- telecommunications,
- aerospace and defense systems,
- legacy enterprise infrastructures.

Why CORBA Declined

Despite its strong architectural design, CORBA gradually declined due to:

- high complexity
- steep learning curve
- heavyweight configuration
- firewall/NAT issues
- verbose tooling
- rise of simpler HTTP-based APIs (REST)

Tutorial:
https://medium.com/@diremund/understanding-corba-building-a-banking-system-with-java-47c67fc42a66