



Il modello CORBA

SCD

Anno accademico 2014/15
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@math.unipd.it

Laurea Magistrale in Informatica, Università di Padova1/35



Sistemi distribuiti: il modello CORBA

Architettura del modello – 2

- ❑ Modello cliente-servente a oggetti distribuiti
 - La realizzazione dell’oggetto risiede nello spazio di indirizzamento del suo processo servente
- ❑ Oggetti e servizi specificati in uno specifico *Interface Description Language*
 - CORBA IDL
- ❑ Precise regole di corrispondenza tra specifiche in CORBA IDL e la semantica equivalente in linguaggi di programmazione (C++, Java, Ada, ...)

Laurea Magistrale in Informatica, Università di Padova3/35



Sistemi distribuiti: il modello CORBA

Architettura del modello – 1

- ❑ Standard industriale promosso da OMG (*Object Management Group*) come modello di riferimento
 - <http://www.omg.org/gettingstarted/corbafaq.htm>

Oggetti delle applicazioni

Utilità verticali

Utilità orizzontali

Oggetti di servizi comuni

Object Request Broker

Laurea Magistrale in Informatica, Università di Padova2/35



Sistemi distribuiti: il modello CORBA

Architettura del modello – 3

Nodo del cliente

Applicazione cliente

Proxy di IDL statico (SII)

Interfaccia di invocazione dinamica (DII)

ORB di lato cliente

Sistema operativo locale

Nodo del servente

Realizzazione oggetto remoto

Skeleton di IDL statico

Skeleton di interfaccia dinamica

Portable Object adapter

ORB di lato servente

Sistema operativo locale

Rete di comunicazione

Architettura generale di un sistema CORBA

Laurea Magistrale in Informatica, Università di Padova4/35



Sistemi distribuiti: il modello CORBA

Architettura del modello – 4

❑ L'ORB è l'infrastruttura di comunicazione tra cliente e servente

○ Dal punto di vista dei processi applicativi l'ORB tratta riferimenti a oggetti (*object reference*)

○ Ciascuna istanza di ORB dovrà rendere questi riferimenti comprensibili ad altri ORB e processi residenti su nodi distinti

❑ Compito principale dell'ORB è localizzare i servizi disponibili a un processo cliente

○ Un sistema CORBA non è compilato come applicazione unica dunque non conosce a priori i nomi dei servizi disponibili

Laurea Magistrale in Informatica, Università di Padova

5/35



Sistemi distribuiti: il modello CORBA

Architettura del modello – 6

❑ Un *object adapter* fa da tramite tra l'ORB e l'oggetto remoto per le richieste in ingresso

○ L'*unmarshalling* delle invocazioni viene eseguito dallo *skeleton* dell'oggetto

❑ 2 tipi di *skeleton*

○ Statico → compilato

○ Dinamico → *skeleton* generico con realizzazione specifica del metodo `invoke` offerto al cliente

- Parte della realizzazione dell'oggetto distribuito come abbiamo già visto in relazione a Java RMI

Laurea Magistrale in Informatica, Università di Padova

7/35



Sistemi distribuiti: il modello CORBA

Architettura del modello – 5

❑ L'interfaccia tra *proxy* e ORB non deve essere necessariamente realizzata in forma standard

○ Essendo specificato in CORBA IDL può essere compilato in un linguaggio a scelta e quindi integrato nella realizzazione concreta del *proxy*

❑ Non tutti i *proxy* e le relative interfacce ORB possono essere realizzati/e staticamente

○ Un'applicazione può voler/dover determinare il servizio di interesse solo a tempo d'esecuzione e invocarlo dinamicamente

○ L'interfaccia offre un metodo `invoke` generico

Laurea Magistrale in Informatica, Università di Padova

6/35



Sistemi distribuiti: il modello CORBA

Architettura del modello – 7

❑ Anagrafe (*directory*) delle interfacce

○ Basato sulle definizioni IDL statiche con identificatore attribuito dal compilatore IDL

- Senza garanzie di unicità (!)

○ Operazioni standard di navigazione nel repertorio

- Uguali per ogni ORB

❑ Anagrafe delle implementazioni

○ Designa ciò che occorre per realizzare e attivare oggetti

○ Modalità strettamente legate alle specifiche istanze ORB

Laurea Magistrale in Informatica, Università di Padova

8/35

Unipd - SCD 2014/15 - Sistemi Concorrenti e Distribuiti



Sistemi distribuiti: il modello CORBA

Architettura del modello – 8

Servizi CORBA

Utilità orizzontali simili ai servizi di interesse generale offerti dal sistema operativo

Service	Description
Collection	For grouping objects into lists, queue, sets, etc.
Query	For querying collections of objects in a declarative manner
Concurrency	To allow concurrent access to shared objects
Transaction	Flat / nested transactions on method calls over multiple objects
Event	For asynchronous communication through events
Notification	For event-based asynchronous communication (with filtering)
Externalization	For marshaling and unmarshaling of objects
Life cycle	For creation, deletion, copying, and moving of objects
Licensing	For attaching a license to an object
Naming	For systemwide name of objects
Property	For associating (attribute, value) pairs with objects
Trading	To publish and find the services on object has to offer
Persistence	For persistently storing objects
Relationship	For expressing relationships between objects
Security	Mechanisms for secure channels, authorization, and auditing
Time	For obtaining current time within specified error margins

Laurea Magistrale in Informatica, Università di Padova

9/35



Sistemi distribuiti: il modello CORBA

Modalità di comunicazione – 2

❑ Richiesta sincrona differita

○ Asincrona rispetto all’invocazione

- Il chiamante procede

○ Sincrona rispetto alla risposta

- Il chiamante si sospende volontariamente in attesa

Request type	Failure semantics	Description
Synchronous	At-most-once	Caller blocks until response returned or exception raised
One-way	Best-effort	Caller continues immediately without waiting for any response from server
Deferred synchronous	At-most-once	Caller continues immediately but can later block until response delivered

Laurea Magistrale in Informatica, Università di Padova

11/35



Sistemi distribuiti: il modello CORBA

Modalità di comunicazione – 1

❑ Modello base: richiesta sincrona

○ L’interfaccia rappresenta all’esterno come “oggetto” qualunque sia la sua possibile realizzazione

○ Semantica *at-most-once*

- Per invocazioni che prevedono ritorno
- Garanzia fornita dall’infrastruttura CORBA
- Solleva eccezione nel chiamante in caso di problemi

❑ Richiesta asincrona (*one-way*)

○ Solo se non ha valori di ritorno

○ Semantica *best-effort* → senza garanzia di consegna

Laurea Magistrale in Informatica, Università di Padova

10/35



Sistemi distribuiti: il modello CORBA

Modalità di comunicazione – 3

❑ Comunicazioni disaccoppiate: notifica di eventi non persistenti

○ Rappresentati da un oggetto

○ Generati da un fornitore

○ Notificati (senza garanzia) tramite un canale eventi

○ Recepiti da ≥1 consumatore

Consumer

Consumer

Event channel

Supplier

Supplier

Supplier

Push event to consumers

Push (have-it)

Modello “pull”
il consumatore interroga il canale sulla presenza di eventi ed il canale interroga i suoi fornitori

Consumer

Consumer

Event channel

Supplier

Supplier

Supplier

Ask suppliers for new event

Pull (go-get-it)

Modello “push”
il fornitore invia l’evento al canale, che lo passa ai suoi consumatori

Laurea Magistrale in Informatica, Università di Padova

12/35



Sistemi distribuiti: il modello CORBA

Modalità di comunicazione – 4

❑ RMI e notifica di evento non conservano la richiesta fino alla consegna della risposta

○ Comunicazioni transitorie

❑ Il modello a code di messaggi permette comunicazioni persistenti

○ Adottato da CORBA in forma basata su oggetti

○ 2 stili di realizzazione, entrambi a carico del chiamante

- Modello a *call-back*
- Modello a *polling*
- Comportano accoppiamento tra chiamante e chiamato

Laurea Magistrale in Informatica, Università di Padova

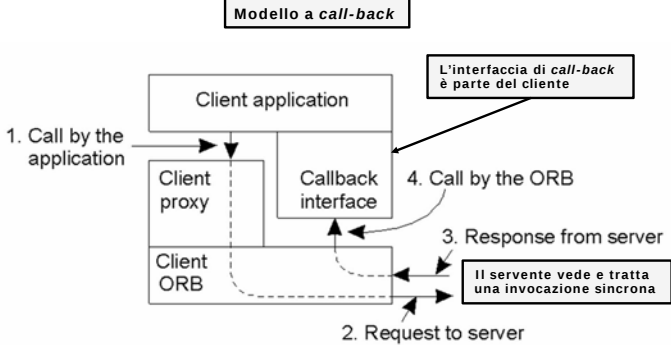
13/35



Sistemi distribuiti: il modello CORBA

Modalità di comunicazione – 6

Modello a *call-back*



Laurea Magistrale in Informatica, Università di Padova

15/35



Sistemi distribuiti: il modello CORBA

Modalità di comunicazione – 5

❑ Modello a *call-back*

○ Il cliente fornisce il riferimento a un proprio oggetto che realizza un'interfaccia di *call-back* cui trasmettere i risultati della richiesta

- La richiesta diventa **asincrona** per il chiamante
- La richiesta resta **sincrona** per il servente

○ L'interfaccia del cliente verso il servente viene sdoppiato

- Uno contiene i metodi invocabili dal cliente trasformati in modo che nessuno di essi contenga parametri di ritorno
 - Quasi come un normale *proxy*
- L'altro (aggiuntivo) contiene i metodi che l'ORB dovrà invocare per restituire il valore di ritorno prodotto dalle richieste del cliente
 - Interfaccia di *call-back*

Laurea Magistrale in Informatica, Università di Padova

14/35



Sistemi distribuiti: il modello CORBA

Modalità di comunicazione – 7

❑ Modello a *polling*

○ L'ORB fornisce un insieme di operazioni astratte che consentono al cliente di interrogarlo circa la presenza di risposte di ritorno

- L'invocazione sincrona viene così resa asincrona nella vista del cliente

○ Il cliente utilizza queste operazioni per la sua vista dell'interfaccia del servente

- Con una operazione invia la chiamata all'ORB richiedendo di trattenere la risposta fino a una futura interrogazione esplicita
 - Realizzata nel *proxy* del cliente
- Con l'altra il cliente interroga l'ORB per ottenere la risposta
 - Realizzata nell'ORB ma automaticamente a partire dall'IDL della richiesta

Laurea Magistrale in Informatica, Università di Padova

16/35

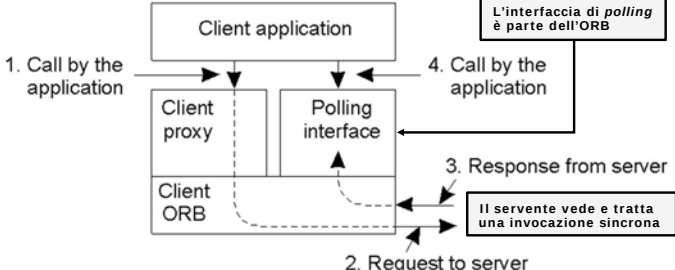
Unipd - SCD 2014/15 - Sistemi Concorrenti e Distribuiti



Sistemi distribuiti: il modello CORBA

Modalità di comunicazione – 8

Modello a *polling*



1. Call by the application

2. Request to server

3. Response from server

4. Call by the application

L'interfaccia di *polling* è parte dell'ORB

Il server vede e tratta una invocazione sincrona

Laurea Magistrale in Informatica, Università di Padova

17/35



Sistemi distribuiti: il modello CORBA

Interoperabilità tra ORB – 2

Tipo di messaggio	Per conto di	Descrizione
Request	Cliente	Il messaggio contiene una richiesta di invocazione
Reply	Servente	Il messaggio contiene la risposta ad una invocazione
LocateRequest	Cliente	Il messaggio contiene la richiesta di localizzazione di un particolare oggetto
LocateReply	Servente	Il messaggio contiene informazione sulla collocazione di un particolare oggetto
CancelRequest	Cliente	Il messaggio indica che il cliente non è più interessato ad una particolare richiesta
CloseConnection	Entrambi	Il messaggio indica che una particolare connessione verrà chiusa
MessageError	Entrambi	Il messaggio contiene informazioni di errore
Fragment	Entrambi	Il messaggio contiene parte di una comunicazione più ampia

Tipo di messaggi richiesti dalla specifica GIOP

Laurea Magistrale in Informatica, Università di Padova

19/35



Sistemi distribuiti: il modello CORBA

Interoperabilità tra ORB – 1

- ❑ L'infrastruttura CORBA si basa su un insieme di ORB eterogenei residenti sui nodi del sistema
 - CORBA specifica ciò che ciascun ORB deve fare ma non ne fornisce realizzazioni standard
- ❑ Un protocollo standard consente agli ORB di comunicare
 - GIOP (*General Inter-ORB Protocol*) ne è la specifica, che richiede *middleware* di comunicazioni affidabili
 - IIOP (*Internet Inter-ORB Protocol*) ne è realizzazione base che si poggia su TCP/IP
 - DIOP (*Datagram Inter-ORB Protocol*) specializza GIOP con semantica *asynchronous one-way* utilizzando UDP/IP

Laurea Magistrale in Informatica, Università di Padova

18/35



Sistemi distribuiti: il modello CORBA

Gestione dei nomi – 1

- ❑ Il riferimento agli oggetti CORBA deve essere **portabile** su diversi linguaggi di programmazione
- ❑ Il riferimento portabile è usato dall'ORB
 - Il cliente ne usa una versione specifica del linguaggio di programmazione
- ❑ La versione portabile è detta **Interoperable Object Reference (IOR)**

Laurea Magistrale in Informatica, Università di Padova

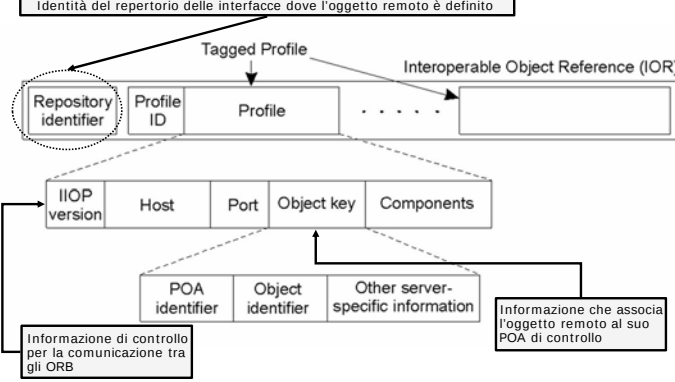
20/35



Sistemi distribuiti: il modello CORBA

Gestione dei nomi – 2

Identità del repertorio delle interfacce dove l'oggetto remoto è definito



Laurea Magistrale in Informatica, Università di Padova

21/35



Sistemi distribuiti: il modello CORBA

Lato cliente

- Dalla specifica IDL dell'oggetto remoto allo *stub* che reifica i messaggi Request e Reply
 - Generazione statica (SII) oppure dinamica (DII)
 - La modalità DII richiede l'acquisizione dello **IOR** dell'oggetto e della *signature* dei suoi metodi tramite interrogazione del repertorio delle interfacce
 - Il compilatore IDL genera un insieme di *file* sorgente da incorporare nel programma cliente
 - `idl2[linguaggio_specifico]` p.es.: `idlj, i[d1]ac`
- Lo *stub* connette il cliente con l'ORB del suo nodo per l'inoltro delle sue invocazioni

Laurea Magistrale in Informatica, Università di Padova

23/35



Sistemi distribuiti: il modello CORBA

Gestione dei nomi – 3

- Il campo `<Object key>` dello IOR riferisce direttamente l'oggetto remoto
 - Se disponibile, questa informazione consente *direct binding*
 - Il riferimento viene subito inviato al corrispondente processo servente (POA)
- L'informazione iniziale può essere invece limitata alla *directory* delle interfacce
 - In questo caso si ha *indirect binding* e il cliente deve prima interrogare il processo gestore della *directory* che esegue tutte le azioni necessarie per consentire e attivare la connessione tra cliente e servente

Laurea Magistrale in Informatica, Università di Padova

22/35



Sistemi distribuiti: il modello CORBA

Lato servente – 1

- *Servant*
 - La parte dell'oggetto che realizza i metodi remoti
 - Realizzato in uno specifico linguaggio di programmazione
 - Non portabile e non necessariamente un oggetto
- *Portable Object Adapter (POA)*
 - Componente che rende il codice di lato servente come un insieme di oggetti a disposizione di clienti distribuiti
 - La sua specifica lo rende portabile su ORB eterogenei
 - Rende il *servant* fruibile ai clienti creandone l'immagine di oggetto → `activate()` `activate_object_with_id (params)`

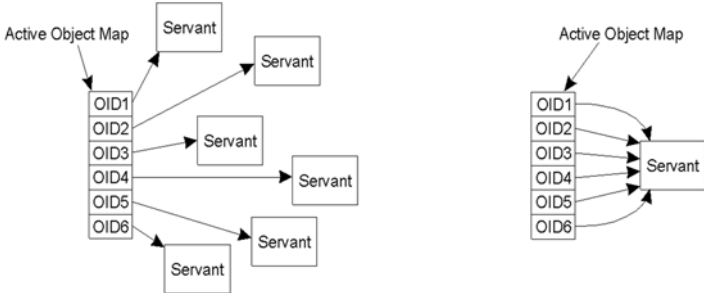
Laurea Magistrale in Informatica, Università di Padova

24/35



Sistemi distribuiti: il modello CORBA

POA



Un POA a supporto di molteplici *servant*

Un POA per un singolo *servant*

Traito da: Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall, Inc.

Laurea Magistrale in Informatica, Università di Padova

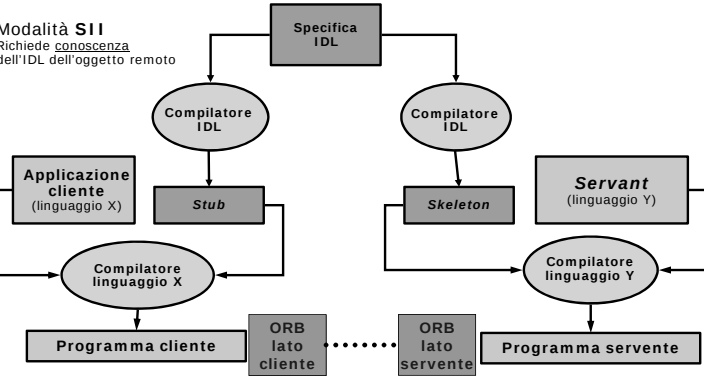
25/35



Sistemi distribuiti: il modello CORBA

Uso del modello

Modalità **SII**
Richiede conoscenza dell'IDL dell'oggetto remoto



Laurea Magistrale in Informatica, Università di Padova

27/35



Sistemi distribuiti: il modello CORBA

Lato servente – 2

❑ **Relazione tra classe *servant* e *skeleton***

- **Ereditarietà** → pura estensione di [nomeInterfaccia]POA
 - Il *servant* fornisce la realizzazione dei metodi astratti dell'interfaccia
- **Delega** → per intermediazione di una classe generata automaticamente dal compilatore IDL
 - La classe di intermediazione [nomeInterfaccia]POA*ie* estende lo *skeleton* dell'interfaccia semplicemente inoltrando al *servant* registrato presso di lui ogni invocazione dei metodi di interfaccia
 - Utile quando il *servant* sia estensione di altre classi specifiche dell'applicazione e il linguaggio di programmazione non consenta ereditarietà multipla
 - Creazione di istanza X di classe *servant*
 - Creazione di istanza di classe di intermediazione passando X come parametro

Laurea Magistrale in Informatica, Università di Padova

26/35



Sistemi distribuiti: il modello CORBA

Esempio – 1

Specifica di interfaccia in IDL

```
interface Message {
    string getMessage();
};
```

Compilazione IDL verso Java

```
idl2java -package msg Message.idl
```

Prodotto della compilazione nella cartella msg (vista parziale)

```
Message.java           // interfaccia Java corrispondente all'interfaccia IDL
MessageOperations.java // interfaccia Java corrispondente alle sole
                        // operazioni e attributi dell'interfaccia IDL
_Message_Stub.java     // la classe stub (proxy) di lato cliente
MessagePOA.java        // la classe skeleton di lato servente
MessageHelper.java     // classe di utilità per l'uso di Message
```

La classe Message.java (generata)

```
package message;
public interface Message extends
    org.omg.CORBA.Object,
    message.MessageOperations,
    org.omg.CORBA.portable.IDLEntity {
    string getMessage();
};
```

L'interfaccia Message, indipendentemente dalla sua realizzazione Java, sarà pubblicata e acceduta come un **oggetto CORBA**, con descrizione IDL portabile

Laurea Magistrale in Informatica, Università di Padova

28/35



Sistemi distribuiti: il modello CORBA

Esempio – 2 (lato cliente)

❑ Le azioni dell'applicazione cliente

1. Inizializzare dell'ORB sul proprio nodo traendone un riferimento locale e ottenendo la propria registrazione

2. Acquisire lo IOR del *servant* (oggetto remoto)

○ Pubblicazione dello IOR in formato stringa per utilizzo diretto

○ Acquisizione tramite servizio di *Naming* e *Trading*

3. Creare istanze della classe *stub* del *servant* identificato

○ Servizio di *narrowing* reso dalla classe *Helper* generata automaticamente dal compilatore IDL ed effettuato a partire dallo IOR dell'oggetto remoto

4. Invocare i metodi del *proxy* in un blocco try-catch per catturare e trattare eventuali errori

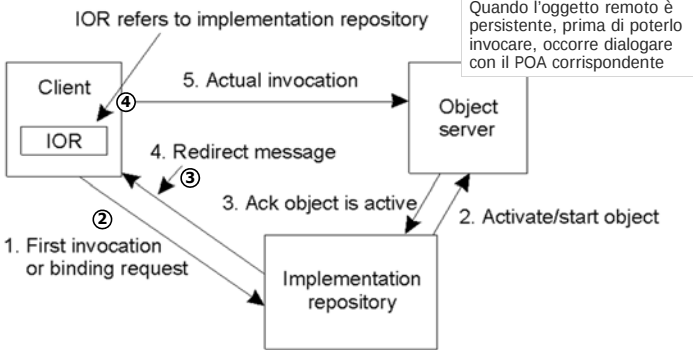
Laurea Magistrale in Informatica, Università di Padova

29/35



Sistemi distribuiti: il modello CORBA

Binding indiretto



IOR refers to implementation repository

Quando l'oggetto remoto è persistente, prima di poterlo invocare, occorre dialogare con il POA corrispondente

Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

Laurea Magistrale in Informatica, Università di Padova

31/35



Sistemi distribuiti: il modello CORBA

Esempio – 3 (lato cliente)

```
import message.*;
public class Cliente {
    public static void main(String[] args) {
        // attiva l'ORB sul nodo del cliente e lo registra
        ① org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args,null);
        // l'ORB trasforma lo IOR ricevuto da stdin in un riferimento
        // a un oggetto CORBA
        ② org.omg.CORBA.Object obj = orb.string_to_object (args[0]);
        // crea l'istanza dello stub dell'oggetto remoto
        // tramite servizio di narrowing reso dall'Helper di Message
        ③ message.Message mess_proxy = message.MessageHelper.narrow (obj);
        // invoca il servizio all'oggetto remoto
        ④ String messaggio = mess_proxy.getMessage ();
        System.out.println ("\\n Messaggio ricevuto dal servente: " + messaggio);
        System.out.println ("... Fine esecuzione.");
    }
}
```

La vista dell'oggetto distribuito presso il cliente è necessariamente quella del suo interfaccia nella rappresentazione del linguaggio di programmazione utilizzato

Laurea Magistrale in Informatica, Università di Padova

30/35



Sistemi distribuiti: il modello CORBA

Esempio – 4 (lato servente)

❑ Le azioni del processo servente

1. Inizializzare l'ORB sul proprio nodo traendone un riferimento locale e ottenendo la propria registrazione

2. Creare un'istanza del POA di nodo configurando le politiche di attivazione degli oggetti (p.es. persistenza)

○ Si può istituire un'intera gerarchia di POA a partire da una radice predefinita

3. Registrare il *servant* sul POA traendone il riferimento IOR all'oggetto CORBA corrispondente

4. Attivare il POA

5. Porre l'applicazione servente in attesa lanciando l'esecuzione della propria vista dell'ORB

Laurea Magistrale in Informatica, Università di Padova

32/35

UnIRD - SCD 2014/15 - Sistemi Concorrenti e Distribuiti



Sistemi distribuiti: il modello CORBA

Esempio – 5 (lato servente)

La classe **servant** realizza l'interfaccia astratta di MessagePOA ereditata da MessageOperations

```
import org.omg.PortableServer.*;
import message.*;
public class MessImpl extends MessagePOA { // per ereditarietà
    public String getMessage () { return ("Utilizzo di CORBA con successo!"); }
}
```

Il processo servente istanzia il **servant** lo registra come oggetto CORBA e si pone in attesa

```
import org.omg.PortableServer.*;
public class Server {
    public static void main (String[] args) {
        try {
            ① org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args,null);
            POA rootPOA = POAHelper.narrow (orb.resolve_initial_references ("RootPOA"));
            ② MessImpl myservant = new MessImpl ();
            ③ org.omg.CORBA.Object obj = rootPOA.servant_to_reference (myservant);
            ④ rootPOA.the_POAManager ().activate ();
            System.out.println (orb.object_to_string (obj)); // da IOR a stringa su stdout
            ⑤ orb.run ();
        } catch (Exception e) { e.printStackTrace (); }
    }
}
```

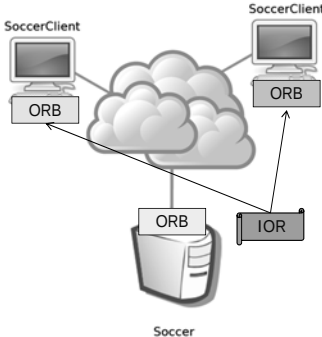
Laurea Magistrale in Informatica, Università di Padova

33/35



Sistemi distribuiti: il modello CORBA

Architetture: progetto calcio



- ❑ **Unico interfaccia di lato *server***
- ❑ **Il suo IOR viene pubblicato all'avvio e reso noto «per altra via» agli altri partecipanti**
- ❑ **In questo modo il lato *client* non ha bisogno di interrogare il NS**

```
module Monitor {
    typedef float BallCoordinates; Type[2];
    typedef float PlayerCoordinates; Type[2][1][2];
    typedef float GameStatus; Type[3];
    interface ClientInterface {
        boolean Subscriber (in string id);
        boolean Unsubscriber (in string id);
        BallCoordinates_Type GetBallPosition();
        PlayerCoordinates_Type GetPlayerPosition();
    };
    interface ServerInterface {
        void PublishGameStatus(in GameStatus_Type GameStatus);
    };
};
```

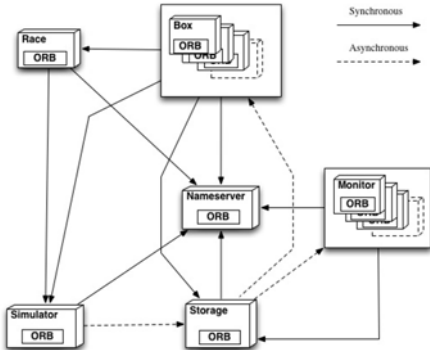
Laurea Magistrale in Informatica, Università di Padova

35/35



Sistemi distribuiti: il modello CORBA

Architetture: progetto F1



- ❑ **Name server unico «noto all'origine»**
- ❑ **Lo IOR degli oggetti remoti viene assegnato dall'ORB di nodo e poi passato al NS**
- ❑ **Le corrispondenti interfacce sono raccolte in moduli distinti**

Laurea Magistrale in Informatica, Università di Padova

34/35