



Università degli Studi di Bologna
Facoltà di Ingegneria

Corso di Reti di Calcolatori M

CORBA - Implementazione Naming Service e Interface Repository

Luca Foschini

Anno accademico 2017/2018

Agenda

- CORBA Naming Service (NS)
 - Registrazione e lookup di un oggetto
 - Gestione Naming Context
- CORBA Interface Repository (IR)
 - Gestione interfacce e IR in CORBA
 - IR di JacORB

Naming Service

Servizi di nomi e CORBA

Finora abbiamo passato direttamente il **riferimento remoto** del server al client sotto forma di *stringa*

Solitamente però tutti i supporti di chiamata di procedura remota includono un **servizio di nomi**

- Java RMI: `rmiregistry`
- Sun RPC: `portmapper`

Anche CORBA include un **servizio di nomi** (il **Naming Service**) per registrare gli oggetti remoti CORBA, e recuperarne i **riferimenti**

JacORB Naming Service

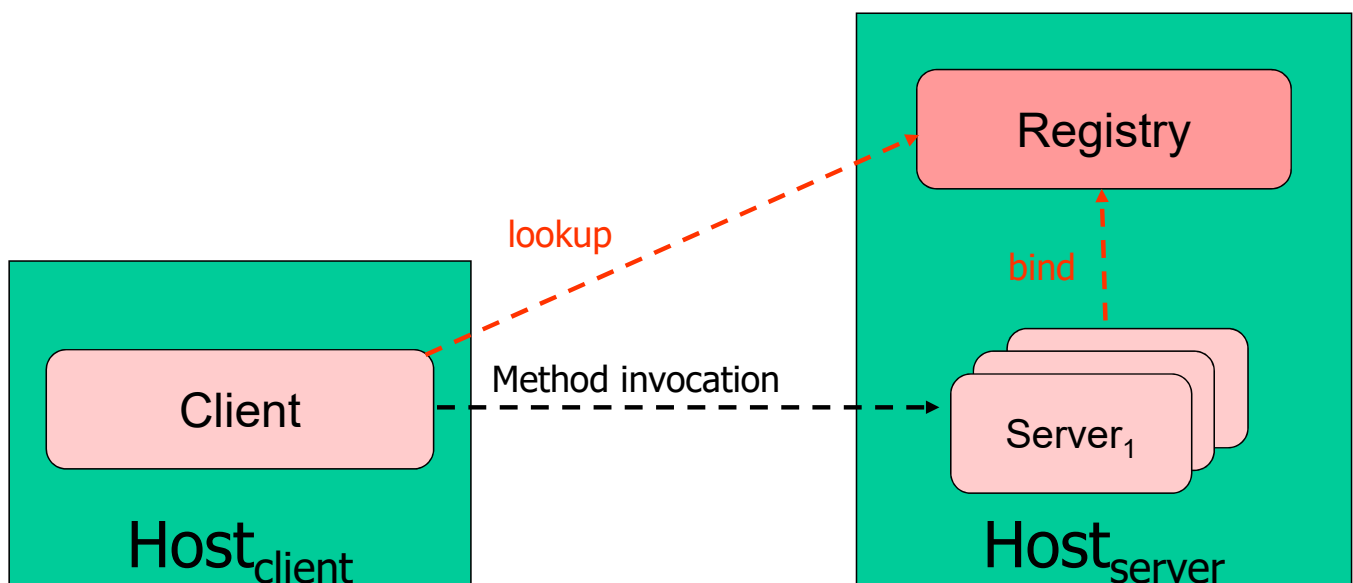
Naming Service:

ns

- Si trova nella cartella `JACORB_HOME/bin`
- Possibili argomenti:
 - `[-Djacorb.naming.ior filename=<filename>]` → per indicare **il file nel quale verrà scritto il riferimento remoto (IOR)** del Naming Service
 - `[-DOAPort=port]` → per indicare la porta di ascolto del Naming Service
- **Problemi:** si sono riscontrati alcuni **problemi nella risoluzione di nomi** che si riferiscono a name server distinti (ad esempio nel caso di *contesti federati*, vedi di seguito...)

CORBA NS e IR 5

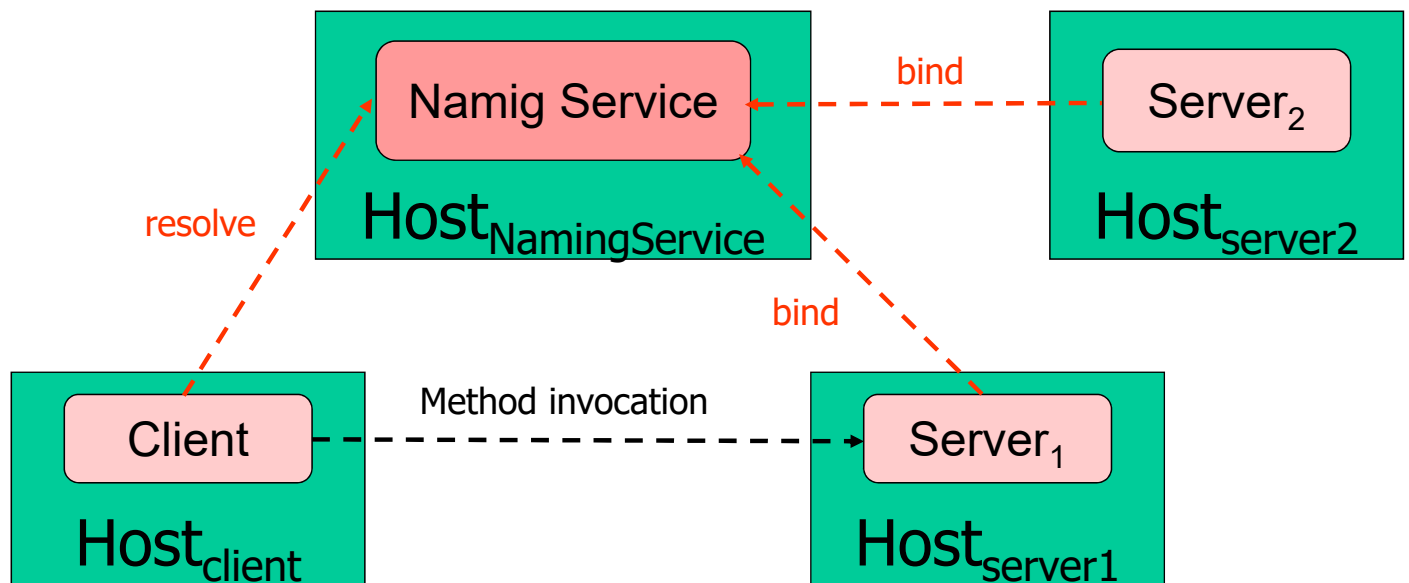
CORBA vs RMI: architettura RMI



- **NON** trasparente alla locazione
- Spazio di nomi **non strutturato**

CORBA NS e IR 6

CORBA vs RMI: architettura CORBA



- **Trasparente** alla locazione
- Inoltre... possibilità di ***strutturare lo spazio di nomi***

CORBA NS e IR 7

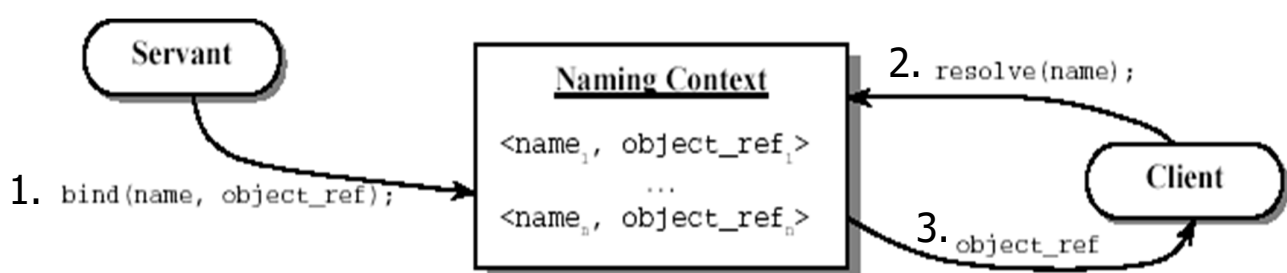
NAMING SERVICE

Consente di trattare *ObjectReference* in modo facile e per avere sempre alcuni sistemi di nomi noti

Name binding come associazione tra **oggetto** e **nome**

Name context come insieme di binding in cui ognuno dei nomi (delle coppie) è unico

I binding sono per definizione relativi ad un contesto specifico e da specificarsi



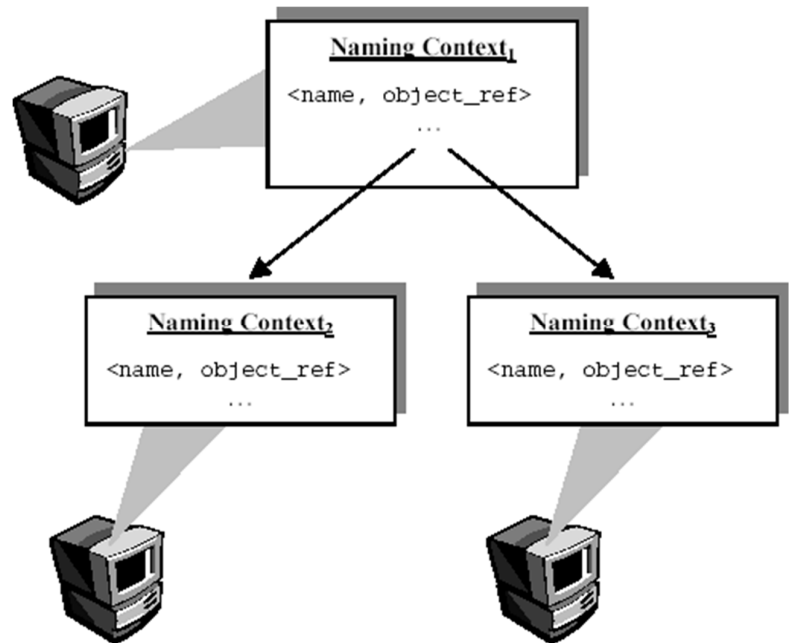
CORBA NS e IR 8

Strutturazione spazio di nomi e naming context

Un nome come una sequenza di componenti di nome

Nomi diversi possono fare riferimento a *oggetti diversi* o allo stesso *oggetto* ritrovandolo con un processo di **risoluzione**

I nomi possono anche fare riferimento a **contesti federati** con server diversi di gestione e connessi tra loro



CORBA NS e IR 9

NAMING SERVICE

Un nome (**Name**) come **semplice** o **composto** da una **sequenza** di componenti di nome (**NameComponent**)

Ogni componente costituito di due parti

[**Identifier** , **Kind**]

Identifier nome vero e proprio

Kind attributo descrittivo, ad esempio *executable*, *postscript*

```
struct NameComponent {string id; string kind;};
```

```
typedef sequence <NameComponent> Name ;
```

La idea è che il servizio (naming service) fornisca **solo meccanismi** e **non politiche** di nessun tipo

NAMING CONTEXT

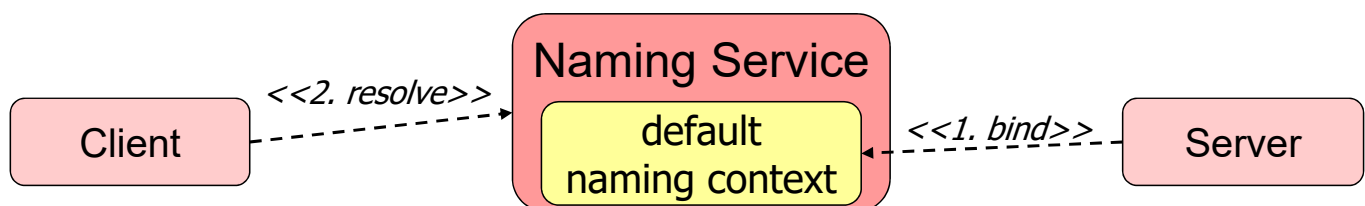
Le operazioni che si possono considerare su un contesto di naming derivano dalla interfaccia **NamingContext** che richiede le tipiche operazioni prevedibili per un sistema di nomi

```
interface NamingContext{  
    void bind(in Name n; in Object obj) raises ...;  
    void rebind(in Name n; in Object obj) raises ...;  
    void unbind(in Name n) ...;  
    void bind_new_context(in Name n)...;  
    object resolve(in Name n)...;  
    void list(in unsigned long how_many,  
        out BindingList bl, out BindingIterator bi);  
    ...}
```

È possibile registrare **un oggetto** con **nomi diversi**
Non è possibile usare lo **stesso nome (completo)** per registrare **oggetti diversi**

CORBA NS e IR 11

Registrazione di un oggetto presso il Naming Context di default



Passi di sviluppo

Server:

1. Creare il **servant**, registrarlo e ottenere dal POA locale il riferimento remoto
2. Ottenere dall'ORB il riferimento al **Naming Service di default**
3. Creare un **nome** per la **registrazione dell'oggetto**
4. Effettuare (presso il **name context di default**) il **re/bind** dell'oggetto

Client:

5. Ottenere dall'ORB il riferimento al **Naming Service di default**
6. Specificare un **nome** per la **risoluzione del riferimento dell'oggetto**
7. Invocare (presso il **Naming Service di default**) il **resolve** dell'oggetto

CORBA NS e IR 12

Registrazione di un oggetto presso il Naming Context di default: server

```
...
MessageImpl servant = new MessageImpl();
POA rootPOA = POAHelper.narrow(orb.
    resolve_initial_references("RootPOA"));
Object obj = rootPOA.servant_to_reference(servant);
// Risoluzione naming context di default
Object tmp_ref =
orb.resolve_initial_references("NameService");
NamingContextExt default_context =
    org.omg.CosNaming.NamingContextExtHelper.narrow(tmp_ref);
// Creazione nome
NameComponent[] tmp_name = default_context.to_name("Pluto");
/* Oppure:
 * NameComponent[] tmp_name = new NameComponent[1];
 * tmp_name[0] = new NameComponent("Pluto", "");
 */
default_context.rebind(tmp_name, obj); // Re/bind
...
```

1. 2. 3. 4.

CORBA NS e IR 13

Risoluzione di un oggetto presso il Naming Context di default: client

```
...
// Risoluzione naming context di default
Object tmp_ref =
    orb.resolve_initial_references("NameService");
NamingContextExt default_context =
    org.omg.CosNaming.NamingContextExtHelper.narrow(tmp_ref);

// Creazione nome
NameComponent[] tmp_name = default_context.to_name("Pluto");
/* Oppure:
 * NameComponent[] tmp_name = new NameComponent[1];
 * tmp_name[0] = new NameComponent("Pluto", "");
 */

// Resolve
Object obj = default_context.resolve(tmp_name);
...
```

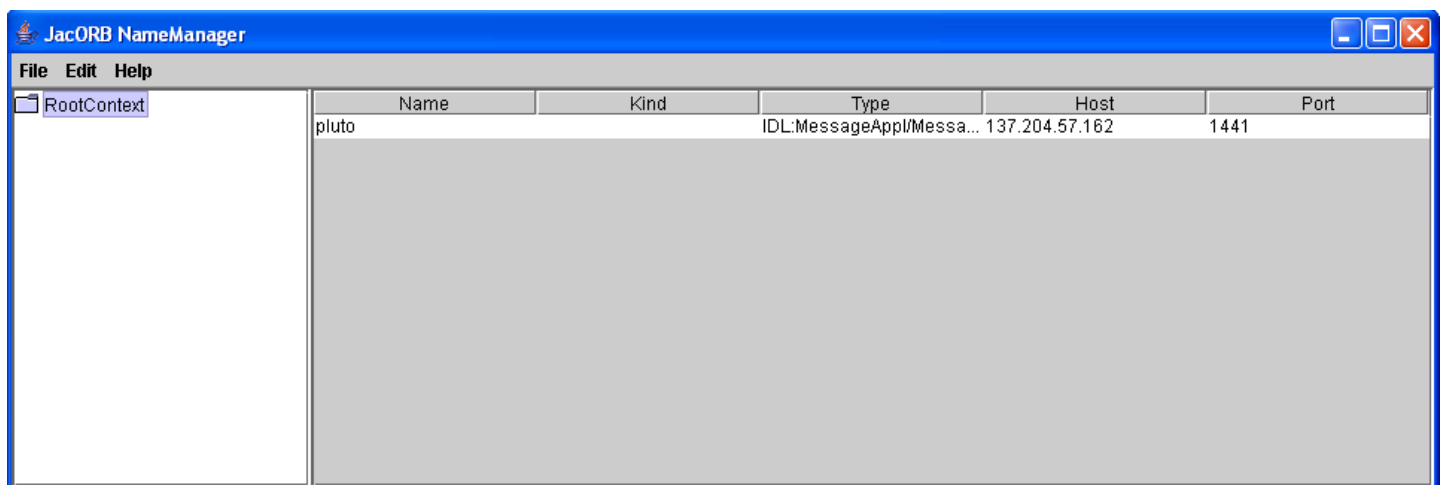
5. 6. 7.

Attivazione ed esecuzione con JacORB

- Naming Service
 - **Attivazione Naming Service di JacORB**; script di attivazione nella directory `JACORB_HOME/bin`: **ns**
 - Si veda la programming guide di JacORB per *impostazioni e settaggi specifici*, ad esempio, porta di ascolto, file salvataggio IOR, ...
- Server
 - Attivazione (e **registrazione**) server: **jaco server**
- Client
 - Attivazione (e **lookup**) client: **jaco client**
- Verifica e gestione NameService
 - **Tool grafico di JacORB** (`JACORB_HOME/bin`): **nmg**

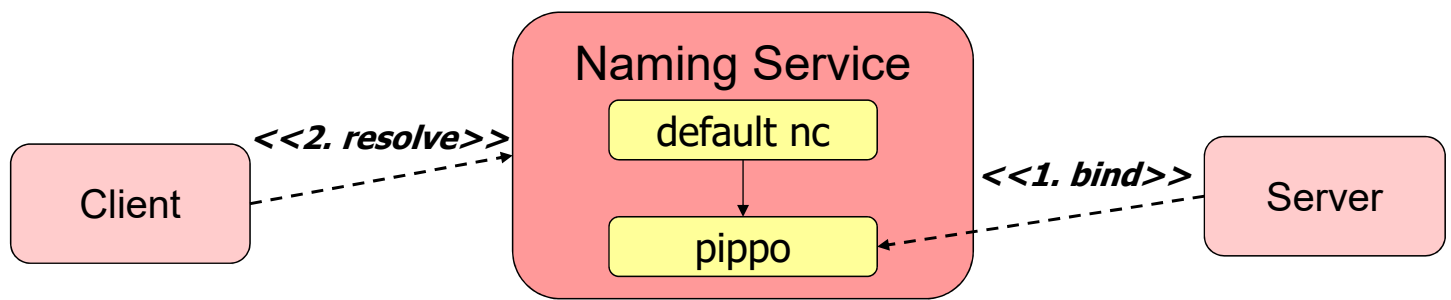
CORBA NS e IR 15

Tool grafico di JacORB: nmg



CORBA NS e IR 16

Creazione di un nuovo Naming Context



Server:

1. Recuperare il riferimento al Naming Service (**contesto di default**)
2. Creare il **nuovo contesto** a partire dal contesto di default
3. Creare un **nome** per la **registrazione dell'oggetto**
4. Effettuare (presso il NameService) il **re/bind** dell'oggetto

Client:

5. Recuperare il riferimento al Naming Service (**contesto di default**)
6. Specificare un **nome assoluto con tutti i naming context**, es. **"pippo/Pluto"**
7. Invocare (presso il NameService) il **resolve** dell'oggetto → naming service restituisce **direttamente** il riferimento eseguendo **in modo trasparente** interrogazioni iterative (se necessario)

CORBA NS e IR 17

Registrazione di un oggetto presso un nuovo Naming Context : server

```
...
// Risoluzione naming context di default
Object tmp_ref =
    orb.resolve_initial_references("NameService");
NamingContextExt default_context =
    org.omg.CosNaming.NamingContextExtHelper.narrow(tmp_ref);
// Se non esiste già, creo il nuovo contesto
NameComponent[] ctxName = default_context.to_name("pippo");
NamingContext ctx = null;
try{ctx=NamingContextExtHelper.
    narrow(default_context.resolve(ctxName)); }
catch(org.omg.CosNaming.NamingContextPackage.NotFound nf) {
    System.out.println("Binding new context: pippo");
    ctx = default_context.bind_new_context(ctxName); }
NameComponent[] tmp_name = new NameComponent[1];
tmp_name[0] = new NameComponent("Pluto", "");
ctx.rebind(tmp_name, obj); // Re/bind
```

1. {

2. }

3. }

4. ...

CORBA NS e IR 18

Registrazione di un oggetto presso un nuovo Naming Context: client

```
...
// Risoluzione naming context di default
Object tmp_ref =
orb.resolve_initial_references("NameService");
NamingContextExt default_context =
    org.omg.CosNaming.NamingContextExtHelper.narrow(tmp_ref); } 5.

// Creazione nome
NameComponent[] tmp_name =
    default_context.to_name("pippo/Pluto"); 6.

// Re/bind
Object obj = default_context.resolve(tmp_name); 7.
...
```

CORBA NS e IR 19

NS: DEPLOYMENT e PERSISTENZA

- Deployment
 - **Solitamente un ns per ogni località** (ad esempio per ogni dipartimento o LAN)
 - Possibilità di suddividere lo spazio dei nomi su **naming context diversi** sia all'interno **stessa macchina fisica**, sia su **server fisici diversi** (**contesti federati**)
 - L'architettura è comunque **molto libera** e **non impone vincoli**
- Persistenza del naming service
 - La specifica CORBA **non richiede** un **supporto persistente**
 - Molte implementazioni (anche JacORB) **mantengono un supporto persistente** (solitamente file locali) con le corrispondenze **nome:oggetto** per tutti i **contesti** e gli **oggetti** registrati
 - Possibili problemi di **inconsistenza** alla nuova attivazione (con creazione nuovo riferimento remoto) di un oggetto già registrato. Possibili soluzioni:
 - Al rebind il ns **prima** controlla se il server è **attivo** sul vecchio riferimento
 - Se non lo trova: sovrascrive registrazione precedente

CORBA NS e IR 20

ESERCIZI NAMING SERVICE

1. Si provi ad effettuare la **registrazione** dell'oggetto server e **lookup** del servizio su un Naming Service in remoto → **problema bootstrap:**
Come recuperare il riferimento ad un Naming Service non locale?

Suggerimenti:

- Il Naming Service di JacORB, al proprio avvio, scrive in un file locale **il proprio riferimento remoto** sotto forma di stringa (il nome del file viene indicato all'interno del file di proprietà `jacorb.properties`)
- Il riferimento può quindi essere copiato e utilizzato dai programmi server e client, al posto dell'invocazione del metodo `resolve_initial_references("NameService")` offerto dall'ORB per **invocare le operazioni del Naming Service**

2. Come stabilire, dato un **nome di oggetto** (ad es. "Pluto"), in **quale contesto l'oggetto è stato registrato?**

Attendiamo le vostre proposte di soluzione (***via email***)
per pubblicarle sul sito del corso

CORBA NS e IR 21

Interface Repository

Gestione Interfacce e Invocazioni in CORBA

CORBA supporta **due modalità** di invocazione

- **Static Invocation:** interfacce degli oggetti note staticamente a **tempo di compilazione**
 - Stub e Skeleton compilati automaticamente dal **compilatore idl**
- **Dynamic Invocation:** interfacce degli oggetti gestite (risolte, modificate, ...) dinamicamente; le interfacce devono essere note **solo al momento dell'uso**
 - **Gestione diretta** delle richieste al **livello applicativo**

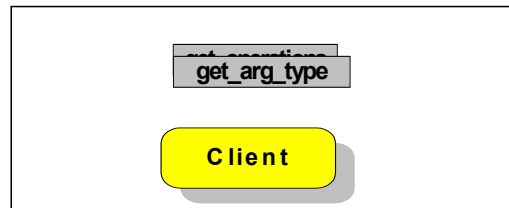
CORBA NS e IR 23

Dynamic Invocation e Gestione Interfacce: POSSIBILITÀ

- Informazioni di tipo (*interfacce*) **cablate nel supporto** (ORB + helper)
 - ☹ Approccio **statico**: esclude modifiche runtime dell'interfaccia
 - ☹ **Oneroso** in termini di risorse (in particolare **footprint dell'oggetto**, ma anche *dati scambiati* in rete) → **BASSO OVERHEAD**
 - 😊 **Consistenza** fra interfacce realizzate dai servant e interfacce definite nel file .idl
- Informazioni di tipo (*interfacce*) gestite da un **servizio esterno all'ORB** → è la scelta di CORBA: **Interface Repository (IR)**
 - 😊 Approccio **dinamico**: possibili modifiche a runtime
 - 😊 Più **leggero** (*footprint dell'oggetto* e *dati scambiati*) → **ALTO OVERHEAD**
 - ☹ Possibili **problemi di consistenza** fra interfacce definite nel file .idl e interfacce gestite dall'IR (si pensi a modifiche runtime...)

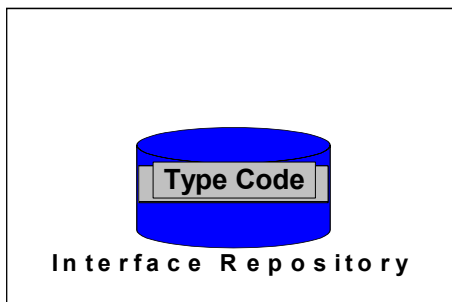
CORBA NS e IR 24

Esempio d'uso del IR



Alto overhead !!

Per recuperare le informazioni relative ad **una sola operazione** sono necessarie ***diverse invocazioni remote***

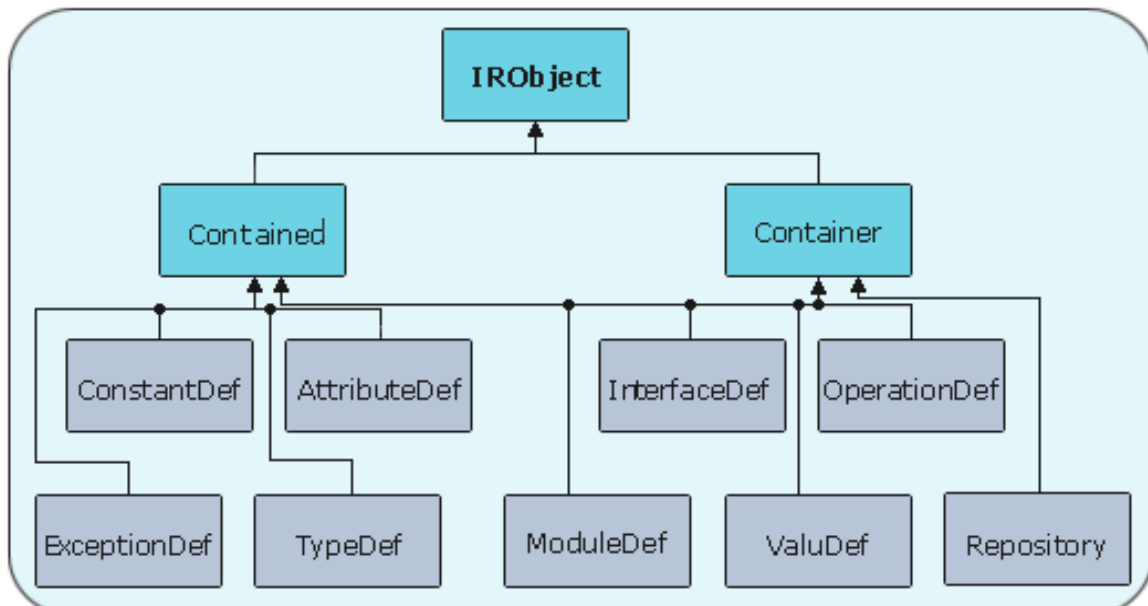


CORBA NS e IR 25

INTERFACE REPOSITORY in CORBA

Interface Repository si occupa di **registrare** tutte le **interfacce** e di gestirne la **memorizzazione** e la **ricerca**

Distingue tra **contenitore** e **contenuti** e richiede **interrogazioni iterative** dell'albero per risalire alle informazioni di interesse



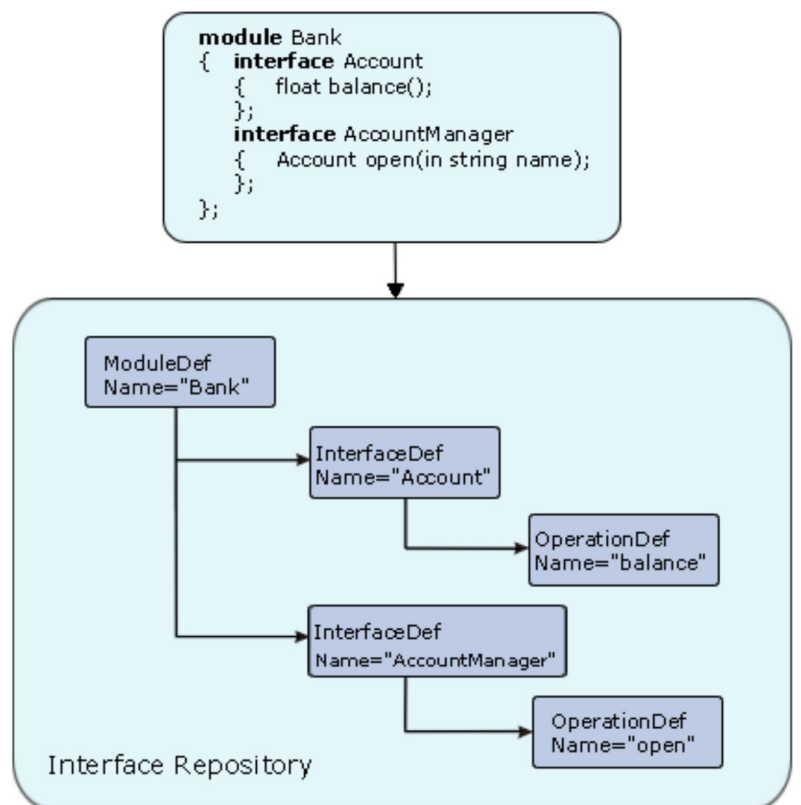
CORBA NS e IR 26

INTERFACE REPOSITORY in CORBA

Ad ogni interfaccia definita e compilata

Si generano delle trascrizioni delle informazioni nell'IR

in base ai tipi che possono essere riconosciuti



CORBA NS e IR 27

JacORB IR

Interface Repository:

ir

- Si trova nella cartella JACORB_HOME/bin
- Argomenti:
 - <class_dir> → per indicare **il direttorio** nel quale si trovano **i file .class**
 - <ior_file> → per indicare **il file nel quale verrà scritto il riferimento remoto (IOR)** dell'Interface Repository

CORBA NS e IR 28

INTERFACE REPOSITORY in CORBA

Interface Repository ad accesso

direttamente o attraverso **utilità proprietarie**

Ogni entità viene anche etichettata da un *RepositoryID*

JacORB usa il seguente formato standard di CORBA:

IDL Es.: IDL:/MessageAppl/Message:1.0

Si standardizzano (IDL CORBA) e raccomandano operazioni di accesso:

```
// interfaccia diretta a IR
```

```
Contained lookup_id(in RepositoryID searchid);
```

```
// interfaccia attraverso Object
```

```
InterfaceDef get_interface();
```

CORBA NS e IR 29

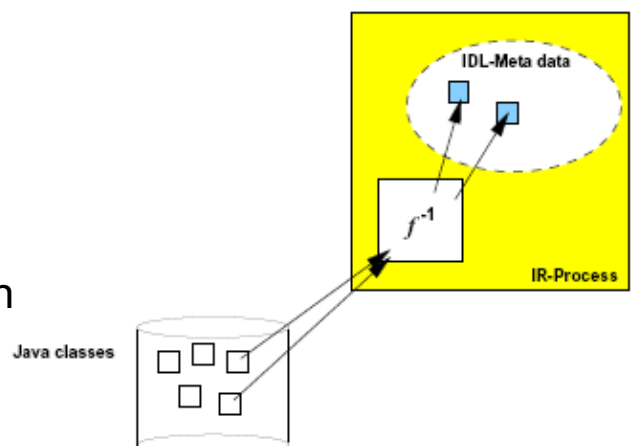
IR in JacORB: aggiornamento interfacce

Obiettivo: **evitare possibili inconsistenze** fra **interfacce** definite nel file **.idl** (e relative implementazioni) e **descrizioni** contenuti **nell'IR**

Scelta implementativa (**proprietaria**): uso di **reflection di Java**

Passi di sviluppo:

1. Scrittura IDL;
2. **Compilazione sorgenti** con compilatore `idl`
3. **Attivazione IR**: comando `ir` (directory `JACORB_HOME/bin`) con indicazione **directory** dove vengono **mantenuti i file .class** e **file salvataggio IOR** del IR



CORBA NS e IR 30

IR in JacORB: dettagli ulteriori

Il mapping da CORBA a Java **non è 1:1**; le **classi Java** generate in modo automatico **non contengono** abbastanza informazioni per **ricostruire** le informazioni contenute nel file **.idl originale**

- Ad esempio **attributi readonly**, oppure **IDL module** non necessariamente corrispondenti a package Java

Per la registrazione delle descrizioni delle interfacce è quindi necessario che il compilatore idl **generi ulteriori informazioni**

Scelta implementativa (e proprietaria) JacORB: rappresentare queste descrizioni come ulteriori file .java, per abilitare **uso reflection**

- Invocazione di **idl** con opzione **-ir**

CORBA NS e IR 31

ACCESSO a IR

- Partendo dal **riferimento all'oggetto remoto** (args[0])

```
Object obj = orb.string_to_object(args[0]);
org.omg.CORBA.Object tmp_interfaccia =
    obj._get_interface_def();

InterfaceDef interfaccia = // narrowing
    InterfaceDefHelper.narrow(tmp_interfaccia);
// Di qui in poi è possibile utilizzare le operazioni
// dell'oggetto remoto InterfaceDef
```

- Partendo dal **riferimento all'IR** (args[0])

```
Object tmp_ir = orb.string_to_object(args[0]);
org.omg.CORBA.Repository irep =
    org.omg.CORBA.RepositoryHelper.narrow(tmp_ir);
// Uso oggetto remoto Repository, ad esempio lookup_id

Contained cont =
    irep.lookup_id("IDL:MessageAppl/Message:1.0");
```

CORBA NS e IR 32

ACCESSO al IR

Tutti gli esempi sopra sono **funzionanti e supportati da JacORB** prestare solo molta **attenzione** in fase di compilazione

- Impostare opportunamente il CLASSPATH in modo da **usare le classi di JacORB** al posto di quelle della JDK di SUN

Ulteriori possibilità e **tool di supporto** di JacORB

- Tool grafico (directory **JACORB_HOME/bin**): **irbrowser**

Per la navigazione grafica dell'IR

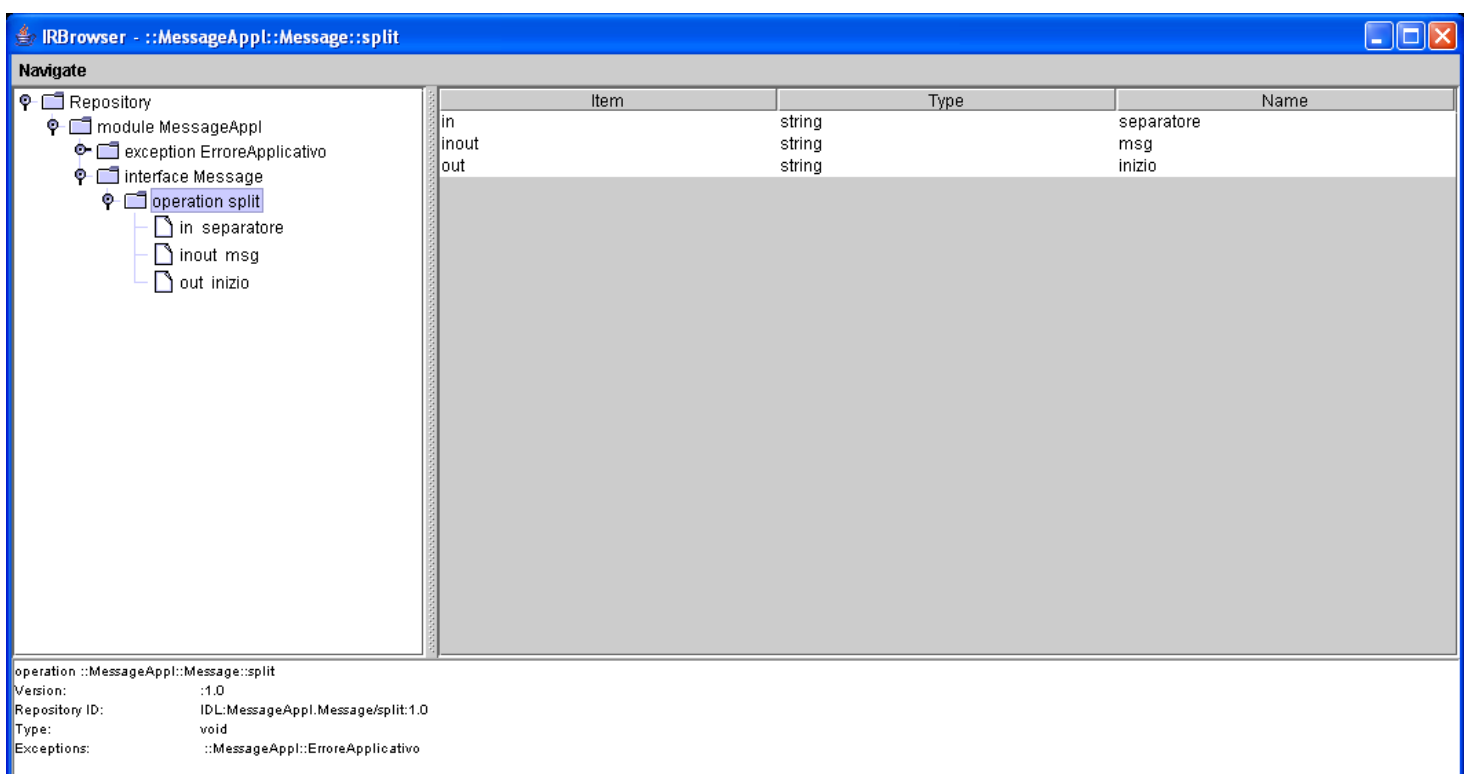
- Tool a riga di testo per l'interrogazione su di specifiche interfacce con nome già noto(directory **JACORB_HOME/bin**):

qir <nomeCompletoInterfaccia>

Es.: **qir IDL:MessageAppl/Message**

CORBA NS e IR 33

Tool grafico di JacORB: irbrowser



CORBA NS e IR 34

IR: DEPLOYMENT e PERSISTENZA

- Deployment
 - **Solitamente un ir per ogni località** (ad esempio per ogni dipartimento o LAN)
 - L'architettura è comunque **molto libera** e **non impone vincoli**
- Persistenza del naming service
 - La specifica CORBA **non richiede** un **supporto persistente**
 - IR di JacORB **non usa un supporto persistente**, ma **costruisce la base di dati al momento dell'attivazione** utilizzando **tecniche di riflessione Java** sui file .class
 - Si possono comunque verificare **problemi di consistenza** ad esempio **cambiando dinamicamente le interfacce** a tempo di esecuzione
 - Inconsistenze fra **classi ispezionate** inizialmente al **momento dell'attivazione** da IR e **modifiche IRObjeto successive**

Esercizio: si provino a **verificare operativamente tali problemi**
Attendiamo le vostre **proposte** (**via email**)