Università degli Studi di Padova

Il modello CORBA

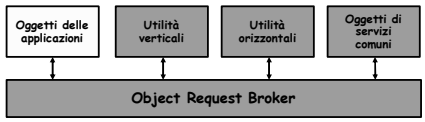
SCD

Anno accademico 2004/5
Corso di Sistemi Concorrenti e Distribuiti

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

Architettura del modello - 1

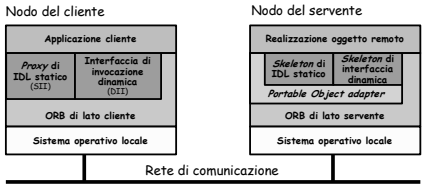
- **Standard industriale promosso da OMG, *Object Management Group*, come modello di riferimento soggetto a realizzazioni diverse**
  - **http://www.omg.org**



Architettura del modello - 2

- **Modello cliente-servente ad 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**
- **Regole precise di corrispondenza tra specifiche in CORBA IDL e l'espressione equivalente in linguaggi di programmazione (C++, Java, Ada, ...)**

Architettura del modello - 3



Architettura generale di un sistema 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 ad oggetti**
  - **Ciascuna istanza di ORB dovrà rendere questi riferimenti comprensibili ad ORB e processi residenti su nodi distinti**
- **Attività chiave dell'ORB è la localizzazione dei servizi disponibili ad un dato processo**
  - **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* ed ORB non deve essere realizzata in forma standard**
  - **Essendo specificato in CORBA IDL può essere compilato in un linguaggio a scelta e quindi facilmente integrato nella realizzazione del *proxy***
- **Non tutti i *proxy* e le relative interfacce ORB possono essere realizzati/e staticamente**
  - **Un'applicazione può localizzare a tempo d'esecuzione il servizio di interesse ed invocarlo dinamicamente**
  - **L'interfaccia offre un metodo `invoke` generico**

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 implementazione specifica del metodo `invoke` offerto al cliente, come parte della realizzazione dell'oggetto distribuito**

Architettura del modello - 7

- **Repertorio delle interfacce**
  - **Basato sulle definizioni IDL statiche, con identificatore attribuito dal compilatore IDL (dunque senza garanzie di unicità)**
  - **Operazioni standard di navigazione nel repertorio → uguali per ogni ORB**
- **Repertorio delle implementazioni**
  - **Designa ciò che occorre per realizzare ed attivare oggetti**
  - **Modalità strettamente legate alle specifiche istanze ORB**

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

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***
    - Utile quando l'invocazione deve restituire valori di ritorno → CORBA garantisce che il metodo sia stato invocato non più di una volta
    - 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**

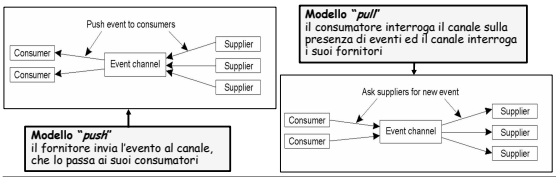
Modalità di comunicazione - 2

- **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 delivery*	*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*

Modalità di comunicazione - 3

- **Non solo RMI, ma anche notifica di eventi**
  - **Evento non persistente rappresentato da un oggetto, generato da un fornitore, recepito da un consumatore e notificato (senza garanzia) tramite un canale eventi**



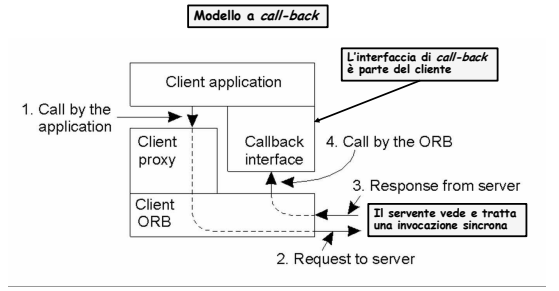
Modalità di comunicazione - 4

- **RMI e notifica di evento non conservano il messaggio fino all'avvenuta consegna**
  - **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***

Modalità di comunicazione - 5

- **Modello a *call-back***
  - **Il cliente fornisce il riferimento ad un (proprio) oggetto che realizza un interfaccia di *call-back* verso cui trasmettere i risultati della richiesta**
    - La richiesta diventa così asincrona per il chiamante
    - La richiesta resta sincrona per il servente
  - **L'interfaccia del cliente verso il servente viene sdoppiato**
    - Uno (quasi come un normale *proxy*) contiene i metodi invocabili dal cliente, trasformati in modo che nessuno di essi contenga parametri di ritorno
    - 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*

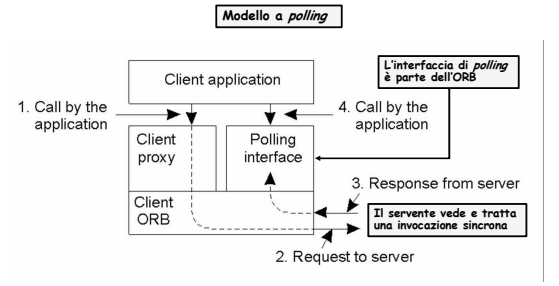
Modalità di comunicazione - 6



Modalità di comunicazione - 7

- **Modello a *polling***
  - **L'ORB fornisce un insieme di operazioni astratte che consentono al cliente di interrogarlo per la presenza di risposte a sue invocazioni**
    - L'invocazione sincrona viene così resa asincrona nella vista del cliente
  - **Il cliente utilizza queste operazioni per trasformare la sua vista dell'interfaccia del servente**
    - Una operazione invia la chiamata all'ORB notificandogli che questi dovrà trattenere la risposta fino ad esplicita interrogazione del cliente
      - Realizzata nel *proxy* del cliente
    - L'altra (duale) consente al cliente di interrogare l'ORB per la risposta alla sua corrispondente invocazione
      - Realizzata nell'ORB, ma automaticamente a partire dall'IDL della richiesta

Modalità di comunicazione - 8



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 implementazione adeguata, che si basa su TCP**

Interoperabilità tra ORB - 2

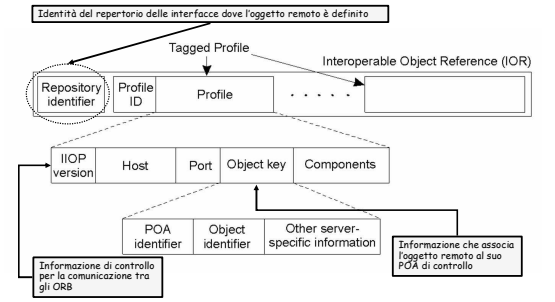
Tipo di messaggio	Per conto di	Descrizione
Request	Cliente	Il messaggio contiene una richiesta di invocazione
Reply	Servente	Il messaggio contiene la risposta ad una invocazione
LocateRequest	Cliente	Il messaggio contiene la richiesta di localizzazione di un particolare oggetto
LocateReply	Servente	Il 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

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* , dove il riferimento viene subito inviato al corrispondente processo servente (POA)**
- **L'informazione iniziale può essere invece limitata al repertorio delle interfacce**
  - **In questo caso, *indirect binding*, il cliente deve prima interrogare il processo gestore del repertorio, che esegue tutte le azioni necessarie per consentire ed attivare la connessione tra cliente e servente**

Lato cliente

- **Dalla specifica IDL dell'oggetto distribuito ad uno *stub* che implementa messaggi `Request` e `Reply`**
  - **Generazione statica (SII) oppure dinamica (DII)**
    - La modalità DII richiede un complesso processo di 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, alcuni dei quali da incorporare nel programma cliente**
    - `idl2`[*linguaggio_specifico*] p.es.: `id2java` , `idl2ada`
- **Compito dello *stub* (*proxy*) è connettere il cliente con l'ORB del suo nodo per l'inoltro delle sue invocazioni**

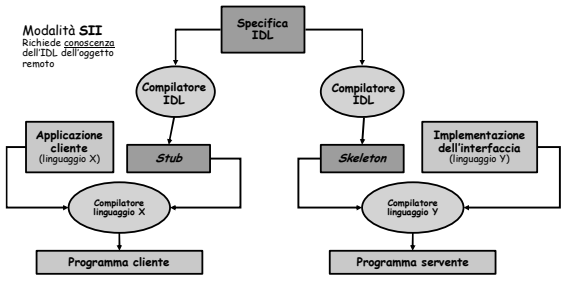
Lato servente - 1

- ***Servant***
  - **La parte dell'oggetto che realizza i metodi invocabili da clienti distribuiti**
  - **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)`

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]`**POATie** 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 ed 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

Uso del modello



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 ed 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 ed acceduta come un oggetto CORBA, con descrizione IDL portabile

Esempio – 2 (lato cliente)

```
import message.*;
public class Cliente {
  public static void main(String[] args) {
    // attivazione dell'ORB sul nodo del cliente e sua registrazione
1.  org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init (args,null);
    // risoluzione dello IOR ricevuto da stdin in un riferimento
    // ad un oggetto CORBA (servizio reso dall'ORB)
2.  org.omg.CORBA.Object obj = orb.string_to_object (args[0]);
    // creazione di istanza dello stub dell'oggetto distribuito
    // tramite servizio di narrowing reso dall'Helper di Message
3.  message.Message mess_proxy = message.MessageHelper.narrow (obj);
    // invocazione del 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

Esempio – 3 (lato cliente)

- **Le azioni dell'applicazione cliente**
  1. **Inizializzare dell'ORB sul proprio nodo, traendone un riferimento locale ed ottenendo la propria registrazione**
  2. **Acquisire lo IOR del *servant* (oggetto remoto)**
     - Varie modalità possibili
       - 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**

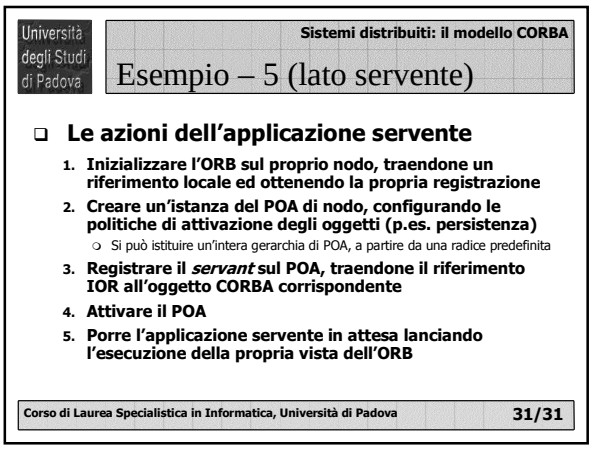
Esempio – 4 (lato servente)

Il ***servant*** che 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 servente che istanzia il ***servant***, lo registra come oggetto CORBA e si pone in attesa di richieste

```
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"));
    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 (); }
  }
}
```


Università degli Studi di Padova
Sistemi distribuiti: il modello CORBA
Esempio – 5 (lato servente)
Le azioni dell'applicazione servente
1. Inizializzare l'ORB sul proprio nodo, traendone un riferimento locale ed 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
Corso di Laurea Specialistica in Informatica, Università di Padova
31/31