RMI/IIOP

RMI/IIOP è una nuova versione di Java RMI che combina la facilità di programmazione di RMI con l'interoperabilità di CORBA.

CORBA

- CORBA (Common Object Request Broker Architecture) è uno standard sviluppato, a partire dal 1989, dall'OMG (Object Management Group), un consorzio di cui oggi fanno parte oltre 800 aziende sparse per il mondo.

- CORBA è una specifica per la programmazione di oggetti distribuiti che consente a client e server di interagire, indipendentemente dalla piattaforma e dal linguaggio di programmazione usato.

- Ogni grande azienda ha sostanzialmente le medesime problematiche nella gestione dei propri sistemi informatici:

1. Un insieme di dispositivi hardware che differiscono per potenza di calcolo, produttore, ect.
2. Software sviluppati in ambienti differenti. Ciò riguarda sistemi operativi, linguaggi di programmazione, librerie, strutture di rete, database, ect.
3. Programmatori con specializzazioni e capacità differenti.
4. Bisogno di mantenere ed estendere vecchie applicazioni.
5. Desiderio di non acquistare i propri prodotti informatici dal medesimo produttore per ridurre la propria dipendenza e quindi la propria vulnerabilità.

- La domanda che si è posto l'OMG è sostanzialmente la seguente:

  "Come si possono costruire sistemi distribuiti tenendo conto di questi fattori?"

  in particolare quando le richieste sono del tipo: *"Prendi questi due sistemi già esistenti ed integrali in modo da fornire un nuovo servizio"*.

- CORBA è stato sviluppato per venire incontro a questi problemi definendo una specifica per sistemi distribuiti ad oggetti, indipendente dalla piattaforma e dal linguaggio di programmazione adottato.

- Dalla prima versione del 1989, ne sono state sviluppate molte altre fino ad arrivare alla versione più recente, CORBA 3.0.2 (Dicembre 2002).

Come funziona CORBA?

- L'Obiettivo di CORBA è quindi di integrare in una singola applicazione distribuita due programmi differenti, scritti in linguaggi differenti, ed in esecuzione su macchine differenti.

- Per fare ciò CORBA ha bisogno di un meccanismo per definire come comunicano i programmi coinvolti, come vengono passati i valori tra programmi, e che protocolli utilizzano.

- Per poter integrare sistemi differenti, CORBA provvede alla creazione di *interfacce* scritte in un linguaggio ben preciso, facile da comprendere, e che possono essere facilmente tradotte nei vari linguaggi di programmazione disponibili.

- I concetti fondamentali di CORBA sono:

  1. **Un IDL (Interface Definition Language)**. IDL è un linguaggio per definire l'interfaccia fornita da un server CORBA. Un'interfaccia IDL descrive come client e server interagiscono; ma non dice nulla sull'implementazione dei metodi del server. Un'interfaccia IDL è concettualmente molto simile ad un'interfaccia remota usata per esprimere il comportamento di un server in Java RMI.

  2. **Il protocollo di rete IIOP**. Buona parte delle performance di CORBA deriva dall'alta velocità di IIOP (Internet Interoperability Protocol) e dalla sua capacità di creare velocemente connessioni tra oggetti.

  3. **Un insieme di traduttori**.

- La capacità di CORBA di interagire con applicazioni scritte in altri linguaggi si basa proprio sulla traduzione delle interfacce IDL nei linguaggi di programmazione utilizzati.
- Un'interfaccia IDL contiene tutte le informazioni necessarie a descrivere l'interazione tra client e server.
- In particolare, a partire da un'interfaccia IDL, possiamo generare stub e skeleton per Java RMI.
- In Java2 questo viene effettuato attraverso il comando `idlj` che produce un insieme di classi Java, incluso interfacce, stub e skeleton.
- In questo modo un client Java RMI può interagire con un server CORBA.
- Nota che `idlj` non ha bisogno dell'implementazione del server per creare gli stub; al contrario di quanto accade in Java RMI con `rmic`.

CORBA vs RMI

CORBA e Java RMI rappresentano due framework per lo sviluppo di sistemi distribuiti ad oggetti. Facciamone un breve confronto.

- CORBA è progettato per garantire la massima interoperabilità e poter così utilizzare applicazione scritte usando linguaggi e piattaforme differenti.
- Esiste una grossa mole di codice e di applicazioni sviluppate usando CORBA.
- Le performance delle applicazioni CORBA sono in generale molto buone.

Se invece consideriamo Java RMI:

- In Java RMI, client e server devono necessariamente essere sviluppati in Java.
- Java RMI è un linguaggio più semplice ed è quindi più facile da usare per sviluppare grossi sistemi.

- Il codice necessario per descrivere un sistema in Java RMI può essere di gran lunga inferiore a quello necessario per descrivere il medesimo sistema in CORBA.

- Le prestazioni di Java RMI sono in generale inferiori a quelle di CORBA.

- Infine, Java RMI, al contrario di CORBA, supporta un meccanismo di garbage collection distribuito.

La scelta di quale specifica utilizzare per sviluppare un sistema distribuito dipende dall'analisi di questi fattori. Comunque, esistono spesso situazioni in cui si vorrebbe avere i vantaggi di entrambe le specifiche.

Così, nel 1999, da una collaborazione tra IBM e SUN, è nato RMI/IIOP, ovvero una nuova versione di Java RMI che al posto del protocollo originale JRMP (Java RemoteMethod Protocol) usa il protocollo di IIOP di CORBA. Combinando così la facilità di utilizzo di RMI con l'interoperabilità di CORBA.

Ciò è stato possibile poichè CORBA ha accettato due principali modifiche a partire da CORBA2.3:

- La specifica *Object by Value* che consenta compatibiità con la *Java Object Serialization.*

- La traduzione Java-to-IDL, che consente la conversion di interfacce remote Java in definizioni CORBA IDL.

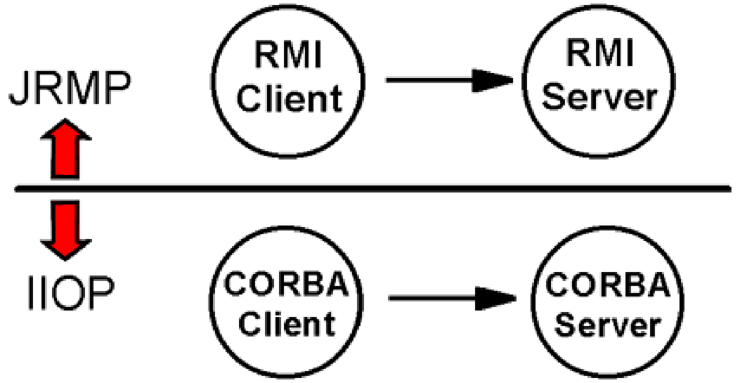

Figura 1: Prima di RMI/IIOP

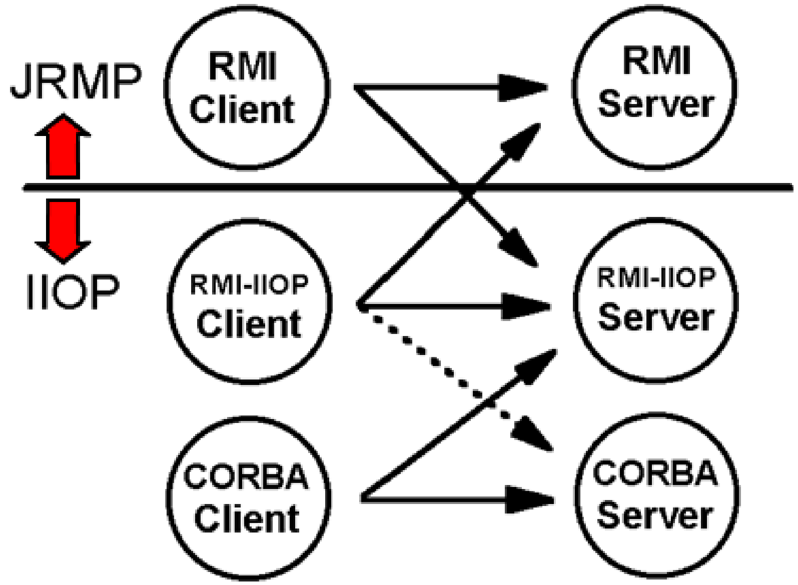

Figura 2: Dopo l'introduzione di RMI/IIOP

RMI/IIOP

- La Figura 1 mostra la situazione prima dell'avvento di RMI/IIOP in cui RMI/JRMP e CORBA non potevano comunicare tra loro poichè basati su protocolli differenti.

- La Figura 2 mostra invece come la tecnologia RMI/IIOP consenta l'interazione di client e server sviluppati in un environment differente.

- La prima cosa che salta all'occhio è che Java RMI supporta entrambi i protocolli JRMP e IIOP.

- Infatti un server RMI può lavorare con entrambi i protocolli senza dover cambiare il proprio sorgente Java. Bisogna solo esportare il server su entrambi i protocolli.

  La stessa flessibilità si applica al codice del client.

Dual export

- Quando si esporta un server RMI non si è obbligati a scegliere tra JRMP e IIOP.

- Un medesimo server RMI può essere esportato su entrambi i protocolli interagendo allo stesso tempo con clients JRMP e IIOP.

- Osservando la Figura 2 notiamo che un un server RMI/JRMP può essere invocato da un client RMI/IIOP.
- Allo stesso modo, un server RMI/JRMP può essere rimpiazzato da uno RMI/IIOP senza che ciò influenzi i suoi clients RMI/JRMP; acquisendo anzi la possibilità di interagire con client CORBA.
- RMI/IIOP infatti è stato sviluppato con l'intenzione di integrare Java RMI con CORBA preservando allo stesso tempo le applicazioni già sviluppate in RMI/JRMP.

Interoperabilità con CORBA

- In Figura 2 la sezione sotto la linea orizzontale rappresenta il mondo IIOP in cui:
  - Un client RMI/IIOP invoca un server CORBA e
  - Un client CORBA (o RMI/JRMP) invoca un server RMI/IIOP.

- Dove con client RMI/IIOP intendiamo un'applicazione scritta in RMI da un programmatore che non conosce nulla di CORBA o IDL.

- Similmente, per client CORBA intendiamo un'applicazione scritta da un programmatore CORBA che non conosce nulla di RMI.

- Ciò è possibile poichè vi è una netta separazione tra interfaccia e implementazione.

- Un'interfaccia Java RMI è l'interfaccia usata da clients RMI/IIOP; mentre un'interfaccia IDL è quella usata da clients CORBA.

- Clients RMI/IIOP e clients CORBA possono interagire con un server sviluppato nell'altro linguaggio purchè trovino le interfacce che si aspetterebbero normalmente.

- L'interoperabilità tra RMI/IIOP e CORBA è attivata fornendo a ciascun utente l'interfaccia che si aspetta e nascondendogli l'implementazione.

- Dato un server CORBA con un'interfaccia IDL possiamo costruire interfacce e stubs per RMI/IIOP attraverso il compilatore `idlj`; viceversa, dato un server RMI/IIOP possiamo costruire un'interfaccia IDL per clients CORBA attraverso `rmic -idl`.

- Infine, si noti che in Figura 2 vi è una linea tratteggiata nel caso di client RMI/IIOP che invoca un server CORBA. Questo perchè un client RMI/IIOP può non essere in grado di invocare un server CORBA.

- Infatti, la semantica di un server CORBA definito attraverso IDL contiene più informazione di quanto ne contenga un server RMI/IIOP.

- Quindi a volte non è possibile tradurre un IDL di un server CORBA in un'interfaccia remota per un server RMI/IIOP.

- Questo ovviamente riguarda server CORBA già esistenti. È invece sempre possibile creare un server CORBA che rispecchi un server RMI/IIOP, creando il suo IDL a partire dall'interfaccia remota del server RMI/IIOP (con `rmic -idl`).

- La Figura 3 mostra che se si implementa un server in RMI/IIOP si ha la massima interoperabilità; similmente clients RMI/IIOP possono interagire con la più grande gamma di servers.

Client \ Server	RMI (JRMP)	RMI (IIOP)	CORBA
RMI (JRMP)	○	○	X
RMI (IIOP)	○	○	◌
CORBA	X	○	○

Figura 3: Matrice RMI/IIOP

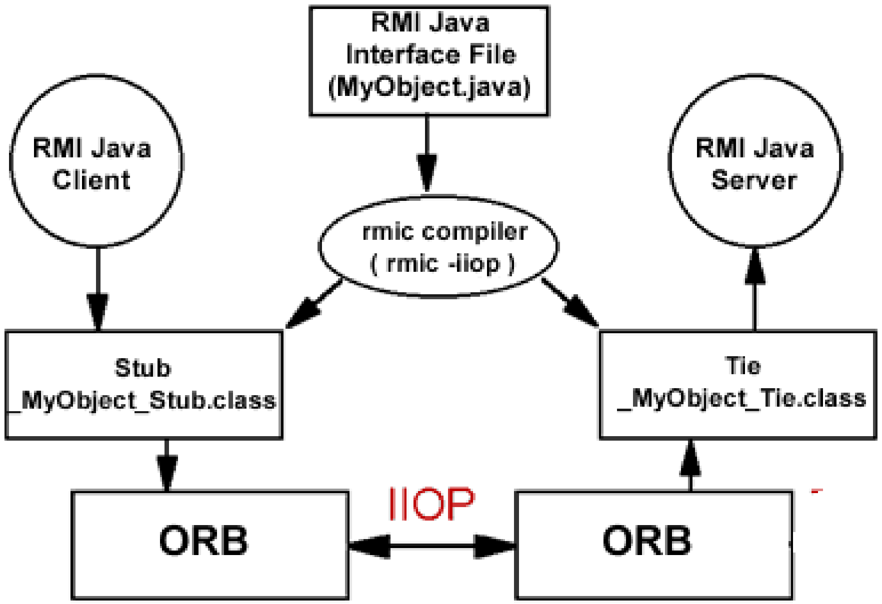

Figura 4: Sviluppo applicazione RMI/IIOP

Programmare client e server in RMI/IIOP

- In Figura 4 schematizziamo le fasi di sviluppo di un server RMI/IIOP.
- Come potete notare è molto simile a quanto avviene con RMI/JRMP.

- Come per RMI/JRMP la semantica di un server è data da un'interfaccia remota (ad esempio `MyObject.java`).

- A differenza di quanto accade in RMI/JRMP la compilazione `rmic` avviene con il parametro `-iiop` per generare le classi *stub* e le classi *tie* per il protocollo IIOP.

- Le classi *tie* sono usate nell'host del server per elaborare invocazioni verso il server ed inoltrare le richieste all'appropriata implementazione. Il loro funzionamento è simile agli skeleton.

- Senza tale opzione `rmic` genera stub e skeleton per il protocollo JRMP.

Programmazione RMI/JRMP vs RMI/IIOP

Vediamo adesso le principali differenze nello scrivere un server ed un client in RMI/IIOP rispetto a quanto avveniva con RMI/JRMP. Cominciamo col server:

1. **Comandi di importazione.** I comandi di importazione devono tenere conto del fatto che:

   - la classe base dei server RMI/IIOP è `PortableRemoteObject`.
   - Il servizio di naming attraverso JNDI fa uso della classe `InitialContext`.

   Quindi:

   ```
   import javax.rmi.PortableRemoteObject;
   import javax.naming.InitialContext;
   ```

2. **Implementazione della classe del server remoto.**

   ```
   public class MyObject_Impl extends PortableRemoteObject
                                implements IntRemota
   ```

3. **Costruttore.** Il costruttore di un server `PortableRemoteObject` non può essere omesso.

4. **Registrazione del server remoto con COSNaming.**

   `InitialContext.rebind(‘‘Troy’’, ObjRef);`

5. **Creazione degli stub e dei tie**. Attraverso `rmic -iiop`.

6. **Lancio del COSNaming**. Digitando ad esempio:

   ```
   tnameserv -ORBInitialPort 5555
   ```

   per lanciare il servizio di COSNaming alla porta 5555.

7. **Creazione dell'interfaccia IDL.** Ciò viene fatto rilanciando `rmic` col parametro `-idl`. Queste interrfacce vengono create per i client CORBA che per l'appunto sono ignare del fatto che il server sia stato sviluppato in RMI/IIOP poichè trovano le loro interfacce CORBA.

Client RMI/IIOP

Per quanto riguarda client RMI/IIOP le differenze riguardano essenzialmente l'importazione delle classi opportune e la lookup attraverso JNDI:
Comandi di importazione.

```
import javax.rmi.PortableRemoteObject;
import javax.naming.InitialContext;
```

Lookup

```
// Supponiamo d'avere definito env opportunamente.
InitialContext IC = new InitialContext(env);
Object obj = IC.lookup(``Troy'');
IntRemota myobj =
  (IntRemota)PortableRemoteObject.narrow(obj,IntRemota.class);
```

Quando un client RMI/IIOP o CORBA spedisce una richiesta di lookup al COSNaming per un determinato server, il servizio ritorna un IOR (Interoperable Object Reference) usato dal client per comunicare col server. Al contrario degli stub, l'informazione contenuta in un IOR è al quanto illegibile.

Supportare sia JRMP e IIOP

Nel caso si voglia esportare un server RMI che supporti entrambi i protocolli bisognerà adottare i seguenti accorgimenti:

1. Il server non deve estendere né `UnicastRemoteObject` né `PortableRemoteObject` e né `Activatable`

2. Il costruttore del server deve prendersi cura di esportare il server nei due protocolli usando il metodo `exportObject` di `UnicastRemoteObject` e `PortableRemoteObject`;

3. Bisogna usare JNDI per implementare sia un registro RMI che il COS Naming di CORBA. Si creano due `InitialContext` uno per consentire il binding del server sul registro RMI ed uno per il COS Naming. Dopodichè si registra il server su entrambi i servizi di naming;

4. Non si passano i valori dei servizi di naming da linea di comando con l'opzione -D.

Vediamo uno schema di dual export:

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Properties;
import javax.naming.InitialContext;
import javax.rmi;

public class DualServer implements IntRemota {
  public DualServer() throws RemoteException {
    UnicastRemoteObject.exportObject(this); // esporta su JRMP
    PortableRemoteObject.exportObject(this); // esporta su IIOP
  }
  //...... implementazione dei metodi remoti
  public static void main(String[] args) throws Exception {
    System.setSecurityManager(new RMISecurityManager);
    DualServer server = new DualServer();
    // BIND SUL REGISTRO RMI
```

```
    Properties p_jrmp = new Properties();
    p_jrmp.put(``java.naming.factory.initial'',
      ``com.sun.jndi.rmi.registry.RegistryContextFactory'');
    InitialContext c_jrmp = new InitialContext(p_jrpm);
    c_jrpm.bind(``MioServerDuale'', server);
    // BIND SUL COSNaming
    Properties p_iiop = new Properties();
    p_iiop.put(``java.naming.factory.initial'',
      ``com.sun.jndi.cosnaming.CNCtxFactory'');
    InitialContext c_iiop = new InitialContext(p_iiop);
    c_iiop.bind(``MioServerDuale'', server);
  }
}
```

Restrizioni nell'uso di RMI/IIOP

RMI/IIOP fornisce quindi la possibilità di definire servers in Java per poi esportarli verso client CORBA. Comunque, se si vuole convertire del codice RMI/JRMP in RMI/IIOP è bene sapere che vi sono alcune differenze/restrizioni di cui tenere conto:

1. Tutte le interfacce remote coinvolte devono essere definite in Java RMI; non è possibile importare interfacce definite precedentemente in altri linguaggi come ad esempio in IDL.

2. La garbage collection distribuita non è supportata da RMI/IIOP. Più precisamente gli oggetti `PortableRemoteObject` non partecipano alla garbage collection distribuita. Di conseguenza l'interfaccia `Unreferenced` non è supportata da IIOP.

3. Le definizioni di costanti nelle interfacce remote devono essere di tipo primitivo o di tipo `String`.

4. Gli oggetti `PortableRemoteObject` non possono estendere `UnicastRemoteObject` o `Activatable` e quindi non possono essere attivabili; il protocollo di attivazione è supportato solo su JRMP.

5. Gli oggetti `PortableRemoteObject` non supportano l'uso esplicito di porte o di socket factories per l'esportazione.

6. Nei server RMI/IIOP, l'uso del registro RMI come servizio di naming è rimpiazzato dal JNDI attraverso CosNaming o LDAP.

7. Il cast ad interfacce remote deve essere preceduto da un'invocazione al metodo `PortableRemoteObject.narrow`:

```
public static Object narrow (Object narrowfrom,
                             Class narrowto)
                      throws ClasssCastException
```

8. La condivisione di referenze remote non è preservata esattamente quando si trasmettono *referenze remote* su IIOP. In pratica, grafi di oggetti con in-degree > 1 non sono trasmessi correttamente. Tutti gli altri grafi di oggetti vengono trasmessi correttamente.

9. L'uso di `rmic -iiop` impone una regola non scritta nella documentazione della Sun. Ovvero, un server remoto non può estendere `RemoteServer` sebbene può estendere `RemoteObject`.