CORBA: Common Object Request Broker Architecture in Software Engineering

Samuel Jursík, Martin Mašek

October 15, 2024

Outline

What is CORBA?

- CORBA stands for **Common Object Request Broker Architecture**.
- It is a **middleware** standard developed by the **Object Management Group (OMG)**.
- CORBA allows different applications (even in different programming languages) to communicate with each other.
- It supports distributed, cross-platform computing by enabling different objects within a network to easily interact and work together.

Who is OMG?

- The **Object Management Group (OMG)** is an international technology standards consortium.
- Founded in **1989**, OMG creates and maintains standards for distributed systems and software interoperability.
- OMG is best known for developing **CORBA**, **UML** (Unified Modeling Language), and **BPMN** (Business Process Model and Notation).
- It includes a wide range of members, from large companies to small organizations, working together to define technology standards.

Why CORBA?

- **Problem:** Applications written in different programming languages and running on different platforms often struggle to communicate.
- **Solution:** CORBA provides a **unified way** to allow these diverse systems to talk to each other, as if they were part of the same application.
- **How it works:** CORBA abstracts the complexities of network communication, so developers can focus on the logic of their applications rather than the technical details of making different systems work together.
- **Goal:** Enable **seamless interoperability** in distributed systems, making it easier to build scalable, flexible, and reusable software components.

Key Concepts of CORBA

- **ORB (Object Request Broker):** Core component that enables communication between clients and servers.
- **IDL (Interface Definition Language):** Language-neutral specification of interfaces.
- **Stubs and Skeletons:** Generated code that mediates between the client and the server.
- **GIOP/IIOP:** Protocols used by CORBA for communication over networks.

The Problem CORBA Solves

- Imagine chefs from different countries want to **share recipes**.
- They each speak different **languages**, use different **ingredients**, and have unique **cooking techniques**.
- The chefs want to collaborate but can't understand each other directly.
- **CORBA** is like an international system that helps these chefs exchange recipes easily.

The Chefs Analogy

- Each chef represents a different software program:
  - Chef A speaks **English** and uses **American techniques**.
  - Chef B speaks **French** and uses **French techniques**.
  - Chef C speaks **Chinese** and uses **Chinese techniques**.
- They need a system that can **translate recipes and requests** between them.
- This system is **CORBA**.

ORB: The Translator (Middleman)

- **ORB (Object Request Broker)** acts like a **translator**.
- Chef A can send a recipe request to Chef B in French, but the ORB handles translating it from English to French.
- Chef B sends back the recipe in French, and the ORB translates it to English for Chef A.
- No matter which language the chefs use, the ORB ensures they understand each other.

IDL: The Universal Recipe Format

- Just like every chef needs to understand the basic structure of a recipe, CORBA uses **IDL (Interface Definition Language)**.
- **IDL** is a universal format that describes the ingredients, steps, and tools needed for a recipe.
- No matter what language or cooking style the chef uses, IDL makes sure the recipe can be understood by everyone.
- In technical terms, IDL defines **how programs communicate**, no matter what language they are written in.

Client and Server: The Recipe Exchange

- In CORBA terms:
  - Chef A is the **client**, requesting a recipe.
  - Chef B is the **server**, providing the recipe.
- The ORB (translator) acts as the **middleman**, ensuring both chefs can communicate smoothly.
- The chefs don't need to know each other's cooking tools or location—CORBA handles it all.

Object Request Broker (ORB)

- ORB is the middleware that handles communication between objects.
- It locates the server objects, passes requests from client to server, and returns results back to the client.
- ORB abstracts details like network communication, providing seamless communication between applications.
- It uses **GIOP (General Inter-ORB Protocol)** and **IIOP (Internet Inter-ORB Protocol)** to send and receive messages over the internet.

Interface Definition Language (IDL)

- IDL is a language-agnostic specification for defining the interfaces of objects.
- It ensures that different programming languages can understand the interfaces and communicate.
- For each IDL interface, CORBA generates **stubs** (client-side) and **skeletons** (server-side) in the required programming language.
- Programmers write an IDL file that specifies the methods and data types for the objects they want to use.
- Example of an IDL interface:

IDL Example

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

How Stubs and Skeletons Work

- **Stubs:**
  - On the client's side.
  - When the client calls a remote object (on another computer), the stub prepares the request.
  - The stub sends the request to the server, making it seem like the object is local.
- **Skeletons:**
  - On the server's side.
  - The skeleton receives the request from the client (via the stub).
  - It unpacks the request and makes sure the correct method on the server is called.
  - Then, it sends the result back to the client.

GIOP - General Inter-ORB Protocol

- **GIOP (General Inter-ORB Protocol)** defines the abstract communication protocol for CORBA ORBs (Object Request Brokers).
- It allows for standardized communication between CORBA-compliant systems.
- GIOP is designed to be independent of the transport layer, making it flexible across different networks.
- Ensures interoperability between CORBA ORBs across different vendors and platforms.

IIOP - Internet Inter-ORB Protocol

- **IIOP (Internet Inter-ORB Protocol)** is a concrete implementation of GIOP over TCP/IP networks.
- It allows CORBA-based systems to communicate over the Internet.
- IIOP defines how GIOP messages are exchanged using the TCP/IP protocol stack.
- With IIOP, CORBA objects can be accessed across distributed systems, regardless of location.

IOR - Interoperable Object References

- **IOR (Interoperable Object References)** are unique identifiers for CORBA objects.
- An IOR contains all the necessary information for a client to locate and communicate with a CORBA object.
- **Structure of an IOR:**
  - Object Key: Uniquely identifies the object on the server.
  - Protocol and Address Information: Specifies the protocol (e.g., IIOP) and the address of the server hosting the object.
  - Optional components: Include information such as security attributes.
- IORs are essential for making remote CORBA objects accessible and usable in a distributed system.
- **Encoding and Transmission:**
  - IORs are encoded in a string format for easy transmission.
  - They can be passed between clients and servers to enable remote method invocation.

Static vs. Dynamic Invocation in CORBA

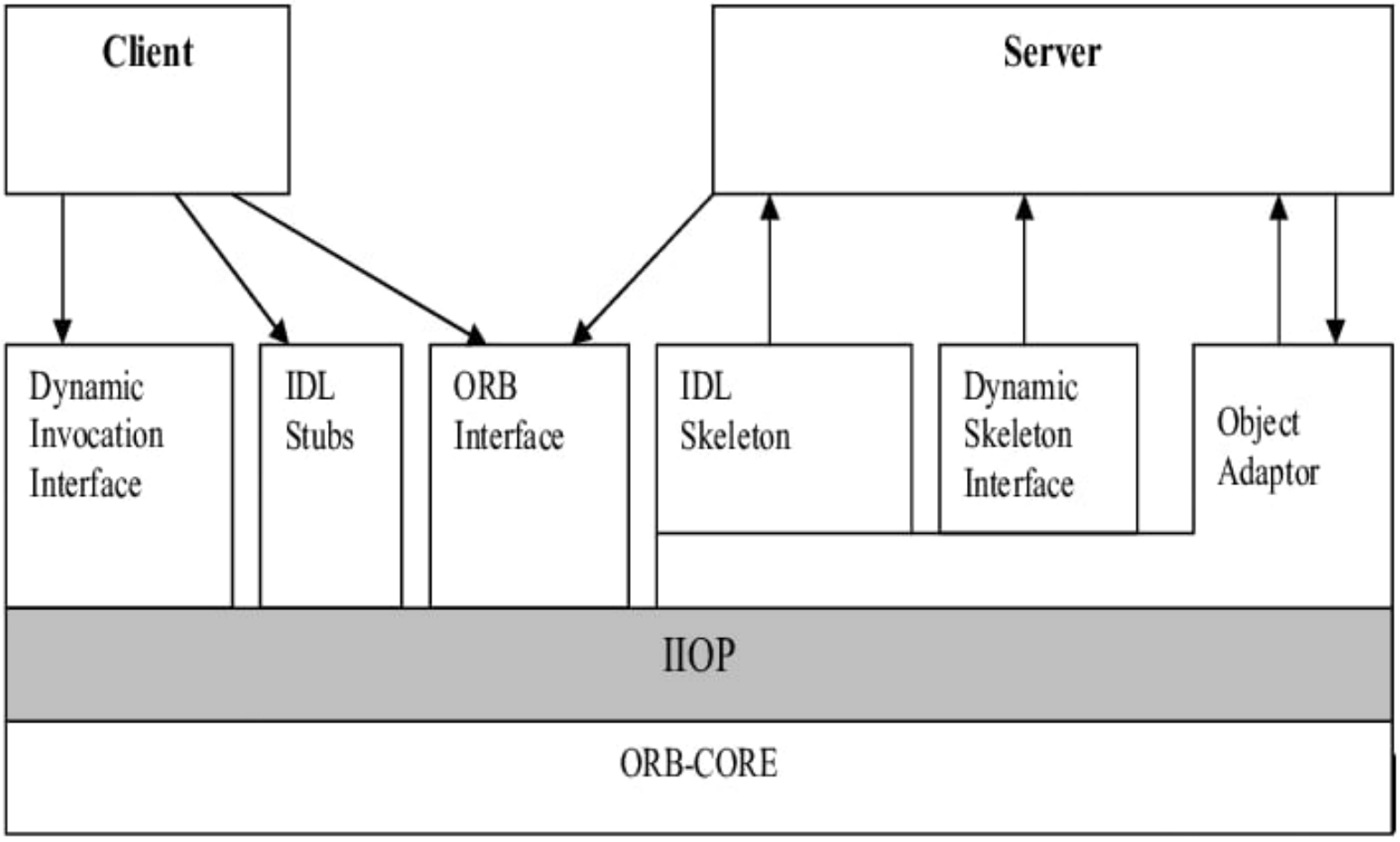
Static Invocation

- **Definition:** Method calls are determined at compile time.
- **Use Case:** When method signatures are known beforehand.
- **Advantages:**
  - Better performance due to compile-time optimizations.
  - Strong type checking, reducing runtime errors.
- **Example:** Using stubs generated from IDL files to call methods directly.

Dynamic Invocation

- **Definition:** Method calls are determined at runtime.
- **Use Case:** When methods or interfaces may change, or are unknown at compile time.
- **Advantages:**
  - Greater flexibility and adaptability to changes.
  - Supports loose coupling and service discovery.
- **Example:** Invoking methods using the Dynamic Invocation Interface (DII) without prior knowledge of method signatures.

CORBA architecture



Formal specification of CORBA-based distributed objects and behaviors - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Components-of-the-CORBA-architecture-and-their-interconnections$_{f}ig1_{4}187856[accessed14Oct2024]$

CORBA Implementation

- **CORBA is a Specification:** It is not a language or a specific software product, but a **standard**.
- **Middleware Implementations:** Many vendors provide their own implementations of the CORBA specification, such as:
  - **ORB implementations** (e.g., **TAO**, **Orbix**, **JavaORB**).
  - These ORBs handle communication and follow the CORBA standards to ensure interoperability between applications.
- **Platform-Neutral:** CORBA enables communication across different operating systems and hardware, abstracting the underlying details of how this happens.

Example of CORBA using OmniORB:

A high-performance CORBA Object Request Broker that supports C++ and Python.

IDL File: Defining the Interface

The IDL file defines the interface for communication between the client and server.

IDL Example

```
module MyModule {
  interface MyInterface {
    string sayHello(in string name);
  };
};
```

Explanation:

- The `module` defines a namespace.
- The `interface` defines the methods available to the client.
- The method `sayHello` takes an input string and returns a string.

Server Code: ORB Initialization

The server initializes the ORB and registers the servant object.

```
import sys
from omniORB import CORBA
import MyModule_idl

class MyInterface_impl(MyModule_idl.MyInterface):
    def sayHello(self, name):
        return "Hello, " + name

orb = CORBA.ORB_init(sys.argv)
poa = orb.resolve_initial_references("RootPOA")
poa_manager = poa._get_the_POAManager()

my_interface_impl = MyInterface_impl()
obj_ref = poa.servant_to_reference(my_interface_impl)
ior = orb.object_to_string(obj_ref)

with open("ior.txt", "w") as f:
    f.write(ior)

poa_manager.activate()
orb.run()
```

Client Code: Accessing the Server

The client initializes the ORB, reads the IOR, and communicates with the server.

```
import sys
from omniORB import CORBA
import MyModule_idl

orb = CORBA.ORB_init(sys.argv)

with open("ior.txt", "r") as f:
    ior = f.read().strip()

obj = orb.string_to_object(ior)
my_interface = obj._narrow(MyModule_idl.MyInterface)

result = my_interface.sayHello("World")
print(result)
```

CORBA Use Cases

- **Distributed Systems:** CORBA facilitates communication between objects across different locations.
- **Heterogeneous Environments:** Allows integration of systems written in different programming languages.
- **Enterprise Systems:** CORBA is commonly used in large, distributed enterprise applications.
- **Legacy Systems Integration:** CORBA helps bridge old and new systems.

Advantages of CORBA

- **Language and Platform Independence:** Facilitates interoperability across different languages and systems.
- **Distributed Objects:** Makes it easier to build distributed, object-oriented systems.
- **Scalability:** Suitable for large enterprise applications.

Disadvantages of CORBA

- **Complexity:** Setting up and managing CORBA systems can be complex.
- **Performance Overhead:** Communication between distributed objects can introduce latency.
- **Obsolescence:** Modern technologies like RESTful APIs, gRPC, and microservices have reduced CORBA's relevance.

CORBA vs. Modern Alternatives

- **CORBA** was widely used in the 1990s and early 2000s.
- Modern alternatives like **gRPC**, **RESTful APIs**, and **SOAP** have become more popular.
- CORBA is still used in some legacy systems but has largely been replaced by simpler, more efficient technologies.

Conclusion

- CORBA played a significant role in enabling distributed object-oriented computing.
- Its language and platform independence were key benefits in heterogeneous environments.
- Despite its decline in use, understanding CORBA is important for working with legacy systems and appreciating the evolution of distributed computing architectures.