Il modello CORBA

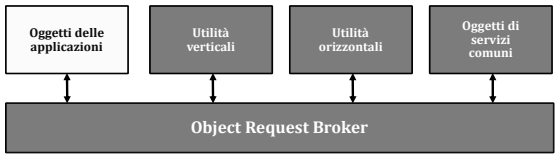
SCD

Anno accademico 2017/18
Sistemi Concorrenti e Distribuiti

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

Architettura del modello – 1

- **Standard industriale dei primi anni '90 promosso da OMG (*Object Management Group*) come modello di riferimento**
  - **http://www.omg.org/gettingstarted/corbafaq.htm**

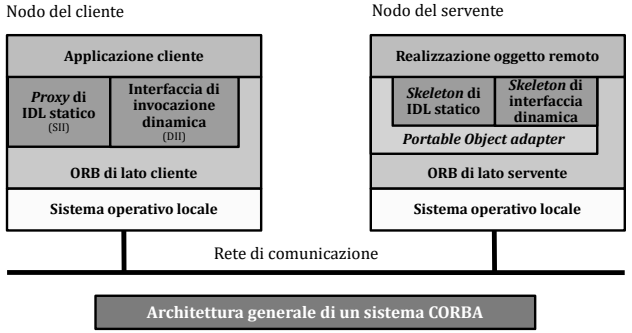


CORBA = Common Object Request Broker Architecture

Architettura del modello – 2

- **Modello cliente-servente a oggetti distribuiti**
  - **La realizzazione dell'interfaccia esposta in remoto risiede nello spazio di indirizzamento del suo processo servente**
- **Per fini di interoperabilità, oggetti e servizi specificati in uno specifico**
  - ***Interface Description Language***
- **Precise regole di corrispondenza tra specifiche in CORBA IDL e la semantica equivalente in linguaggi di programmazione (C++, Java, Ada, …)**

Architettura del modello – 3



Architettura generale di un sistema CORBA

Architettura del modello – 4

- L'Object Request Broker è l'infrastruttura di comunicazione tra cliente e servente
  - Dal punto di vista dei processi applicativi, l'ORB tratta riferimenti opachi a oggetti (*object reference*)
  - Ciascuna istanza di ORB di nodo li rende *inter-operable* per uso di altri ORB e processi residenti remoti
- 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

Architettura del modello – 5

- L'interfaccia tra *proxy* e ORB non deve essere necessariamente realizzata in forma standard
  - Essendo espressa in IDL può essere compilata in linguaggi a scelta e poi integrata nel sorgente 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 che espone a livello utente il servizio interposto tra *proxy*/*skeleton* e RMId

Architettura del modello – 6

- Un *object adapter* intermedia tra l'ORB e l'oggetto remoto per le richieste in ingresso
  - L'*unmarshalling* delle invocazioni viene eseguito dallo *skeleton* dell'oggetto
- CORBA prevede 2 tipi di *skeleton*
  - Statico → compilato
  - Dinamico → *skeleton* generico la cui istanza concretizza il metodo `invoke()` offerto al cliente
    - Analogo a quello che abbiamo già visto in relazione a Java RMI

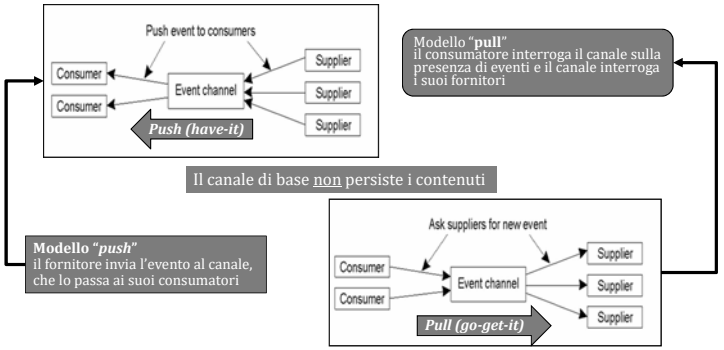
Architettura del modello – 7

- Anagrafe (*directory*) delle interfacce
  - Basato sulle definizioni IDL statiche con identificatore attribuito dal compilatore IDL
    - Senza garanzie di unicità (!)
  - Operazioni standard di navigazione nell'anagrafe
    - Uguali per ogni ORB
- Anagrafe delle implementazioni
  - Ciò che occorre per realizzare e attivare oggetti
  - Modalità legate alle specifiche istanze di ORB

Comunicazioni dirette

- **Richiesta sincrona (modello base)**
  - **Semantica *at-most-once***
    - Garanzia fornita dall'infrastruttura CORBA
    - Solleva eccezione nel chiamante in caso di problemi
- **Variante differita**
  - **Il chiamante può prima procedere e poi bloccarsi per attendere risposta**
- **Richiesta asincrona (*one-way*)**
  - **Solo in assenza di valori di ritorno**
  - **Semantica *best-effort***
    - Non trattiene copia della risposta fino a conferma (comunicazione transitoria)

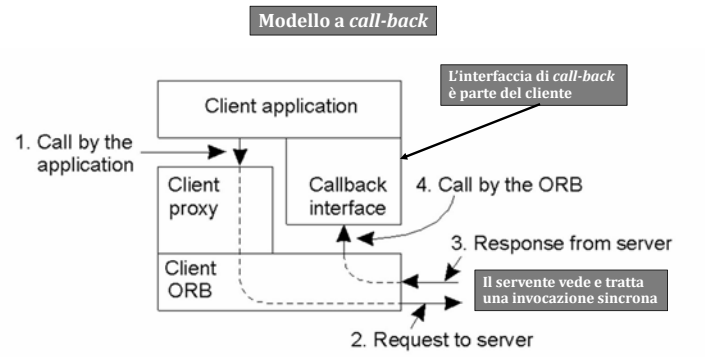
Comunicazioni indirette



Comunicazione a messaggi – 1

- **Modello a *call-back***
  - **Il cliente fornisce il riferimento a un suo interfaccia di *call-back* cui inviare i risultati**
    - In tal modo la richiesta diventa **asincrona** per il chiamante
    - Ma resta **sincrona** per il servente
  - **L'interfaccia del cliente verso il servente viene sdoppiato**
    - Uno contiene i metodi invocabili dal cliente (nel *proxy*) trasformati in modo che nessuno di essi contenga parametri di ritorno
    - L'altro contiene i metodi che l'ORB dovrà invocare per restituire il valore di ritorno prodotto dalle richieste del cliente

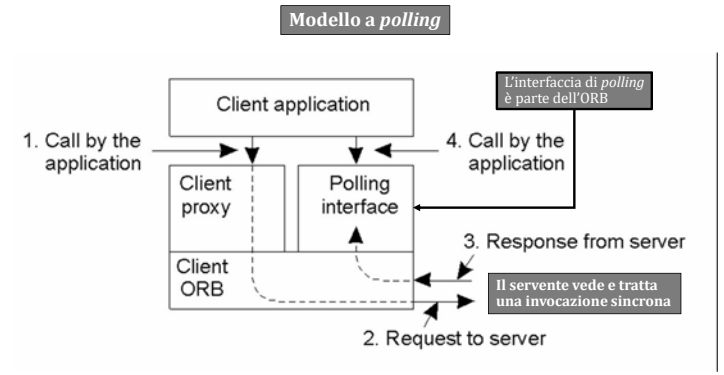
Comunicazione a messaggi – 2



Comunicazione a messaggi – 3

- **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
    - Con l'altra interroga l'ORB per ottenere la risposta

Comunicazione a messaggi – 4



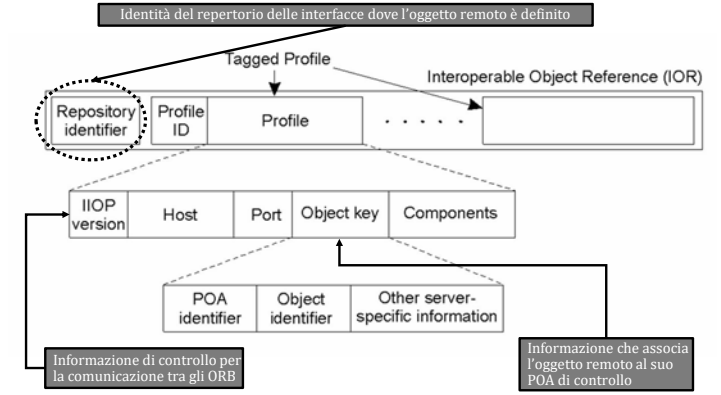
Interoperabilità tra ORB – 1

- **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**

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)**

Gestione dei nomi – 2



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 inviato al processo servente (POA)**
- **L'informazione iniziale può essere invece limitata al *registry***
  - **In questo caso si ha *indirect binding***
  - **Il cliente deve chiedere al gestore del *registry* di consentire e attivare la connessione con il servente**

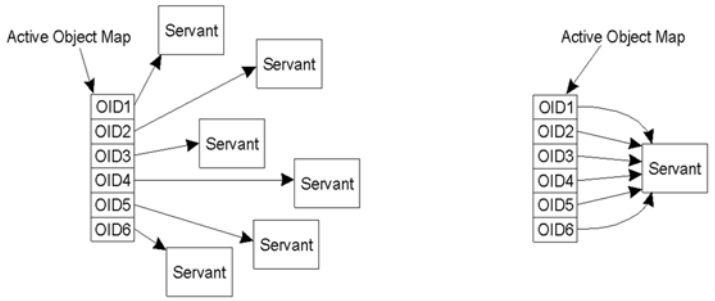
Lato cliente

- **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 interrogazione del *registry* per acquisire lo IOR dell'oggetto e la *signature* dei suoi metodi**
- **Il compilatore IDL genera un insieme di *file* sorgente da incorporare nel programma cliente**
  - `idl2[linguaggio_specifico] p.es.: idlj, i[dl]ac`
- **Lo *stub* connette il cliente con l'ORB del suo nodo per l'inoltro delle sue invocazioni**

Lato servente

- ***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 aggiuntivo che presenta l'implementazione del *servant* 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)`

POA



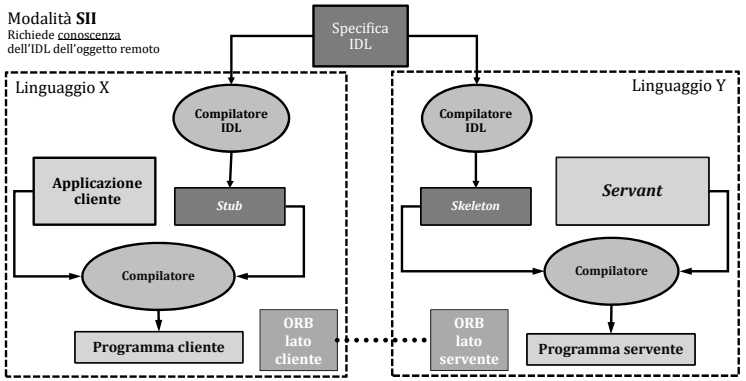
Un POA a supporto di molteplici *servant*

Un POA per un singolo *servant*

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

Uso del modello

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



Esempio – 1

Specifica di interfaccia in IDL

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

La classe Message.java (generata)

```
package message;
public interface Message extends
   org.omg.CORBA.Object,
   message.MessageOperations,
   org.omg.CORBA.portable.IDLEntity {
   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
```

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

Esempio – 2 (lato cliente)

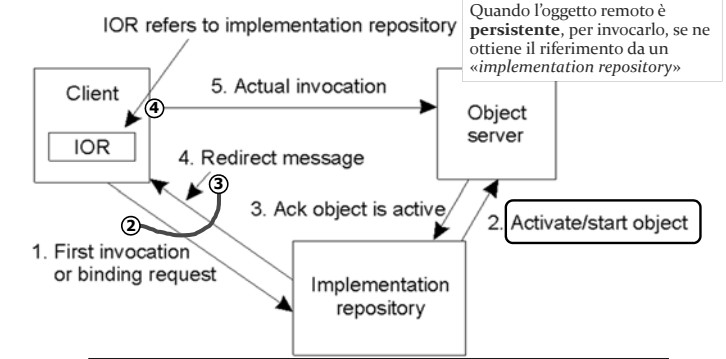
- **Le azioni dell'applicazione cliente**
  1. **Inizializzare dell'ORB sul proprio nodo traendone un riferimento locale e ottenendo la propria registrazione**
  2. **Acquisire lo IOR del *servant* (oggetto remoto)**
     - Pubblicazione dello IOR in formato stringa per utilizzo diretto
     - Acquisizione 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**

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
①org.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
②org.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
③message.Message mess_proxy = message.MessageHelper.narrow (obj);
  // invoca il servizio all'oggetto remoto
④String 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

Binding indiretto



Quando l'oggetto remoto è **persistente**, per invocarlo, se ne ottiene il riferimento da un «*implementation repository*»

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

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)**
     - Gli oggetti **transitori** sono disponibili solo finché il loro POA esiste
     - Per quelli **persistenti** serve passare attraverso un «***implementation repository***» per ottenere la loro disponibilità
  3. **Registrare il *servant* sul POA traendone il riferimento IOR all'oggetto CORBA corrispondente**
  4. **Attivare il POA**
  5. **Lanciare l'esecuzione della propria «vista» dell'ORB**

Esempio – 5 (lato servente)

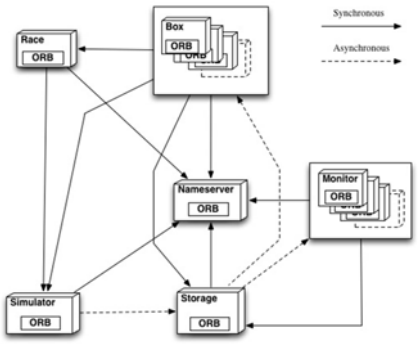
La classe ***servant*** realizza l'interfaccia astratta di MessagePOA ereditata da MessageOperations

```
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

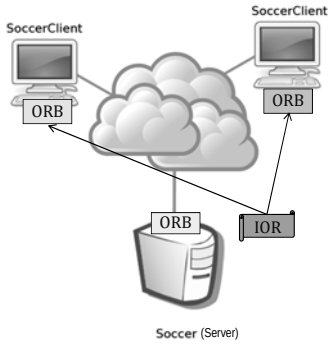
```
import org.omg.PortableServer.*;
public class Server {
 public static void main (String[] args) {
  try {
① org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args,null);
② POA rootPOA = POAHelper.narrow (orb.resolve_initial_references ("RootPOA"));
② MessImpl myservant = new MessImpl ();
③ org.omg.CORBA.Object 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 (); }
 }
}
```

Esempi d'uso – 1



- ***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**

Esempi d'uso – 2



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

```
module Monitor {
    typedef float BallCoordinates_Type[2];
    typedef float PlayersCoordinates_Type[2][11][2];
    typedef short GameStatus_Type[3];
    interface Client IoServer {
        boolean Subscribe( in string ior);
        boolean Unsubscribe( in string ior);
        BallCoordinates_Type GetBallPosition();
        PlayersCoordinates_Type GetPlayersPositions();
    };
    interface ServerToClient {
        void PublishGameStatus(in GameStatus_Type GameStatus);
    };
};
```