



Informatica Industriale

Prof. Davide Brugali

2.2 – CORBA

CORBA

- CORBA is a standard defined by the Object Management Group: **COMMON OBJECT REQUEST BROKER Architecture**
 - exists since 1991.
- CORBA is mature as enterprise middleware:
 - scalability;
 - fault tolerance;
 - security;
 - transactions;
- High level of industry support.
- But CORBA is still low-level:
 - you must integrate the architecture yourself.

Why CORBA?

❑ CORBA is a middleware standard:

- before CORBA, middleware was dominated by proprietary tools;
- CORBA makes it possible to be independent from vendors.

❑ CORBA is a "software bus":

- the basis of a component architecture;
- defines a standard way to plug components.

❑ CORBA is a distributed object architecture:

- implements 3-tier and multi-tier architectures.

2.2 - CORBA

Why CORBA?

❑ Characteristics

- 1 Interoperability: objects communicate independently from programming language, OS, network protocols.
- 2 Location transparency: objects are binding at run-time ("locators").
- 3 Interface/implementation: IDL allows interfaces to be defined in a dedicated language.
- 4 Self-descriptiveness: the interface repository makes it possible to discover services dynamically.

❑ Proposes technical services (CORBA services):

- Persistence, Transactional, Security, Concurrency, Lock, Events ...

2.2 - CORBA

Informatica Industriale - Prof. Davide Bruggi

4/17

Component Interfaces

- ❑ Interfaces are often described using an Interface Definition Language (IDL)
- ❑ IDLs are programming-language independent
- ❑ IDLs can generate proxy objects which permit cross language method calls.

Interface Definition Languages

- ❑ COM and CORBA all use an Interface Definition Language (IDL)
- ❑ IDLs:
 - Specify the contract that a server provides
 - Define the types passed between client and server
 - Permit cross-language method invocation
 - Are used to generate metadata
 - Are used to generate proxy objects

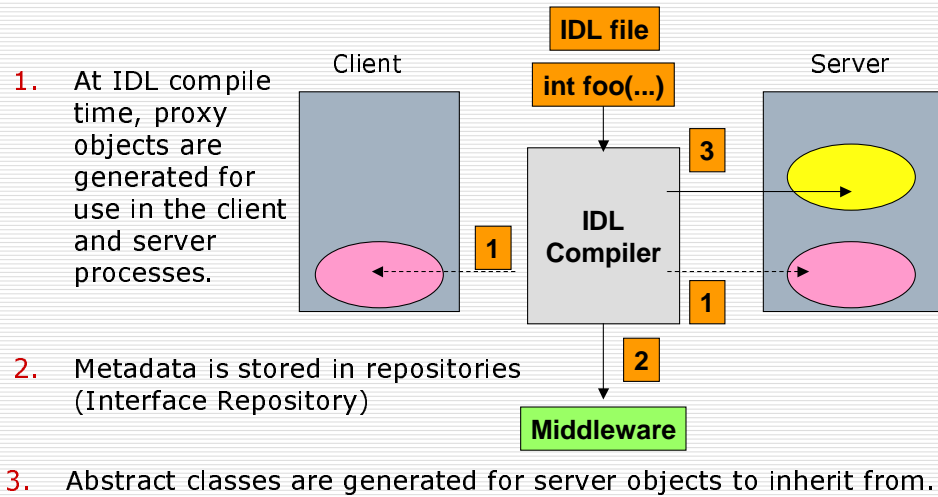
IDL files

- ❑ Programmers write an IDL file to express the interfaces that their server provides.
- ❑ IDLs are often based on C/C++ style syntax.
 - CORBA IDL compilers can put metadata in an interface repository.
 - COM IDL compiler can put metadata in a type library.
 - Java provides a Class Object for every type.
 - .NET provides a Type Object for every type.

IDL - Operations

```
module CORBA
{
    interface ORB
    {
        typedef string ObjectId;
        typedef sequence <ObjectId> ObjectIdList;
        exception InvalidName {};
        ObjectIdList list_initial_services ();
        Object resolve_initial_references (
            in ObjectId identifier)
            raises (InvalidName);
    }
}
```

IDL Compile

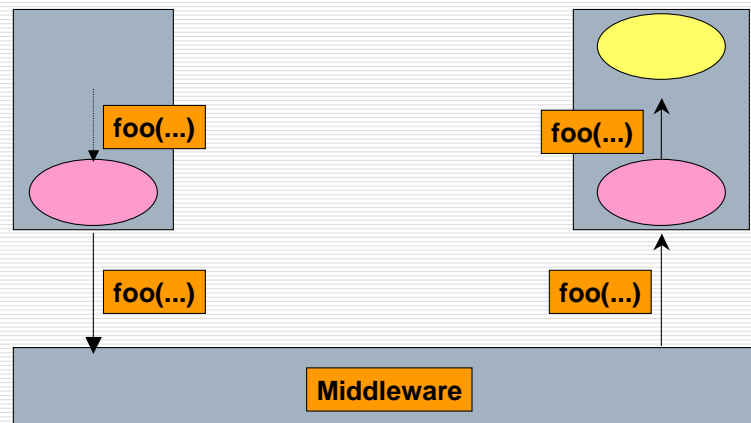


2.2 - CORBA

Informatica Industriale - Prof. Davide Brugali

9/17

Run-Time Invocation

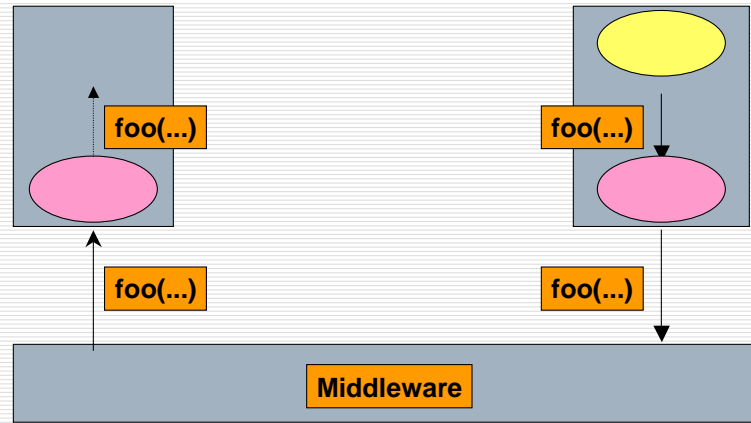


2.2 - CORBA

Informatica Industriale - Prof. Davide Brugali

10/17

Run-Time Return

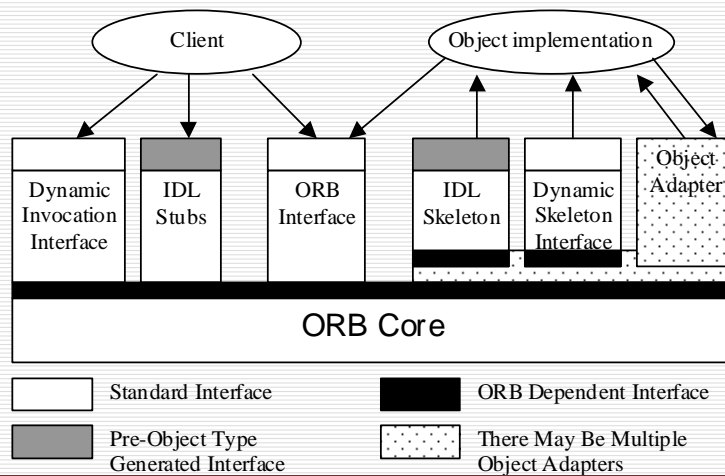


2.2 - CORBA

Informatica Industriale - Prof. Davide Brugali

11/17

Struttura dell'ORB



2.2 - CORBA

Informatica Industriale - Prof. Davide Brugali

12/17

CORBA Services

- ❑ The **Trader Service** is similar to the yellow pages. It allows client objects to find out which distributed server objects support a given interface.
- ❑ The **Naming Service** allows client objects to identify which interface is supported by an object that has been registered with a given name.
- ❑ In some cases, the client object has to request the service of a server object, whose interface was not known at compile-time. This means that the client object does not have a stub of the remote server object, but it can use a generic interface, called **Dynamic Invocation Interface** that makes it possible to construct method invocations at run-time.

CORBA Services

- ❑ **Persistent object** service for storing and retrieving objects in/from a persistent storage
- ❑ **Transaction service** for atomic database operations
- ❑ **Life-cycle** service to create, delete, copy and move objects
- ❑ **Event service** for asynchronous communication among distributed objects.

Esempio

```
package mioServer;
public interface Calcolatrice extends org.omg.CORBA.Object {
    public int ultimoRisultato();
    public int addiziona(int op1, int op2);
    public int sottrai(int op1, int op2);
}
```

```
//File server.idl
module mioServer {
    interface Calcolatrice {
        long ultimoRisultato();
        long addiziona(in long op1, in long op2);
        long sottrai(in long op1, in long op2);
    };
};
```

ORB Server

```
// File: startCalcServer.java
public class startCalcServer {
    public static void main( String[] args ) {
        // Inizializza l'ORB.
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( args, null );
        // Inizializza il BOA.
        org.omg.CORBA.BOA boa = orb.BOA_init();
        // Crea un'istanza del nostro server.
        mioServer.Calcolatrice unaCalcolatrice=new mioServer.CalcolatriceImpl("CalcServer");
        // Comunica all'ORB l'esistenza dell'istanza appena creata.
        boa.obj_is_ready( unaCalcolatrice );
        System.out.println( unaCalcolatrice + " è attiva." );
        // Aspetta le chiamate dei client.
        boa.impl_is_ready();
    }
}
```


Client (Applet)

```
// Il metodo init() inizializza l'ORB.  
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( this, null );  
  
// Il metodo bind() trova un'istanza attiva del nostro  
// server e ne restituisce un riferimento.  
calcolatrice = mioServer.CalcolatriceHelper.bind( orb, "CalcServer" );
```