



Università degli Studi di Bologna  
Facoltà di Ingegneria

## Corso di Reti di Calcolatori M

### *CORBA - Implementazione Politiche di Attivazione Lato Server*

Luca Foschini

Anno accademico 2014/2015

## Portable Object Adapter (POA)

---

Il POA, tra le sue responsabilità, ha la gestione del **ciclo di vita** degli oggetti CORBA

Una programmazione avanzata prevede la possibilità di impostare **strategie di gestione** degli oggetti, **configurando** (lato server) **il POA**:

- Gestione della **persistenza** degli object reference
- Gestione dell'**attivazione** degli oggetti
- Gestione dell'accesso degli oggetti serventi in caso di **richieste concorrenti**

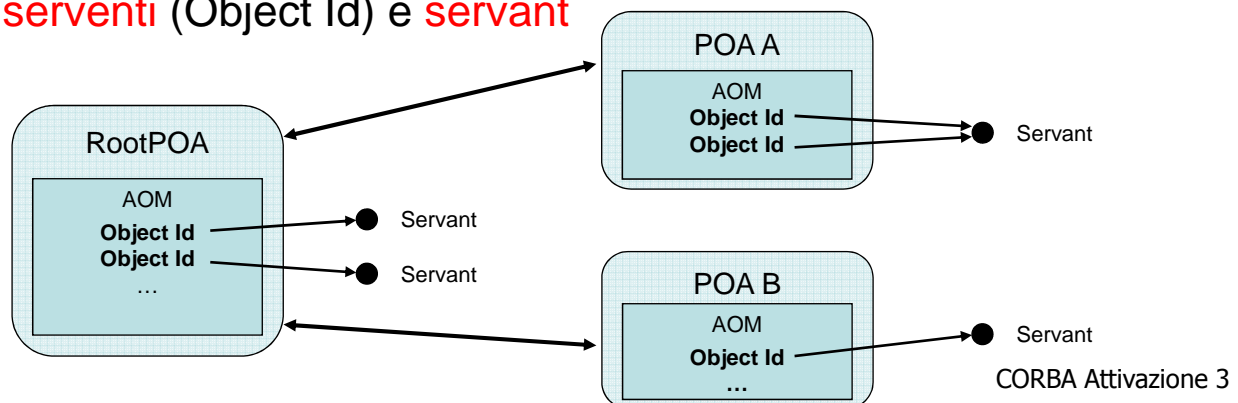
**NOTA:** **nessuna modifica** al codice degli oggetti serventi

# Organizzazione POA e Gestione Servant

La creazione di **più POA** (organizzati in una struttura ad albero) consente di suddividere gli **oggetti serventi** in **gruppi** (ciascuno con un proprio POA) e di operare **diverse politiche di gestione**

- **Una per ogni POA** → una per gruppo di servant

**Active Object Map (AOM)** → corrispondenza fra **oggetti serventi (Object Id)** e **servant**



## POA e Politiche di Gestione

**Diverse politiche** possibili (sottolineate quelle di default):

**Thread** (ORB\_CTRL\_MODEL, SINGLE\_THREAD\_MODEL)

**Lifespan** (TRANSIENT, PERSISTENT)

**Object ID Uniqueness** (UNIQUE\_ID, MULTIPLE\_ID)

**ID Assignment** (USER\_ID, SYSTEM\_ID)

**Servant Retention** (RETAIN, NON\_RETAIN)

**Request Processing** (USE\_ACTIVE\_OBJECT\_MAP\_ONLY,  
USE\_DEFAULT\_SERVANT,  
USE\_SERVANT\_MANAGER)

**Implicit Activation** (IMPLICIT\_ACTIVATION,  
NO\_IMPLICIT\_ACTIVATION)

# Retention e Request Processing Policy

- **Retention** policy: indica l'utilizzo o meno dell'AOM
  - **RETAIN**: memorizzazione di tutti gli Object Id nell'AOM
  - **NON\_RETAIN**: **NON** si usa **AOM** → uso di **Default Servant**, o di **Servant Manager**
- **Request Processing** policy: indica la modalità di reperimento degli oggetti serventi per l'elaborazione delle richieste

- 1 servant  
per più ogg.  
serventi
- 1 servant  
∇ ogg.  
servente
- **USE\_ACTIVE\_OBJECT\_MAP\_ONLY**: dispatching effettuato per **i soli oggetti serventi registrati** presso AOM
  - **USE\_SERVANT\_MANAGER**: politiche di **attivazione/disattivazione** dell'oggetto servente a carico del Servant Manager → gestite **direttamente dal programmatore**
  - **USE\_DEFAULT\_SERVANT**: le richieste destinate ad **oggetti serventi** non reperibili nel POA (se è impostata una politica **NON\_RETAIN**, oppure l'oggetto servente **non è nell'AOM**) sono destinate ad un particolare **servant**, il **Default Servant**

CORBA Attivazione 5

## Politiche di Attivazione degli Oggetti

Nella configurazione di default (**USE\_ACTIVE\_OBJECT\_MAP\_ONLY**) il POA riesce ad inoltrare richieste **solo** agli oggetti CORBA **già attivi**

Come **gestire direttamente attivazione/disattivazione**?

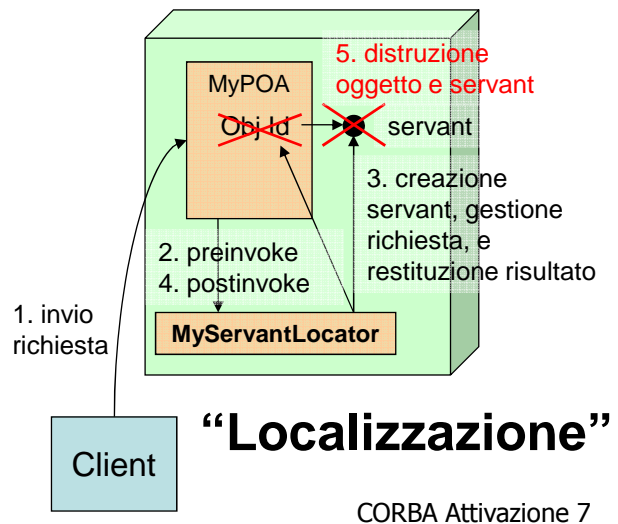
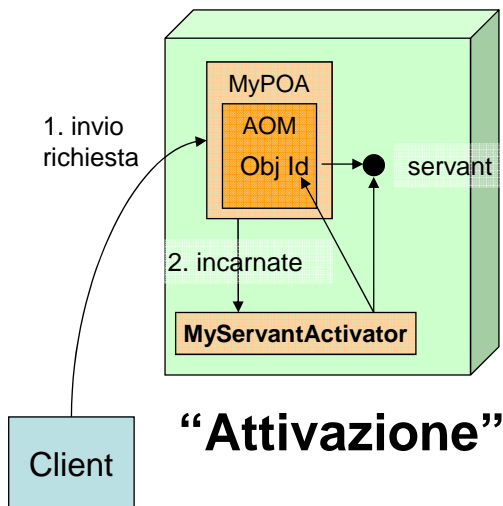
- Configurazione **USE\_SERVANT\_MANAGER** policy e uso di due politiche di attivazione → a seconda di **retain policy**
  - **RETAIN**: se il POA riceve una richiesta per un oggetto servente **non attivo**, cioè non ancora registrato presso l'AOM, il POA:
    - **Attiva** l'oggetto servente (e il relativo servant), lo **registra** presso l'AOM, e lo **lascia attivo** anche dopo il servizio della richiesta
    - **Invia** tutte le richieste successive alla prima all'oggetto attivato
  - **NON\_RETAIN**: l'AOM non viene utilizzato, perciò ∇ richiesta il POA:
    - **Attiva** l'oggetto servente (e il relativo servant)
    - **Disattiva** l'oggetto servente (e il relativo servant) **subito dopo** aver restituito il risultato

CORBA Attivazione 6

# Servant Manager

In realtà, il POA usa due ServantManager per gestire l'attivazione:

- Nel primo caso (**RETAIN**) si utilizza il **ServantActivator**
- Nel secondo caso (**NON\_RETAIN**) il **ServantLocator**



## Esempio: Message - IDL

```
module MessageApp {  
    /* Eccezione in caso di errore durante l'elaborazione */  
    exception ErroreApplicativo {  
        string codice_errore;  
    };  
  
    interface Message {  
        /* Separa un messaggio in due parti, all'invocazione:  
        * msg -> messaggio  
        * separatore -> separatore  
        *  
        * al ritorno:  
        * inizio -> prima parte del messaggio  
        * msg -> parte restante del messaggio  
        */  
        void splitMessage( inout string msg, out string inizio, in string separatore)  
            raises (ErroreApplicativo);  
    };  
};
```

Vogliamo applicare le politiche di attivazione appena viste all'oggetto CORBA che presenta questa interfaccia

# Esempio: Message - Oggetto Servente

```
public class MessageImpl extends MessagePOA
{
    org.omg.PortableServer.POA myPOA = null;

    public void splitMessage(
        org.omg.CORBA.StringHolder      msg,
        org.omg.CORBA.StringHolder      inizio,
        java.lang.String                 separatore )
        throws org.omg.CORBA.SystemException,
               serverPack.MessageApp.ErroreApplicativo
    {
        StringTokenizer tok =
            new StringTokenizer(msg.value, separatore);
        // inizio (out string)
        inizio.value = tok.nextToken();
        // msg (inout string)
        msg.value = tok.nextToken();
    }
}
```

Realizzazione per estensione

Logica applicativa, unica parte scritta dal programmatore

CORBA Attivazione 9

## Passi di Sviluppo e Modifiche Necessarie

1. L'applicazione **client** rimane **inalterata**
2. L'**oggetto server** rimane pressoché inalterato; solo lievi modifiche, se non erano già state previste alla realizzazione dell'oggetto servente:
  - Inizializzazione del **POA**
  - Riscrittura (**override**) del metodo che restituisce il **POA di default** → **\_default\_POA**
3. Il **server**, invece deve essere modificato:
  - Realizzazione del **servant manager**
    - MyServantActivator → caso politica **RETAIN**
    - MyServantLocator → caso politica **NON\_RETAIN**
  - Modifica l'**applicazione server**
    - Creare un nuovo POA
    - Configurare le politiche di gestione
    - Creare e impostare il Servant Manager sul POA

CORBA Attivazione 10

## Modifiche Oggetto Servente

```
public class MessageImpl extends MessagePOA
{
    org.omg.PortableServer.POA myPOA = null;

    // Costruttore, che inizializza il POA
    public MessageImpl(POA myPOA){ this.myPOA=myPOA; }

    public void splitMessage(
        org.omg.CORBA.StringHolder      msg,
        org.omg.CORBA.StringHolder      inizio,
        java.lang.String                 separatore )
        throws org.omg.CORBA.SystemException,
            serverPack.MessageApp.ErrorApplicativo
    { // Qui c'è la logica applicativa! ... }

    // Override default POA
    public POA _default_POA(){ return myPOA; }
}

```

Inizializzazione POA

Override del metodo `_default_POA`

CORBA Attivazione 11

## Attivazione: MyServantActivator

```
public static class MyServantActivator
    extends ServantActivatorPOA
{ // Metodo per l'attivazione; invocato automaticamente
  // alla prima attivazione dell'oggetto
  public Servant incarnate(byte[] values, POA myPOA)
    throws ForwardRequest {
    System.out.println
      ("Richiesta di attivazione oggetto con ID "
       + new String(values));
    return new MessageImpl(myPOA);
  }

  // Metodo per liberare le risorse; invocato
  // automaticamente alla disattivazione del POA
  public void etherealize(byte[] values, POA myPOA,
    Servant servant, boolean param, boolean param4) {
    System.out.println("etherealize dell'oggetto con ID "
      + new String(values));
  }
  // Metodo etherealize per la disattivazione (e.g.
  // memorizzazione stato in DB); qui è vuoto
}

```

Metodo **incarnate** per l'attivazione dell'oggetto

Metodo **etherealize** per la **disattivazione** (e.g. memorizzazione stato in DB); qui è vuoto

CORBA Attivazione 12

# Attivazione: Modifiche Applicazione Server

---

**Prima** (vedi lucidi esercitazioni precedenti)

```
...
// Creo oggetto servente
MessageImpl servant = new MessageImpl();
// Registro e attivo l'oggetto presso il root POA
org.omg.CORBA.Object obj =
    rootPOA.servant_to_reference(servant);
...
```

**Ora:** 1) creazione **politiche** per la configurazione del POA

```
...
Policy[] policies = new Policy[2];
policies[0] = rootPOA.create_servant_retention_policy
    (ServantRetentionPolicyValue.RETAIN);
policies[1] = rootPOA.create_request_processing_policy
    (RequestProcessingPolicyValue.USE_SERVANT_MANAGER);
```

CORBA Attivazione 13

## Attivazione: myPOA e Servant Activator

---

2) creazione e configurazione **myPOA**, impostazione **ServantActivator**

```
POA my_poa = rootPOA.create_POA
    ("myPOA", rootPOA.the_POAManager(), policies);
// creazione e impostazione Servant Activator sul POA
MyServantActivator myActivator = new MyServantActivator();
ServantManager servant_manager_reference =
    myActivator._this(orb);
my_poa.set_servant_manager(servant_manager_reference);
```

3) creazione **Object Id** e **Object Reference** dell'oggetto servente

```
byte[] id = "message_impl".getBytes();
// Preparo l'oggetto CORBA, non ancora attivato
// N.B.: Dato l'Object Id e l'interfaccia CORBA
// (MessageHelper.id()) ottengo un Object Reference
tmp_obj = my_poa.create_reference_with_id
    (id, MessageHelper.id());
```

Il resto del codice è **identico** (anche la registrazione presso il naming service)

CORBA Attivazione 14

## Localizzazione: MyServantLocator

---

```
public static class MyServantLocator
    extends ServantLocatorPOA
{ private Servant servant;

    public synchronized Servant preinvoke
        (byte[] values, POA myPOA, String str, CookieHolder
        cookieHolder) throws ForwardRequest
    { System.out.println
        ("preinvoke per Obj Id "+new String(values));
        servant = new MessageImpl(myPOA);
        return servant;
    } //preinvoke

    public void postinvoke(byte[] values, POA myPOA, String
        str, java.lang.Object obj, Servant servant)
    {System.out.println("postinvoke dell'oggetto con ID " +
        new String(values));
        // de-allocazione memoria
        servant = null;
    } //postinvoke
}
```

Metodo **preinvoke** per l'attivazione

**Attivazione** dell'oggetto

**Postinvoke** invocato automaticamente dopo la gestione della richiesta → **libera risorse**

CORBA Attivazione 15

## Localizzazione: Modifiche Applicazione Server

---

**Prima** (vedi lucidi esercitazioni precedenti)

```
...
// Creo oggetto servente
MessageImpl servant = new MessageImpl();
// Registro e attivo l'oggetto presso il root POA
org.omg.CORBA.Object obj =
    rootPOA.servant_to_reference(servant);
...
```

**Ora:** 1) creazione **politiche** per la configurazione del POA

```
...
Policy[] policies = new Policy[2];
policies[0] = rootPOA.create_servant_retention_policy
    (ServantRetentionPolicyValue.NON_RETAIN);
policies[1] = rootPOA.create_request_processing_policy
    (RequestProcessingPolicyValue.USE_SERVANT_MANAGER);
```



# Localizzazione: myPOA e Servant Locator

---

2) creazione e configurazione **myPOA**, impostazione **ServantLocator**

```
POA my_poa = rootPOA.create_POA
    ("myPOA", rootPOA.the_POAManager(), policies);
// creazione e impostazione Servant Locator sul POA
MyServantLocator myLocator = new MyServantLocator();
ServantManager servant_manager_reference =
    myLocator._this(orb);
my_poa.set_servant_manager(servant_manager_reference);
```

3) creazione **Object Id** e **Object Reference** dell'oggetto servente

```
byte[] id = "message_impl".getBytes();
// Preparo l'oggetto CORBA, non ancora attivato
// N.B.: Dato l'Object Id e l'interfaccia CORBA
// (MessageHelper.id()) ottengo un Object Reference
tmp_obj = my_poa.create_reference_with_id
    (id, MessageHelper.id());
```

Il resto del codice è **identico** (anche la registrazione presso il naming service)