Proxy

In Java RMI uno schema di agente molto diffuso è quello del proxy.

- Un proxy è un server che viene interposto tra il client ed il vero server.
- Un proxy ed il suo server sono compatibili dal punto di vista dei tipi, ovvero un proxy può implementare una delle interfacce implementate dal server o estendere una delle classi base del server.
- Un proxy mantiene sempre una referenza al server vero e proprio con cui comunicherà in maniera trasparente al client, il quale crederà di intergire direttamente col server.
- I proxy vengono usati ogni qual volta sia necessario introdurre un'entità intermedia di controllo.
- Vediamo adesso alcune forme di proxies.

Smart proxy

- Uno smart proxy è qualcosa di più di un semplice proxy.
- Uno smart proxy può essere usato per nascondere la presenza di un'interfaccia remota ad un client.
- Il client infatti interagisce con un oggetto locale (privo di interfacce remote) senza sapere che questo è in realtà uno smart proxy che delega certe funzionalità ad un server remoto mentre altre sono svolte localmente.
- Questo è un meccanismo efficiente ed elegante per distribuire l'implementazione tra macchina locale e server remoto, in maniera del tutto trasparente al client.

Remote Proxy

- Un remote proxy è un rappresentante locale di un server remoto (che si trova in un altro host).

- Uno stub RMI è esattamente un remote proxy.
- Infatti uno stub contiene (o meglio nasconde al client) il meccanismo attraverso cui è possibile comunicare col server remoto.
- Uno stub implementa la medesima interfaccia del server remoto. In tal modo il client pensa di interagire direttamente col server remoto.

Smart Remote Proxy

- In tale schema il client, invece di interagire con uno stub interagisce con un proxy locale che implementa l'interfaccia remota e che opera da proxy verso lo stub al server remoto.
- In pratica, quando il client richiede una referenza ad un server remoto, invece di ricevere uno stub, riceve per deep copy, uno smart remote proxy. Uno smart remote proxy è un oggetto serializzabile che contiene al suo interno una referenza al server remoto, ovvero

uno stub. Quindi, al momento della serializzazione il client riceverà sia il proxy che lo stub remoto; sebbene potrà direttamente interagire solo col primo.

- Questo schema è utile quando il server vuole che alcune operazioni siano eseguite presso il client mentre altre siano demandate al server.

- Questo schema ha delle ovvie ripercussioni sulla sicurezza visto che l'implementazione del proxy è fornita interamente dal server, ed il client deve fornire permessi adeguati per le operazioni eseguite dal proxy.

Activatable Proxy

- Abbiamo visto che esistono per lo meno due modi per passare da un server `UnicastRemoteObject` ad uno `Activatable`.
  - Estendendo `Activatable` invece di `UnicastRemoteObject`, oppure

- esportando il server con `Activatable.exportObject` invece di `UnicastRemoteObject.exportObject`.

- Un'altra possibilità è quella di lasciare il server Unicast inalterato ed introdurre un proxy attivabile che inoltri tutte le richieste all'oggetto unicast.

- In questo caso, ovviamente, nel registro RMI verrano riportati gli stub ai proxy attivabili e non quelli al server Unicast.

- In questo modo vi è una netta separazione in fase di progettazione tra ciò che riguarda l'attivazione e ciò che riguarda l'implementazione del server.

- Al costo, ovviamente, di una invocazione in più, quella tra il proxy ed il server.

- Una tale soluzione è auspicabile quando il server unicast deve essere continuamente attivo per delle elaborazioni locali, mentre solo saltuariamente avvengono delle invocazioni remote che comunque necessitano una pre-elaborazione da parte dei proxy.

Schemi di programmazione distribuita

Client-server

Client e server giocano un ruolo diverso. Il client interroga il server richiedendo un servizio. Il server riceve richieste e ritorna risultati. Solitamente è il client ad iniziare la connessione ed anche quello che chiude la connessione quando ha ottenuto quanto desiderato.

Client-dispatcher-server

- In questo caso esiste un dispatcher che media tra client e server. Il client comunica col dispatcher per accedere al server indicandone il nome. Il dispatcher si prende cura di inoltrare la richiesta al server con quel nome.

- Il sistema RMI implementa tale schema utilizzando stubs e skeletons.

Peer to Peer

In tale schema non vi è distinzione tra client e server. Tutte le entità coinvolte possono fornire e richiedere servizi.

Singleton

- Una classe viene detta *Singleton* se ne esiste una sua sola istanza.
- Una classe singleton viene spesso usata per incapsulare un processo che deve essere sequenzializzato tra più utenti (ad esempio, lo spooler di una stampante) oppure per rappresentare una risorsa di cui esiste una sola istanza (ad esempio un file systems o un database).
- La classe `java.lang.Runtime` è un esempio di singleton, che rappresenta il sistema a runtime di Java.
- In generale una classe singleton è priva di stato.
- A tal proposito, quando si progetta un server remoto, ci si può chiedere se creare o meno più istanze del server per sopperire al-

le richieste dei client. In generale, è preferibile avere più istanze di un server se i clients devono registrare al suo interno i loro cambiamenti di stato. Altrimenti è sufficiente un singleton.

- I server registrati sul registro RMI sono in generale singleton perchè la medesima istanza viene usata da più clients.

Abstract Remote

In tale schema vi è un classe astratta che implementa un'interfaccia remota:

```
public interface RemoteService extends Remote { ....
}
public abstract class AbstractService implement RemoteServer {
  public AbstractService() throws RemoteException {...}
}
public class ConcreteService extends AbstractService {
............
}
```

- Una classe astratta non deve fornire il codice dell'implementazione, eccezion fatta per il costruttore. In particolare non deve reiterare la dichiarazione dei metodi remoti.

- Una classe astratta remota può essere compilata da `rmic` generando stub e skeleton.

- IMPORTANTE: le implementazioni concrete non vengono elaborate da rmic.

- Di conseguenza, in fase di sviluppo posso cambiare l'implementazione del server remoto senza dover rilanciare rmic e modificare gli stub e gli skeleton.

- La compilazione rmic dovrà essere rieseguita solo se si cambia l'interfaccia remota.

- In questo modo si semplifica sensibilmente la fase di progettazione del server.