Università degli Studi di Bologna
Facoltà di Ingegneria

Corso di

Reti di Calcolatori M

CORBA - Implementazione

Invocazione statica: prima parte

Luca Foschini

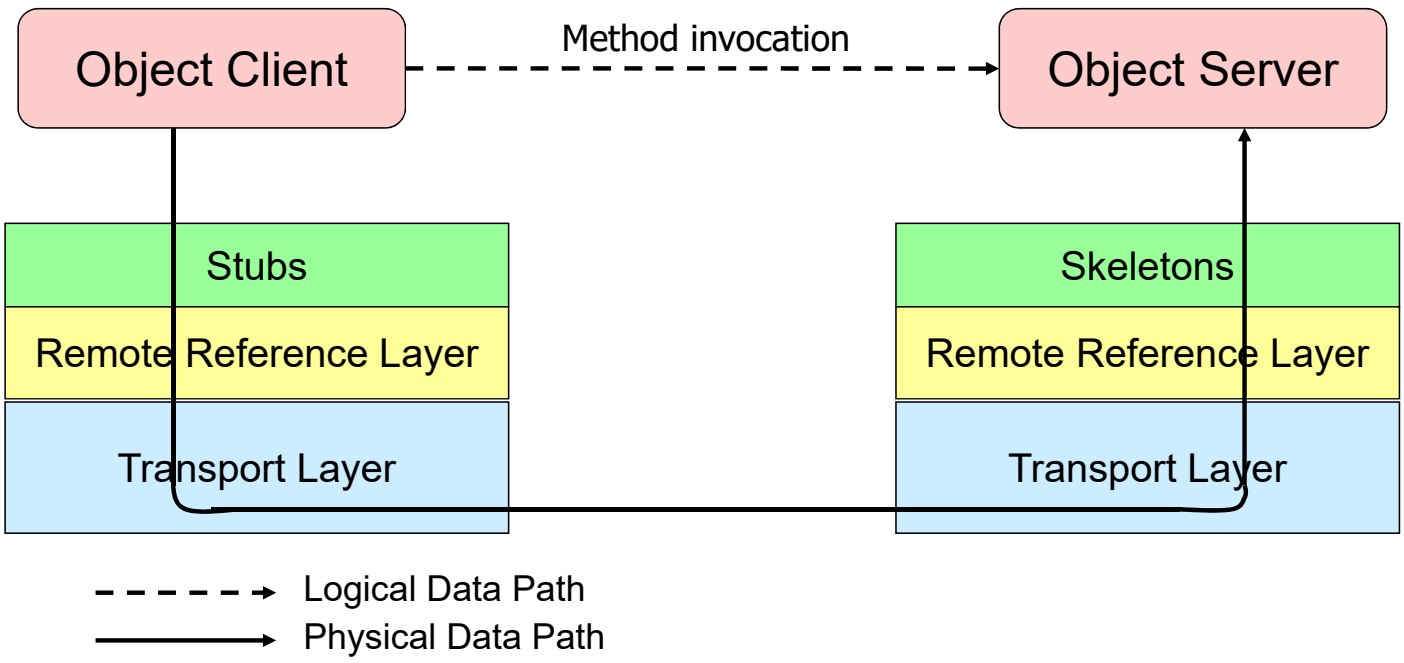
Anno accademico 2016/2017

Agenda

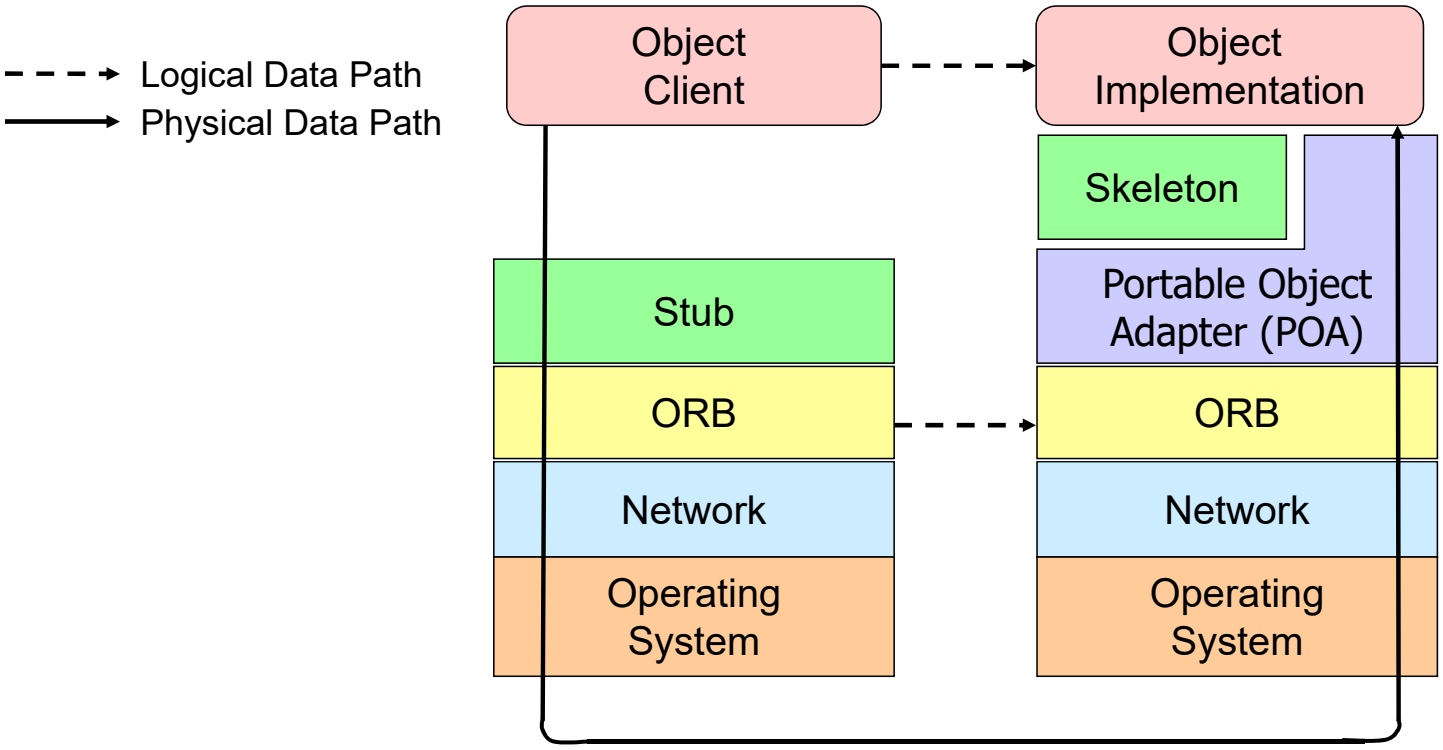
- Sviluppo di un'applicazione CORBA
  - Interfaccia, Client e Server
- Esempio completo di client statico
- Esempio completo di server statico
- Tool di sviluppo

Sviluppo di un'Applicazione CORBA

CORBA vs RMI: architettura RMI



CORBA vs RMI: architettura CORBA



ORB

Object **R**equest **B**roker è il **cuore** dell'architettura CORBA

Consente i collegamenti in modo **statico** e **dinamico** tra le entità **Object Request Broker** (**ORB**) per

- controllo **allocazione** e **visibilità** di oggetti
- controllo dei **metodi** e della **comunicazione**
- **gestione facilitata** di tutti i possibili servizi

Dal punto di vista realizzativo → possibilità diverse

- Implementazioni **locali** o **distribuite**
- A livello di **S.O.** o a **livello utente**

CORBA ADAPTER

Adattatore (Object Adapter) **componente di sistema** per superare **disomogeneità e differenze** tra le realizzazioni degli **ambienti di servizio** dei diversi servitori o servant

(NON per la presentazione dei dati)

OA dalla parte del **server**, con compiti tipici:

- funzionalità di ***registrazione*** dell'oggetto
- generazione dei ***riferimenti esterni*** all'oggetto
- ***attivazione*** dell'oggetto *e dei* ***processi interni***
- ***demultiplexing*** *delle richieste* in modo da disaccoppiarle
- ***invio delle richieste*** (upcall) agli oggetti registrati

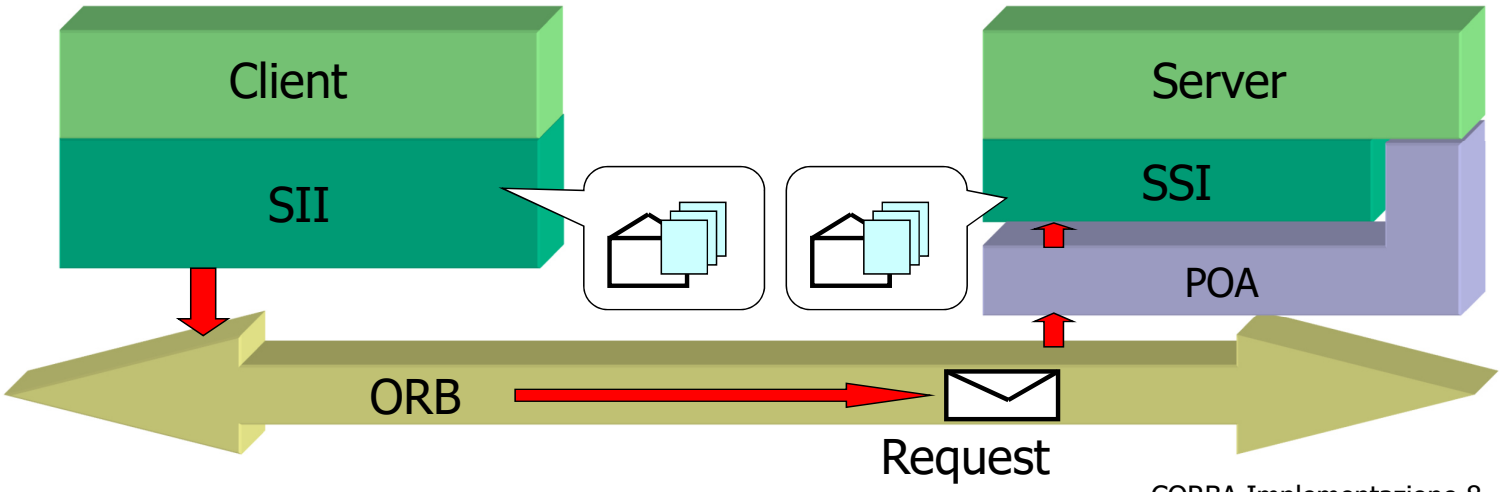
I primi adattatori erano Basic (**BOA**), poi Portable (**POA**)

OA non sono **OGGETTI** CORBA, ma **PSEUDO OGGETTI** CORBA

Invocazioni statiche

Static Invocation Interface (SII) e Static Skeleton Interface (SSI)

- Creazione e consumo delle richieste (Request lato client e ServerRequest lato server) gestite da **stub** e **skeleton**
  non **PSEUDO OGGETTI** CORBA
  ma **proxy di supporto locali**



Caratteristiche principali invocazioni statiche

- Comunicazione mediata da stub e skeleton
  - Lato client: Static Invocation Interface (SII)
  - Lato server: Static Skeleton Interface (SSI)
- Richiede la disponibilità dell'**interfaccia** (specificata con **Interface Definition Language – IDL**) del servizio in fase di programmazione/compilazione
- Usata nella **maggioranza delle applicazioni**
- *Stub* e *Skeleton* generati automaticamente dal compilatore IDL ciascuno nel suo linguaggio
- Facile utilizzo
- In generale, CORBA supporta ***DIVERSI LINGUAGGI*** (C, C++, Java, Cobol, …)

In questo corso utilizzeremo come linguaggio Java e come ORB e tool di sviluppo l'ORB open source JacORB

JacORB

- Strumento opensource *pure Java* che supporta in modo ampio lo standard CORBA; elenco completo delle feature alla pagina: `http://www.jacorb.org/features.html`
- Link per il download: `http://www.jacorb.org/download.html`
- Si consiglia di scaricare ***l'ultima versione stabile*** e il ***Full Source Code***
- Installazione ben documentata, basta seguire nella pagina JACORB_HOME\doc\install.html
- Uso dello strumento `ant` per la compilazione

- Attenzione: come detto nella pagina di installazione, per il corretto funzionamento del sistema è necessario ***rinominare*** il file di `configurazione jacorb_properties.template` in `jacorb.properties`, e ***modificarlo*** compatibilmente con la propria configurazione locale

Uso di CORBA con JacORB

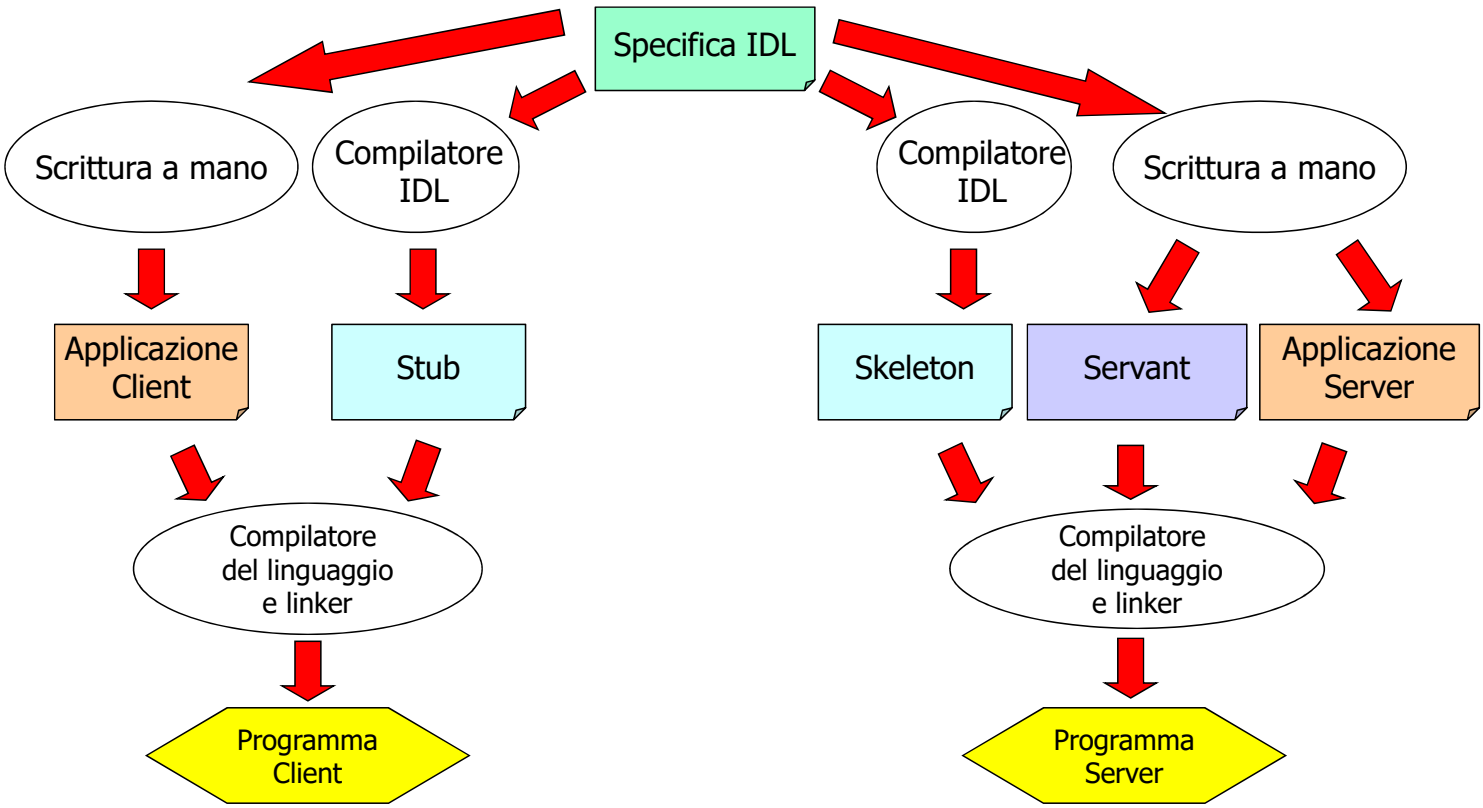
Passi di sviluppo simili a quelli visti per Java RMI, è necessario:

1. **Definire** interfacce (con IDL)
2. **Compilare** le interfacce con compilatore IDL (con `idl`)
3. **Definire** l'implementazione del componente remoto (**Servant**)
4. **Pubblicare (lato Server) il riferimento remoto al server** ottenuto con l'invocazione di `object_to_string`, che restituisce un riferimento all'oggetto remoto come stringa
5. **Ottenere (lato Client) il riferimento al server** partendo dalla stringa pubblicata dal server (vedi punto 4.)
6. **Compilare** tutte le classi (con `javac`), sia scritte **da noi** (Servant, Server e Client), sia **generate in automatico** (stub, skeleton, e classi di supporto richieste da CORBA)
7. **Attivare prima** il Server, **poi** il Client

A questo punto l'interazione tra il cliente e il server può procedere

In realtà il tutto è leggermente più complesso, ma i dettagli li vediamo dopo

Passi di sviluppo di un'applicazione CORBA



Compilatore IDL e uso supporto JacORB

Compilatore IDL:

```
idl [flag]* file [file]*
```

- Si trova nella cartella JACORB_HOME/**bin**
- Lanciare il comando con opzione --help per vedere i diversi parametri

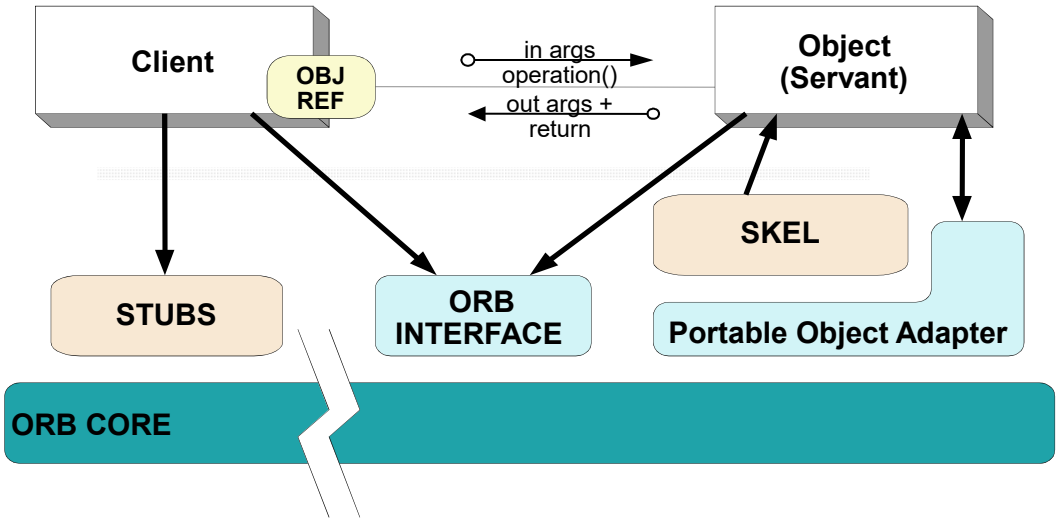
Esecuzione con JacORB:

Per utilizzare il supporto JacORB invece del supporto built-in per CORBA della SUN distribuito con la JDK, è necessario utilizzare le ***librerie JacORB***

L'esecuzione dei programmi (client e server) viene quindi facilitata dallo script **jaco** che invoca l'interprete java, ma utilizzando le librerie JacORB, ad esempio:

`jaco server` oppure **`jaco client`**

Una visione d'insieme



Per usare oggetti CORBA → necessari ***riferimenti (object reference)***

Creazione nuovi riferimenti remoti (attraverso ORB e POA)

- Registrazione di un nuovo oggetto: **`servant_to_reference`**

Ottenimento e passaggio riferimenti remoti (attraverso l'ORB)

- Per oggetti noti e di supporto: **`resolve_initial_references`**
- Come ***stringhe*** (passate da server a client in modi diversi: file system, mail, …): **`string_to_object`** e **`object_to_string`**

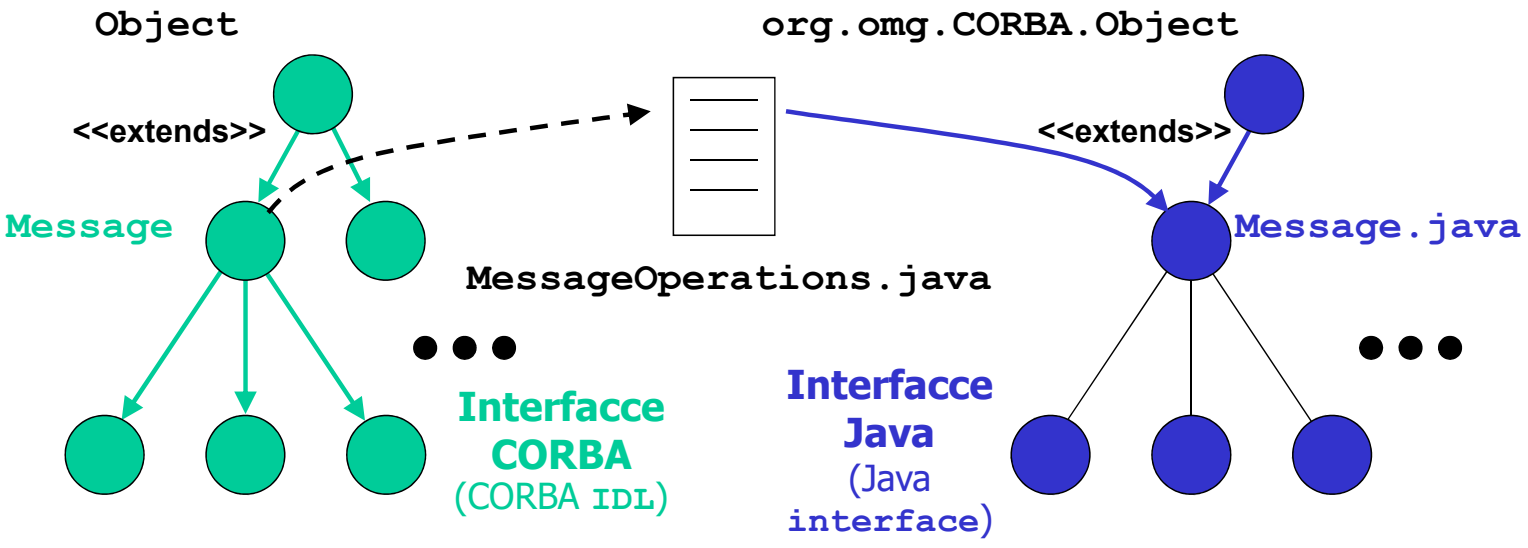
Servizio di echo remoto: interfaccia IDL

Definizione dell'interfaccia IDL del servizio di echo (**scritta da noi**)
→***INDIPENDENTE*** dal **linguaggio di implementazione**
→Tipi **primitivi** (string, boolean, char, …), **oggetti CORBA**, tipi **complessi** (composti a partire da tipi primitivi e oggetti CORBA, ad esempio uso di *typedef)*

```
/* message.idl */
interface Message
{
 /* Echo di una stringa
  * Un solo argomento di ingresso:
  * msg -> messaggio di cui vogliamo fare la echo, di tipo string
  *
  * Un solo risultato restituito come parametro di ritorno della
  * chiamata, di tipo string
  */
 string echo(in string msg);
};
```


Interfacce in CORBA e nei linguaggi

CORBA definisce **SOLO** interfacce → passaggio da interfacce ***CORBA*** a interfacce del linguaggio, come ad esempio, in ***Java:***



NON sono Object (classe) di Java, ma una **gerarchia** (di interface) a parte **che realizza i tipi di CORBA in Java**.

Servizio di echo remoto: interfacce Java

Generazione automatica delle interfacce Java col compilatore IDL di JacORB (programma **idl** nella directory $JACORB_HOME/bin)

```
>idl message.idl
```

```
/* MessageOperations.java
 * idl generated
 */
public interface MessageOperations
{
  String echo(String msg);
} // interface MessageOperations
```

Traduzione delle operazioni e attributi CORBA in metodi e variabili Java → **solo** mondo Java

```
/* Message.java - idl generated */
public interface Message extends
  MessageOperations,
  org.omg.CORBA.Object,
  org.omg.CORBA.portable.IDLEntity
{} // interface Message
```

Interfaccia **Message** è il punto di contatto fra il mondo **CORBA** e **Java**: estende **sia** le operazioni specificate nell'IDL CORBA, **sia** l'interfaccia del generico oggetto CORBA

Interoperabilità: casting e narrowing (1)

PROBLEMA: per abilitare ***l'interoperabilità fra linguaggi diversi*** (*tipati e non tipati*) i riferimenti remoti (CORBA) ricevuti dal client sono "opachi" e non object tipati
Ogni volta che operiamo con un *linguaggio tipato* vorremmo però ottenere (possibilmente) un ***riferimento tipato*** all'interfaccia dell'object (es. Message)

Questo problema ***NON c'è*** per ***ambienti omogenei***
Ad esempio, nel caso **RMI**: Java è un linguaggio tipato e supporta direttamente il ***casting*** → lo stub viene passato per valore (serializzazione), e lo stub restituito dall'operazione di lookup è già un oggetto ***"tipato"*** (Java)

Per ottenere il proxy locale (stub) all'oggetto remoto basta quindi effettuare ***il casting sul riferimento remoto*** recuperato

```
Message m = (Message)Naming.lookup(completeName);
```

Interoperabilità: narrowing (2)

Tipi CORBA sono più generali: per garantire ampia interoperabilità con i vari ambienti di sviluppo → **il riferimento** all'oggetto remoto (**Object**) contiene informazioni sul tipo dell'oggetto (identificatore unico di tipo CORBA), ma ***NON è*** un oggetto "tipato" per uno specifico linguaggio di implementazione:

```
org.omg.CORBA.Object obj =
   orb.string_to_object(objectReferenceString);
```

Nei linguaggi tipati si definisce ***narrowing*** l'esecuzione di due operazioni:

1. Il **controllo** che il riferimento opaco ottenuto sia del **tipo atteso** (individuando l'interfaccia specifica *nel grafo di ereditarietà CORBA* → da cui narrowing, cioè "restringere")
2. La **creazione** di uno stub locale all'oggetto come riferimento locale tipato *in termini del linguaggio di implementazione locale* (nel nostro caso Java). Lo **stub** offre localmente tutte le operazioni dell'interfaccia dello specifico oggetto remoto CORBA atteso (Message, e quindi anche MessageOperations):

```
Message message_stub = MessageHelper.narrow(obj);
```


Servizio di echo remoto: implementazione del client (1)

Il client, **scritto da noi**: (1) inizializza l'ORB, (2) ottiene il riferimento all'oggetto CORBA, (3) effettua il narrowing e (4) richiede i servizi all'oggetto remoto
In blu le operazioni CORBA standard, **NON dipendenti da Java**

```
/* client.java */
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class client
{
  public static void main(String[] args)
  {
    BufferedReader input = new BufferedReader
       (new InputStreamReader(System.in));
    try
    {
      // Ottenimento di un riferimento locale all'ORB
(1)  org.omg.CORBA.ORB orb =
          org.omg.CORBA.ORB.init((String[]) null, null);
```

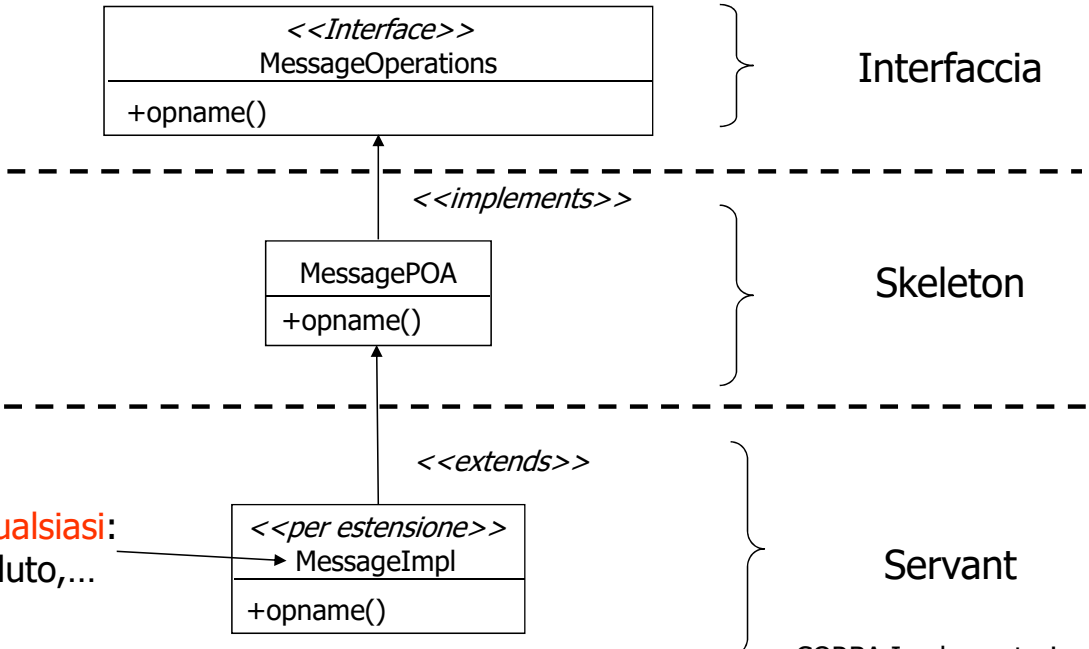
Servizio di echo remoto: implementazione del client (2)

```
      // Lettura del riferimento opaco all'oggetto remoto (passato come
      // argomento del programma)
(2)   org.omg.CORBA.Object obj = orb.string_to_object(args[0]);

      // Narrowing → per ottenere il riferimento allo stub
(3)   Message message_stub = MessageHelper.narrow(obj);

      // Logica di servizio
      System.out.print("Dammi la stringa di cui vuoi la echo:");
      String inString = input.readLine();
(4)   String result = message_stub.echo(inString);
      System.out.println("Ecco il risultato della echo: "+ result);
    }
    catch (Exception e){
      System.out.println("Eccezione, la seguente: " + e);
      e.printStackTrace();
    }
  } //main
}  //client
```


Implementazione servant in Java

Il **servant** (MessageImpl.java) realizza le funzionalità dell'oggetto CORBA a partire dallo skeleton (MessagePOA.java), nel caso più semplice la realizzazione del servant segue lo schema seguente

NOTA: uso ereditarietà semplice a livello di implementazione dei servant in Java imposta dal linguaggio (Java) NON da CORBA



Servizio di echo remoto: implementazione del servant in Java

Il servant MessageImpl, **scritto da noi**, realizza la logica applicativa **estendendo** la classe MessagePOA che realizza lo skeleton del servizio; MessagePOA viene generata **automaticamente insieme a tutte le altre classi di supporto**, invocando il compilatore IDL (>`idl message.idl`)

```
/* MessageImpl.java */
public class MessageImpl extends MessagePOA
{
  /**
   * Implementazione del servizio di echo
   */
  public String echo(String msg)
  { return msg; }
}
```

Classe **astratta** che implementa l'interfaccia MessageOperations; la realizzazione dei metodi è lasciata alle classi che la estendono
→ MessageImpl

Servizio di echo remoto: implementazione del server (1)

Il server, **scritto da noi**: (1) inizializza l'ORB, (2) crea il servant, (3) ottiene un riferimento al POA, (4) registra il servant, (5) attiva il POA e stampa il **riferimento remoto dell'oggetto** (come stringa) e (6) si mette in attesa delle richieste
In blu le operazioni CORBA standard, **NON dipendenti da Java**

```
/* server.java */
import org.omg.PortableServer.*;

public class server
{
  public static void main(String[] args)
  {
    try
    { //Inizializzazione ORB
(1)   org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init((String[])null,
                                                     null);

      // Creazione oggetto servant (new legata al linguaggio Java
      // NON a CORBA
(2)   MessageImpl servant = new MessageImpl();
```

Servizio di echo remoto: implementazione del server (2)

```
      // Ottengo un riferimento al POA di default per l'host locale
      POA poa = POAHelper.
(3)       narrow(orb.resolve_initial_references("RootPOA"));

      // Registro l'oggetto presso il POA, e ottengo il riferimento esterno
(4)   org.omg.CORBA.Object obj = poa.servant_to_reference(servant);

      // Attivazione POA -> da questo momento in avanti il POA
      // e' pronto a smistare le richieste ricevute al servant registrato
(5)   poa.the_POAManager().activate();

      // Stampo il riferimento esterno su standard output come stringa
      System.out.println(orb.object_to_string(obj));

      // Attivazione dell'orb -> aspetto più legato a Java
(6)   orb.run();
    } catch (Exception e)
    { System.out.println("Ho ricevuto questa eccezione: "+e);
      e.printStackTrace();
    }
  } //main
} //server
```


Servizio di echo remoto: compilazione ed esecuzione

- Compilazione **server**:

javac Message.java
MessageOperations.java
MessageHelper.java
MessageHolder.java
MessagePOA.java
MessageImpl.java
server.java

1) Esecuzione **server** (invocando **jaco**, nella directory JACORB_HOME/bin):

jaco server
IOR:0000104944…

1) Esecuzione **server** con passaggio IOR da file:

jaco server >ior.txt

- Compilazione **client**:

javac Message.java
MessageOperations.java
MessageHelper.java
MessageHolder.java
_MessageStub.java
client.java

2) Esecuzione **client**: (invocando **jaco**)

jaco client IOR:0000104944…

2) Esecuzione **client** con passaggio IOR da file:

jaco client `cat ior.txt`

Bibliografia

- S.Russo, C.Savy, D.Cotroneo, A.Sergio, “Introduzione a CORBA”, Ed. McGrow-Hill (2002)

- F. Bolton, “Pure CORBA – A Code-Intensive Premium Reference”, Ed. SAMS (2002)

- R.Orfali, D.Harkey, *“Client/Server Programming with Java and CORBA, 2nd ed.”*, Ed. Wiley (1998)

- JacORB Programming Guide: `JACORB_HOME/doc/ProgrammingGuide/ProgrammingGuide.pdf`