Programação em Sistemas Distribuídos

MEI-MI-MSI

2018/19

CORBA

Prof. António Casimiro

CORBA

Ciências ULisboa

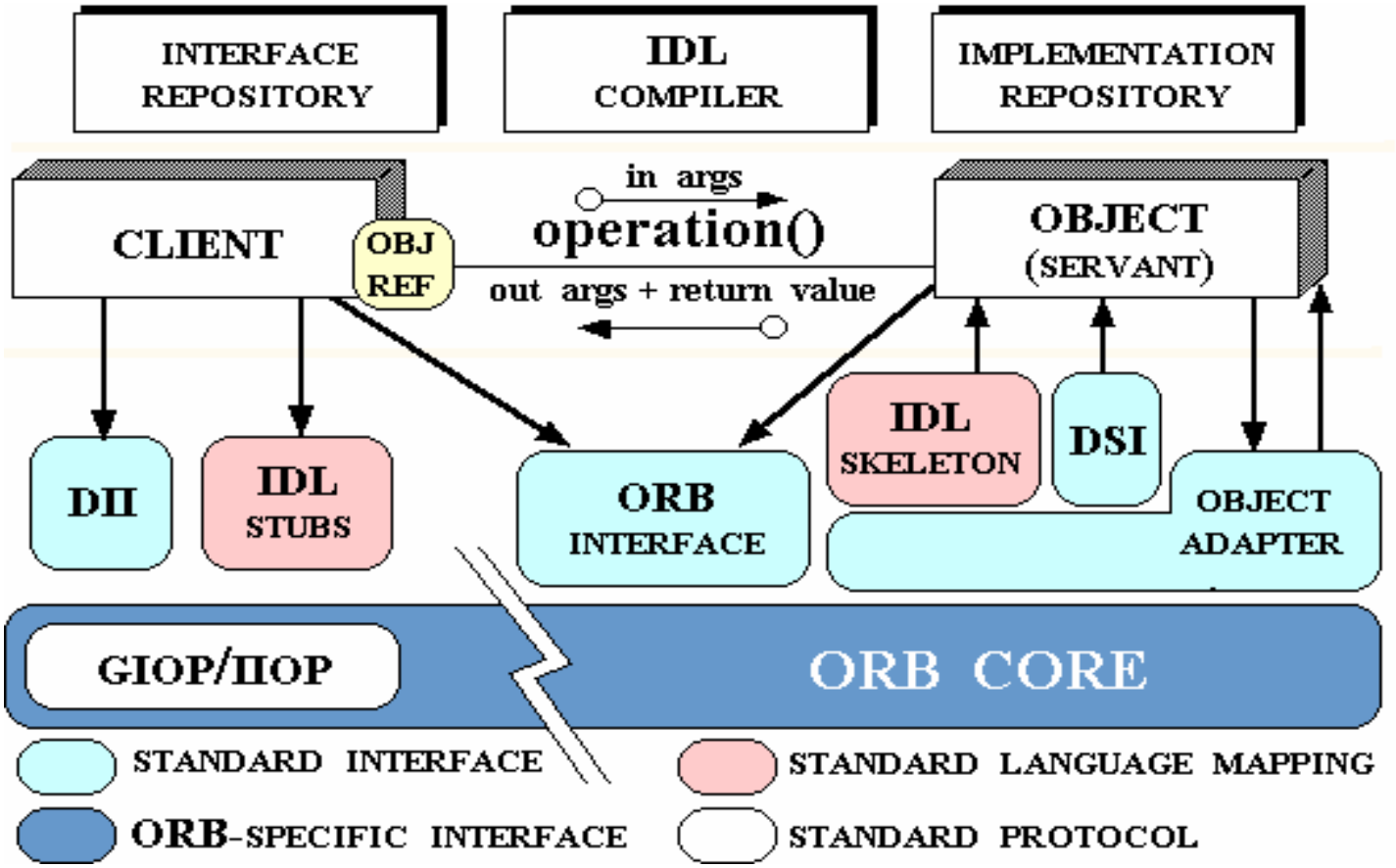
- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- Platform independent: hardware, operating system, programming language, network protocols, etc.
- Set of specifications defined by OMG (**O**bject **M**anagement **G**roup)
- Among them, OMG defined a remote object invocation service

The CORBA architecture
Fundamental concepts

Ciências ULisboa

- **IOR** (Interoperable Object Reference)
  - "Contact details" used by clients to communicate with a CORBA object
  - Uniquely identifies an object on a remote CORBA server
  - IOR works (or interoperates) across different CORBA implementations
- **ORB** (Object Request Broker)
  - Underlying communication support
- **OA** (Object Adapter)
  - Connects a request using an object reference with the proper code to service that request
- **GIOP** (General Inter-ORB Protocol)
- **IIOP** (Internet Inter-ORB Protocol)
- **IDL** (Interface Definition Language)
- **Objects** (Implement the defined interface)

The CORBA architecture
Components view

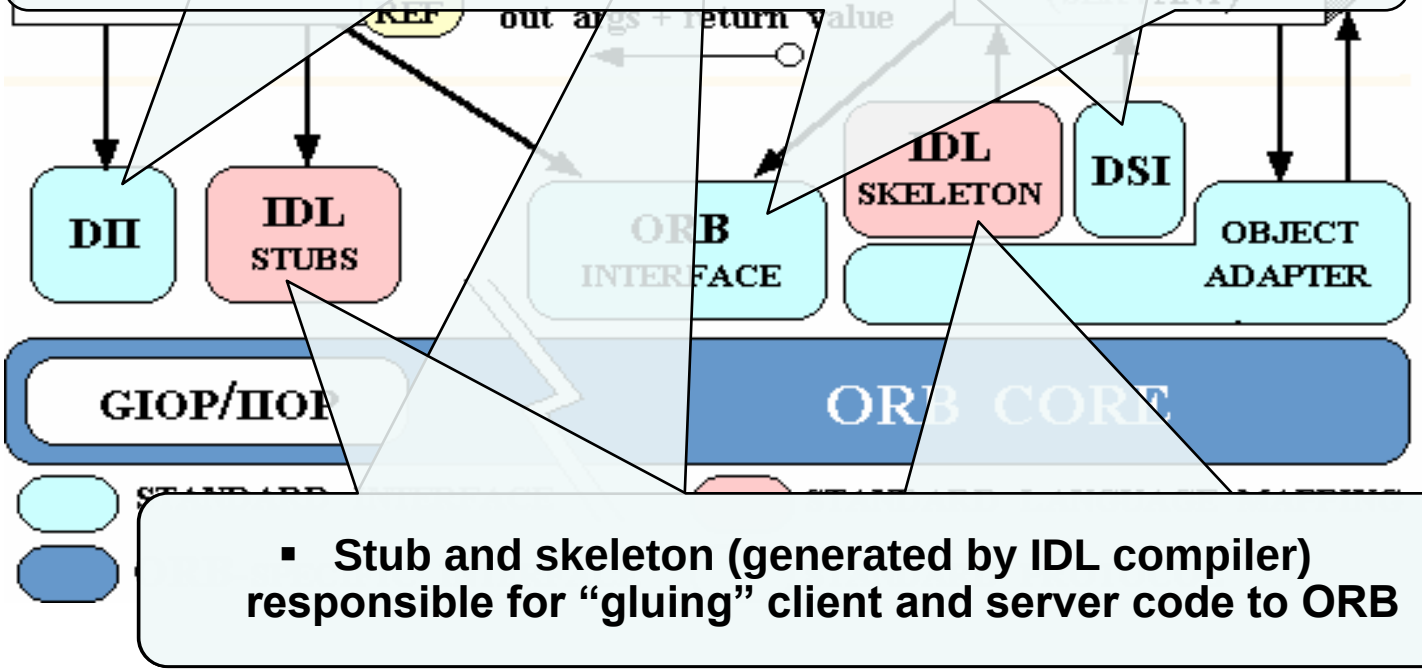


The CORBA architecture

Components view

- Dynamic Invocation Interface – allows calling methods without accessing stub (without IDL)
- Dynamic Skeleton Interface – allows calling methods of implementations which ignore at compile time their interface

- ORB interface – implements methods and their interfaces useful to all objects

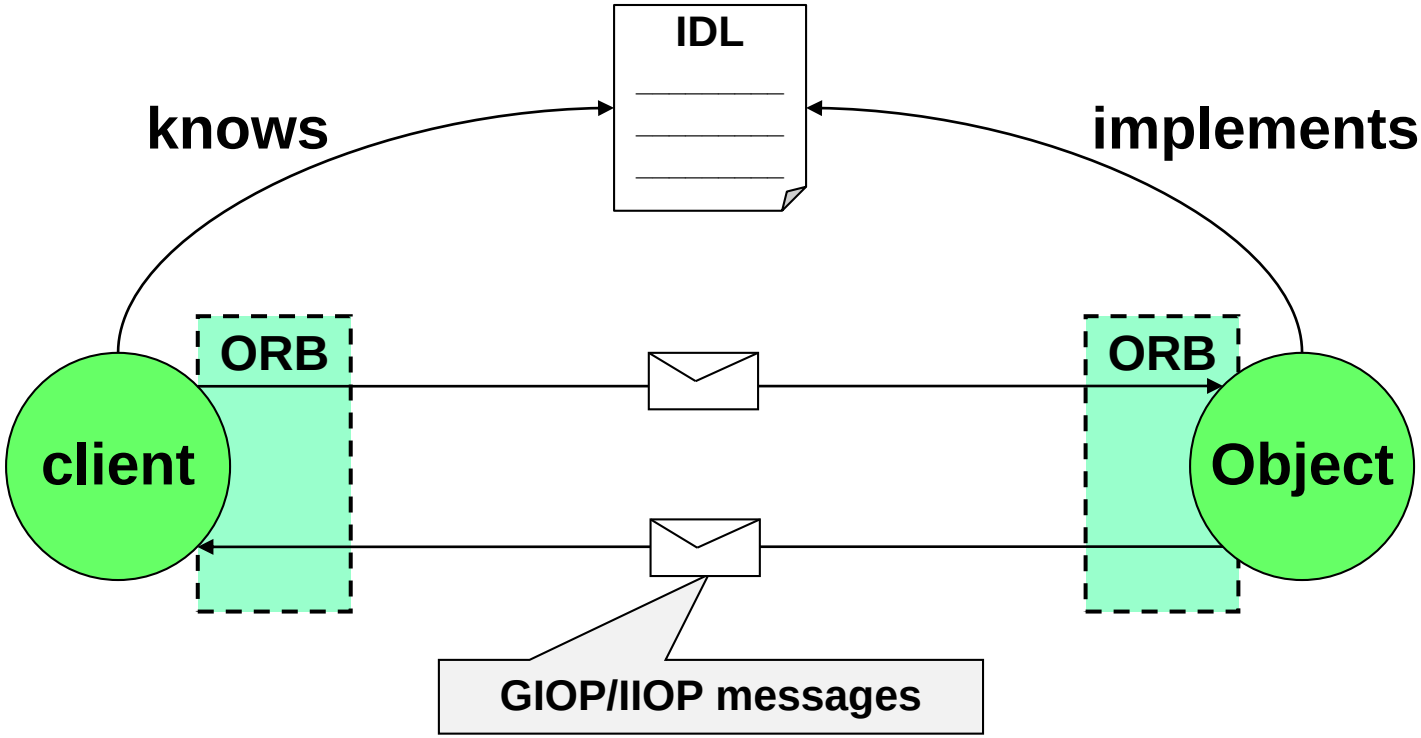


- Stub and skeleton (generated by IDL compiler) responsible for “gluing” client and server code to ORB

IDL

- Interface Definition Language
- Used for describing the interfaces of the CORBA objects
- Syntax similar to C++, but only containing the declarative part
- There are IDL mappings available for several programming languages, e.g., Java, C, C++, Lisp, Ada and Cobol

CORBA
Overview of interactions



Ciências
ULisboa

Design elements

“As is”, culled from several sources on manuals and the Internet, with aim of giving design and programming examples

Example: IDL Interface

```
module hello{
  interface HelloServer{
    readonly attribute long numberOfHellos;
    readonly attribute string name;

    string sayHello(in string name);
  };
};
```

Example: Interface implementation

```
package hello;

public class HelloServerImpl extends HelloServerPOA {
  private String name;
  private int numberOfHellos;

  public HelloServerImpl(String name){
    this.name = name;
    this.numberOfHellos = 0;
  }

  public String name(){
    return name;
  }

  public int numberOfHellos(){
    return numberOfHellos;
  };

  public String sayHello(String name){
    numberOfHellos++;
    return "Hello "+name;
  }
}
```

Example: Server

Ciências ULisboa

```
package hello;

import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import java.io.*;

public class Server {
  public static void main(String[] args)throws Exception {
    String name = args[0];
    String refFile = args[1];

    ORB orb = ORB.init(args,null);

    org.omg.CORBA.Object ref = orb.resolve_initial_references("RootPOA");
    POA poa = POAHelper.narrow(ref);
    poa.the_POAManager().activate();

    HelloServerImpl helloImpl = new HelloServerImpl(name);

    ref = poa.servant_to_reference(helloImpl);

    PrintWriter out = new PrintWriter(new FileWriter(refFile));
    out.println(orb.object_to_string(ref));
    out.close();

    orb.run();
  }
}
```

Example: Client

```
package hello;

import org.omg.CORBA.*;
import java.io.*;

public class Client{
  public static void main(String[] args) throws Exception {
    String name = args[0];
    String refFile = args[1];

    ORB orb = ORB.init(args,null);

    BufferedReader in = new BufferedReader(new FileReader(refFile));
    String ior = in.readLine();
    in.close();

    org.omg.CORBA.Object ref = orb.string_to_object(ior);

    HelloServer hello = HelloServerHelper.narrow(ref);

    System.out.println("name: "+hello.name());
    System.out.println("number of hellos: "+hello.numberOfHellos());
    System.out.println(hello.sayHello(name));
  }
}
```

IDL
Modules and interfaces

Ciências ULisboa

- **Modules**:
  - Define a Java or UML package
  - Syntax:
    - module <name>{...};
- **Interfaces**:
  - Define a CORBA object interface
  - Interfaces can inherit from another interface
  - Syntax:
    - interface <name>:<super>,<super>...{...};

IDL
Attributes and operations

Ciências ULisboa

- **Attributes**:
  - Define a read only or read/write interface attribute (a variable implemented by the object)
  - Sintaxe:
    - [readonly] attribute <type> <nnme>;
- **Operations**:
  - Define interface methods
  - Call semantic is synchronous (blocking) by default, but may be non-blocking if defined as oneway
  - Parameters can be passed by value (in), by reference (inout), or can be return values (out)
  - Syntax:
    - [oneway] <type> <method name>
      (in|out|inout <type> <name>[,...])
      [raises {<exception>}];

IDL
Exceptions

Ciências ULisboa

- **Exceptions**:
  - Besides system pre-defines exceptions, the user can also define exceptions in the interface
  - An exception is a special conditions that may be raised during the execution of a remote method
  - Syntax:
    - exception <name> {<field>[;...]};
- Fields:
  - An exception may contain several fields, like in a structure
  - Syntax:
    - <type> <name>;

IDL
Basic data types

Ciências ULisboa

- Basic data types or primitive types
  - `boolean`
  - `octet`
  - `long`
  - `long long`
  - `char`
  - `string`
  - `float`
  - `double`
  - `long double`
  - `wchar`
  - `wstring`
  - `unsigned short`
  - `unsigned long`
  - `unsigned long long`
  - `any`

- The **any** data type allows you to specify that an attribute value, an operation parameter, or an operation return value can contain an arbitrary type of value to be determined at runtime

IDL
Complex types

Ciências ULisboa

- **Enum** types:
  - A type in which identifiers can be assigned to members of a set of values
  - Syntax:
    - enum <name> {values};
- **Struct** types:
  - Define a “register” with one or several fields
  - Syntax:
    - struct <name> {<fields>};
- And also **unions, sequences** and **arrays**
- In Java mappings, exceptions, enums, structs and unions generate Java classes, whereas sequences and arrays generate arrays

IDL
Contants and type definitions

Ciências ULisboa

- **Constant**:
  - It is possible to define constants in IDL
  - Syntax:
    - const <type> <name> = <expression>;
- **Type definitions**:
  - To name some data type
  - Syntax:
    - typedef <oldType> <newType>;
    - <oldType> can be a basic type or a complex type

IDL
Example

Ciências ULisboa

```
module Finance {
    struct PersonalDetails {
          string name;
          short age;
    }
    interface Bank {
       exception Reject {
          string reason;
       };
       exception TooMany {}; // Too many accounts.
       Account newAccount(in string name) raises (Reject, TooMany);
       PersonalDetails getPersonalDetails(in string name);
       void deleteAccount(in Account a);
    };
    interface Account {
       attribute float balance;
       readonly attribute string owner;

       void makeDeposit(in float amount,  out float newBalance);
       void makeWithdrawal(in float amount, out float newBalance);
       oneway void notice(in string notice);
    };
};
```

CORBA programming

Ciências ULisboa

- CORBA provides a robust and flexible programming model to buid distributed systems
- A considerable part of this model is defined in some objects and classes of the API
- Main objects of the CORBA API:
  - org.omg.CORBA.Object
  - <interface>Helper
  - <interface>POA
  - org.omg.CORBA.ORB
  - org,omg.PortableServer.POA

org.omg.CORBA.Object

Ciências ULisboa

- Representation of a reference
- All stubs extend this class
- The operations are in fact executed by the ORB, but are defined in this class for comodity
- Main operations:
  - **`is_equivalent(obj)`**: Returns true if the two object references are identical. It may return true if they refer to the same object (depends on the ORB implementation)
  - **`non_existent()`**: Returns true if the ORB knows authoritatively that the server object does not exist
  - **`release()`**: Signals that the caller is done using this object reference, so that associated resources can be freed

<interface>Helper

- When compiling IDL to Java, there will be an **helper class** for each interface
- It cannot be instantiated and all its methods are static
- This class provides a set of methods used by the stub and the skeleton of that interface
- It also provides some useful methods for the programmer:
  - **`narrow(obj)`**: Casts a generic object reference (org.omg.CORBA.Object) to a specific interface object type
  - **`insert(any,obj)`**: Inserts a reference to an object that implements some <interface> in a Any generic type
  - **`extract(any)`**: Extracts from a Any generic type a reference to an object that implements <interface>

<interface>POA

- Is the skeleton of an interface
- Is generated by the IDL compiler
  - Every interface has its own skeleton
- The interface implementation must extend this class and implement the abstract methods defined in it
- The only method that is useful for the programmer is `this_object()`, which returns to the calling servant a reference to the object it is incarnating for that request

org.omg.CORBA.ORB

Ciências ULisboa

- This interface defines operations that are useful to the programmer, independently of the object adaptor or the used interface
- Obtaining a reference to the ORB is the first step in any program in which CORBA is used
- The operation defined in this interface are implemented directly in the ORB core

org.omg.CORBA.ORB

Ciências ULisboa

- Main ORB operations:
  - **`init(str[],props)`**: This static method must be called as a first step to obtain a reference to the ORB, while indicating some properties that the ORB might need to exhibit
  - **`run()`**: This method is typically used on the server side, to set the ORB waiting for requests. It is blocking and must be called after the initialization has been completed
  - **`work_pending()`**: This method checks if the ORB still has pending tasks, like serving some requests
  - **`perform_work()`**: With this operation it is possible to direct the ORB to perform a unit of work, like serving a pending request

Note: The **`perform_work()`** and the **`work_pending()`** operations can be used in conjuntion, as an alternative to using **`run()`**, in order to allow the server to perform some intermediate tasks in between serving requests

org.omg.CORBA.ORB

Ciências ULisboa

- Main ORB operations (cont):
  - **shutdown(bool)**: Stops the ORB, which will not serve any more requests, but may complete the pending requests, depending on the provided parameter
  - **destroy()**: This method should be called after **shutdown()**, in order to free all resourced used by the ORB
  - **object_to_string(obj)**: This method transforms an object reference into the corresponding string of the form `IOR:...`
  - **string_to_object(str)**: This method transforms a string of the form `IOR:...` into the corresponding object reference. It also accepts URLs of the form `corbaloc:...`
  - **resolve_initial_references(str)**: Resolves a specific object reference from the set of available initial service names, which might be set through initialization properties

org.omg.PortableServer.POA

Ciências ULisboa

- Default object adaptor of CORBA
- Responsible for the activation and management of object in the ORB
- It is possible to define a hierarchy of POAs in the server, with specific policies for the threading model, persistence of references, etc.
- The root POA ("RootPOA") can be obtained by calling `resolve_initial_references("RootPOA")` in the ORB

org.omg.PortableServer.POA

Ciências ULisboa

- Main operations:
  - **the_name()**: Returns the name of the POA
  - **the_POAManager()**: Returns a reference to the POAManager that controls this POA. The POAManager implements the method **activate()**, which must be called to activate the POA, enabling it to process requests
  - **activate_object(servant)**: Activates the servant object returning the object ID assigned by the POA to the object
  - **deactivate_object(oid)**: Deactivates the object specified by the object ID
  - **servant_to_reference(servant)**: Similar to **activate_object(servant)**, but returns an object reference instead of na object ID