UNIVERSITÀ DEGLI STUDI DI BRESCIA
FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA INFORMATICA
DIPARTIMENTO DI ELETTRONICA PER L'AUTOMAZIONE

Il Pattern **PROXY**

Ex presentazione realizzata dallo studente
Paolo Melchiori (matricola 65734)
nell'a.a. 2007-2008

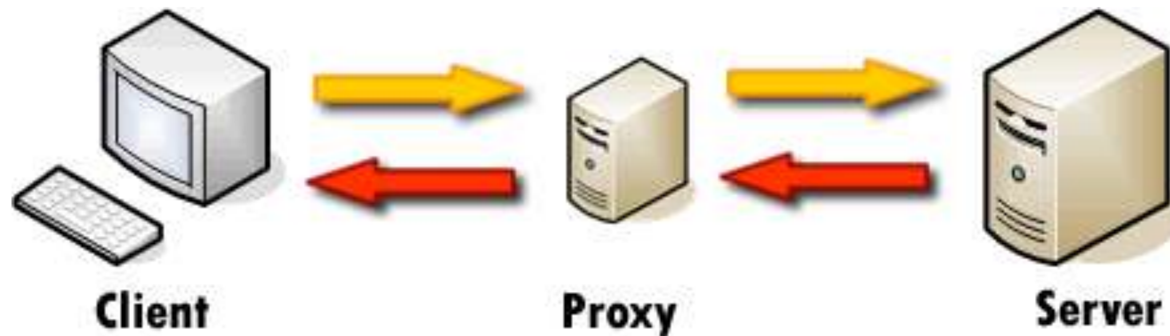
Un album fotografico…

Immaginiamo di essere contattati da un cliente interessato a creare un programma che realizzi un **album digitale** in grado di catalogare e visualizzare foto presenti sia nel computer locale dell'utente sia in un'ipotetica cartella remota, per esempio, condivisa da più persone o semplicemente a disposizione su qualche sito internet (versione "*on-line*").

Nel caso in cui si tratti di poche fotografie può essere utile, soprattutto al fine della maggiore velocità di accesso ai contenuti, caricare tutte le foto a cui si può essere interessati.

Tale soluzione evidenzia notevoli **problemi prestazionali**, se non altro nella soluzione "on line", con la quale si genererebbe una mole notevole di traffico inutile anche perché sarebbero scaricate foto a cui l'utente potrebbe non essere interessato (*purtroppo molti siti web flash-based sembrano aver abbracciato questa teoria*).

Una soluzione più furba!



PROXY
Da Wikipedia, l'enciclopedia libera.

Un proxy è un **programma che si interpone** tra un client ed un server, inoltrando le richieste e le risposte dall'uno all'altro.

La struttura generale

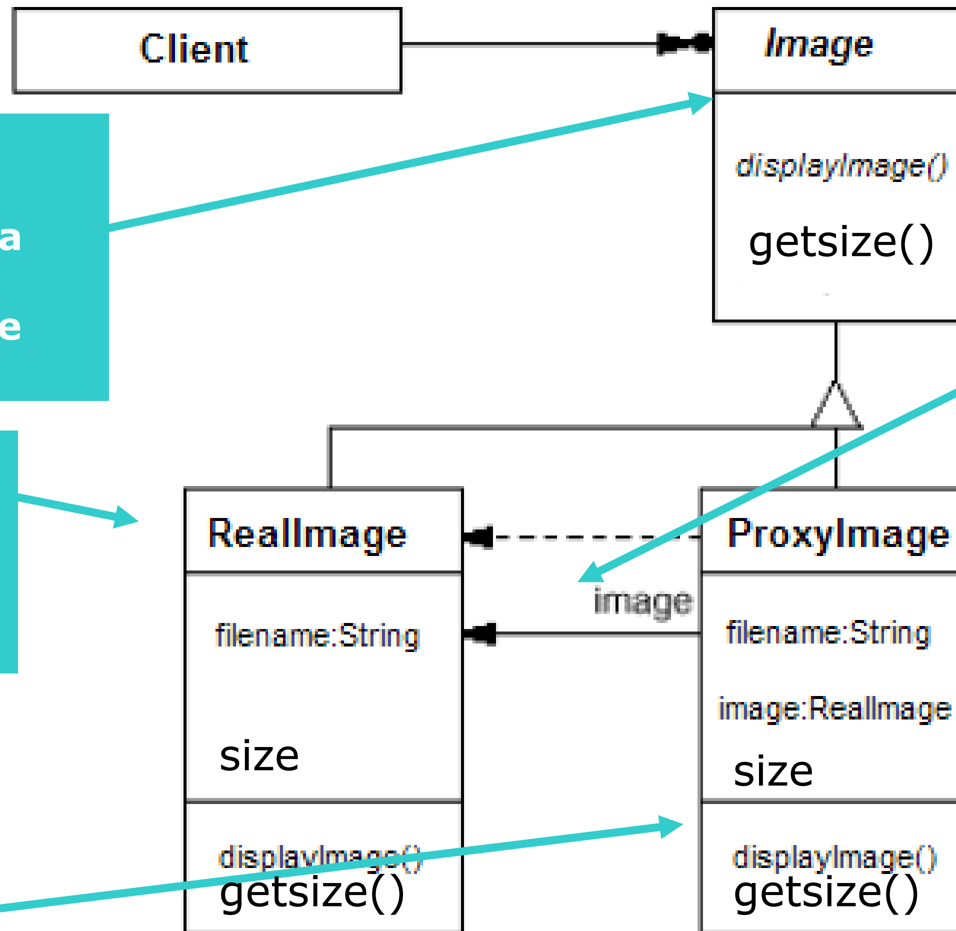


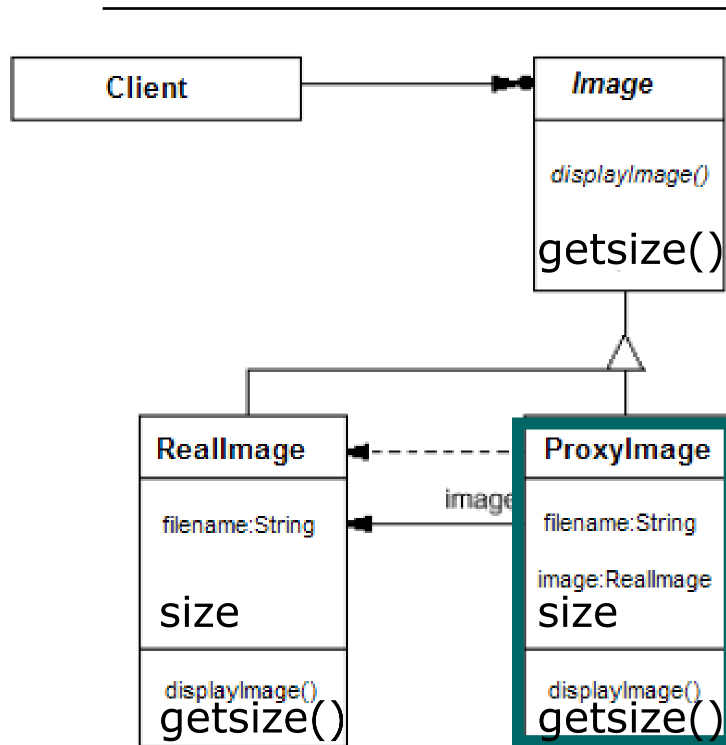
Image
Fornisce un'interfaccia comune per RealSubject e Proxy

Consente al Proxy di accedere all'oggetto a cui fa riferimento

RealImage
Caratterizza l'oggetto reale al quale il client è interessato

ProxyImage
Grazie all'interfaccia comune si può utilizzare Proxy come sostituto di RealSubject

La classe ProxyImage

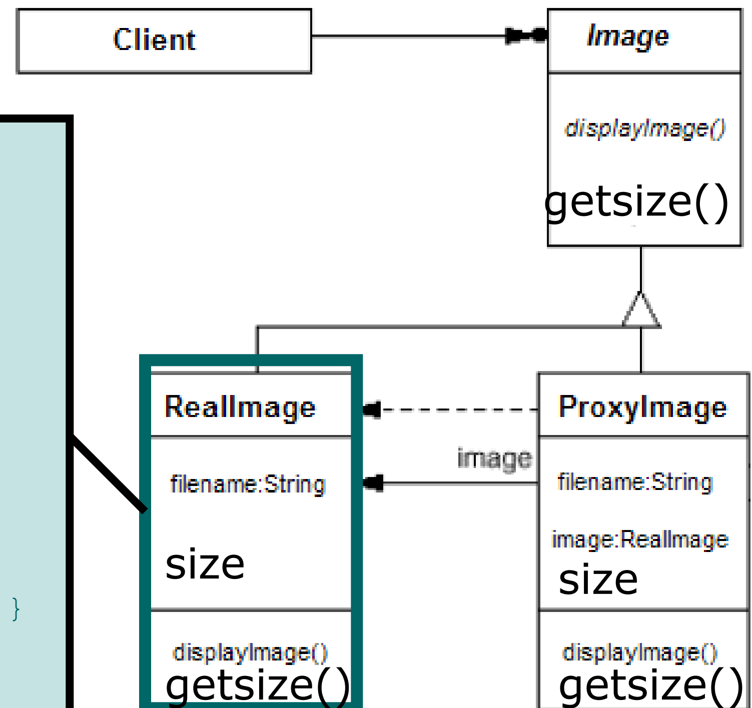



```
public class ProxyImage extends Image {
private String filename;
private RealImage image;
private int size;
public ProxyImage(String filename, int s) {
        this.filename = filename;
        size = s;
        }
public void displayImage() {
        if (image == null) {
        image = new RealImage(filename);
        }
        image.displayImage();
        }
}
public int getsize() {
if (image == null) return size;
return image.getsize()}
```

Nel caso in cui la visualizzazione dell’immagine non sia ancora stata richiesta, il metodo displayImage() crea una nuova istanza di RealImage altrimenti utilizza quella già creata precedentemente.

La classe RealImage

```
public class RealImage extends Image {

private String filename;
private int size;

public RealImage(String filename, int s) {
        this.filename = filename;
        size = s;
        System.out.println("Loading"+
                            filename);
                            }


public void displayImage() {
System.out.println("Displaying " + filename);}

}

public int getsize() {return size;}
```



Il costruttore della classe avvisa l'utente del caricamento in corso mentre il metodo displayImage(), invocato da ProxyImage, evidenzia sullo schermo il nome del file visualizzato.

Un esempio di esecuzione

```
public class ProxyEsempio {
public static void main(String[] args) {
        ArrayList sequenza = new ArrayList();

        sequenza.add(new ProxyImage("Photo_1.jpg"));
        sequenza.add(new ProxyImage("Photo_2.jpg"));
        sequenza.add(new ProxyImage("Photo_3.jpg"));

        ((ProxyImage)sequenza.get(0)).displayImage();
        ((ProxyImage)sequenza.get(1)).displayImage();
        ((ProxyImage)sequenza.get(0)).displayImage();
        }
            }
```

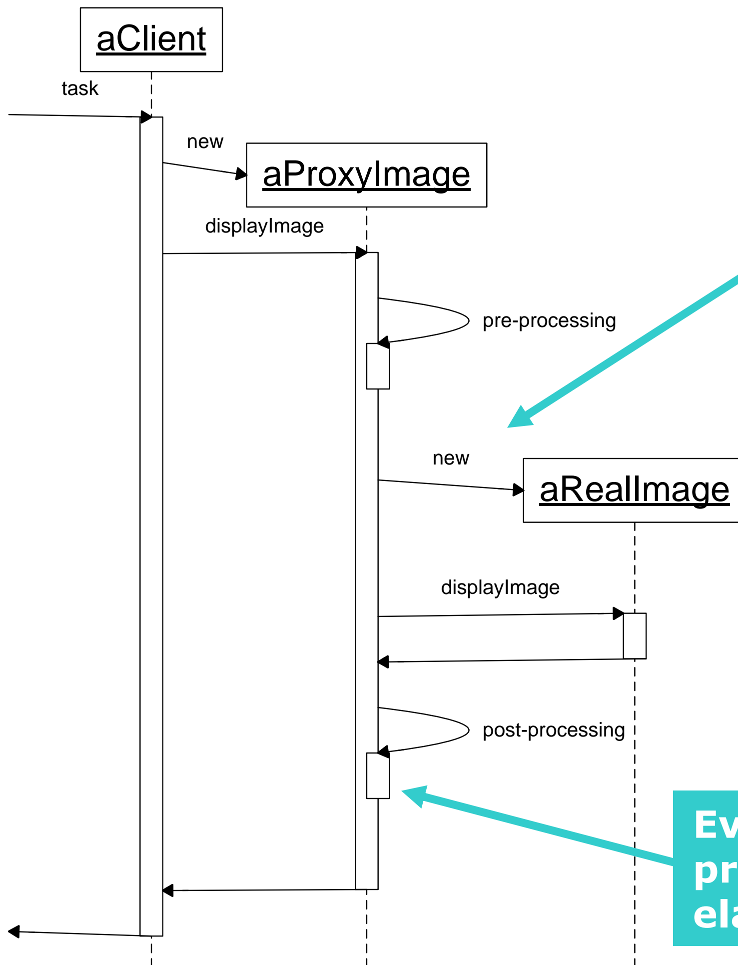
Immaginiamo un semplice main che gioca il ruolo di client richiamando alcune immagini caricate nell'array

Un esempio di esecuzione

- Loading Photo_1.jpg
- Displaying Photo_1.jpg
- Loading Photo_2.jpg
- Displaying Photo_2.jpg
- Displaying Photo_1.jpg

Ogni immagine è caricata solamente quando il codice richiede che venga visualizzata.
La terza visualizzazione non richiede l'istanziazione di alcun nuovo oggetto: l'immagine è immediatamente visualizzata

Il diagramma di sequenza



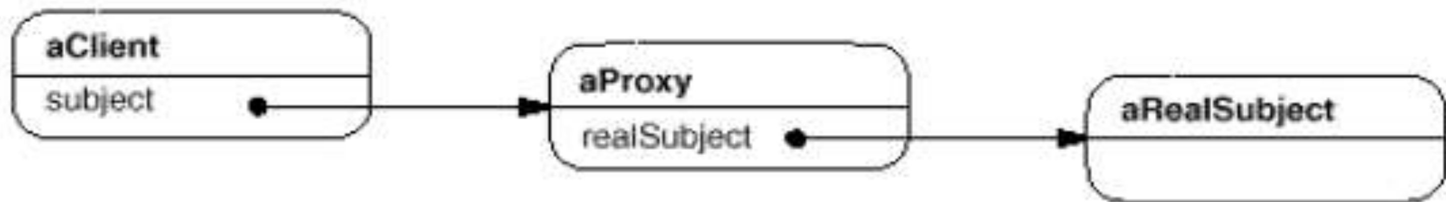
Dopo che un'immagine è già stata istanziata non sarà necessario creare un nuovo oggetto

L'oggetto immagine viene creato solamente se effettivamente richiesto

Eventuali fasi di pre e post elaborazione

Diagramma degli oggetti

Come abbiamo potuto apprezzare anche dal diagramma di sequenza precedente, il diagramma degli oggetti, ottenibile anche per la nostra applicazione d'esempio, può essere così rappresentato



Possiamo notare che **aProxy** continua a mantenere il riferimento all'oggetto reale a cui si riferisce mentre **aClient** continua a puntare all'oggetto **aProxy** che fornisce la stessa interfaccia di **aRealSubject**

Quali vantaggi?

Punti a favore

• I client interessati vengono alleggeriti da molteplici **operazioni di calcolo** effettuate in seguito e solo se necessarie

• Possibilità di **riusare oggetti** già istanziati

• **Non c'è connessione diretta** tra client e server, tutto è vincolato dall'opera del proxy

Punti a sfavore:

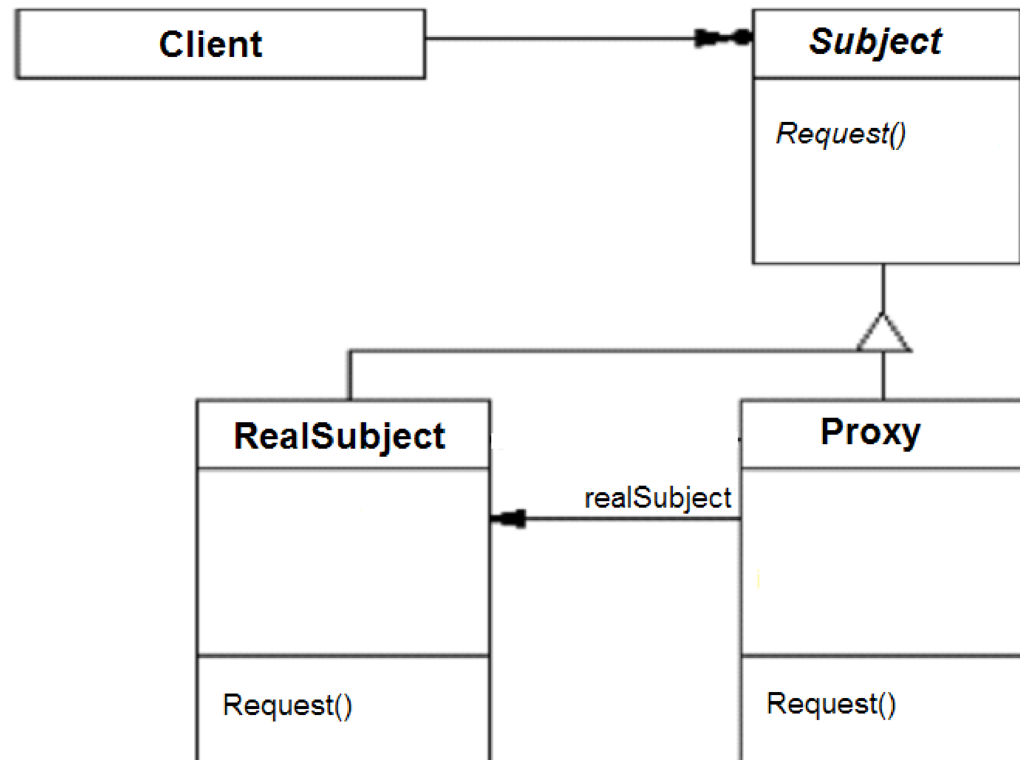
• A ogni invocazione bisogna controllare **l'effettiva esistenza** dell'oggetto

• L'architettura **non è completamente black-box**, infatti il client deve conoscere l'esistenza del proxy per poter istanziare oggetti della classe ProxyImage e non RealImage

Il Pattern Proxy, scheda tecnica

- **Alias**
  Surrogate (*surrogato, segnaposto*)
- **Classificazione**
  Pattern strutturale, basato su oggetti
- **Scopo**
  Fornire un surrogato di un oggetto per controllarne l'accesso
- **Motivazioni**
  - Evitare spreco di risorse di calcolo e di rete
  - Diminuzione del numero di oggetti istanziati
  - Controllo degli accessi a un oggetto
  - Nascondere agli utilizzatori il fatto che un oggetto risieda in uno spazio di indirizzamento diverso

La struttura generale



Applicabilità

- **Remote Proxy**
  Implementa un **rappresentante locale**, in genere più accessibile, di un oggetto esistente in remoto; il client, di fatto, ignora la presenza dell’originale

- **Virtual Proxy**
  La principale funzione è quella di **creare oggetti particolarmente costosi** solo su richiesta del client (*l’esempio visto appartiene a questa categoria*)

- **Protection Proxy**
  Al fine di implementare regole di accesso agli oggetti in funzione dei **privilegi** di cui è in possesso il client

- **Funzionalità aggiuntive**
  Nel caso in cui si vogliano sfruttare funzioni quali il **conteggio** degli accessi a un oggetto, o la possibilità di **bloccare in scrittura** un oggetto già aperto in tale modalità da un altro client

Conseguenze

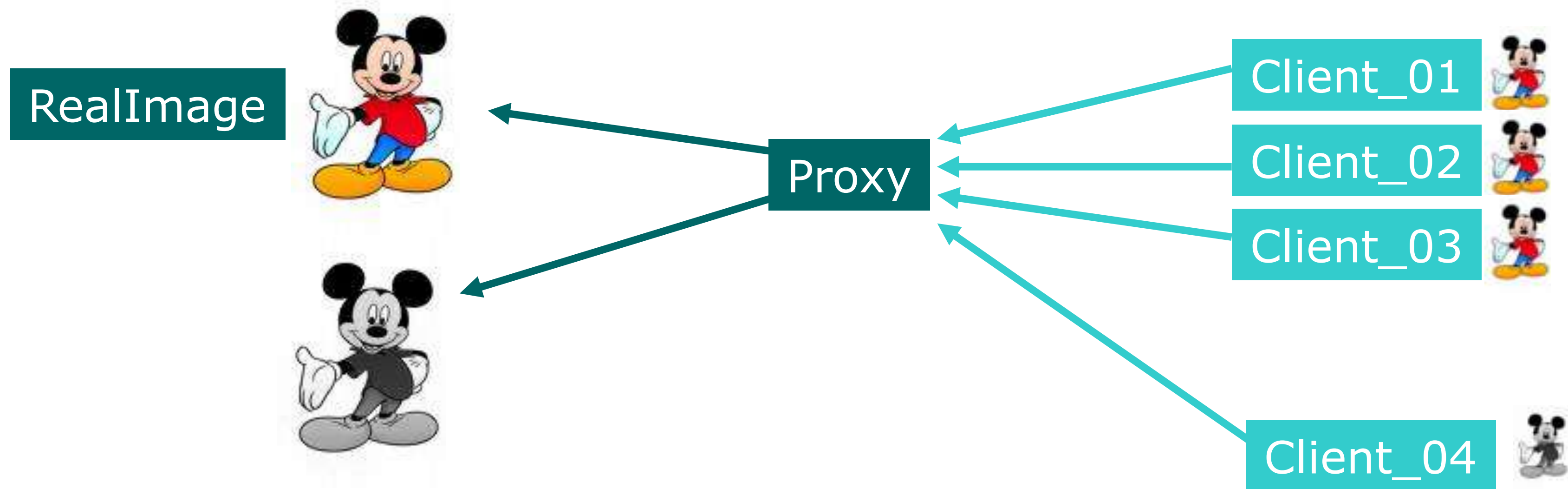
L'utilizzo di un Proxy inserisce un livello intermedio tra il fruitore del servizio (il Client) e il fornitore dello stesso (il Server). In base, dunque, alla tipologia di proxy, si potranno avere comportamenti diversi:

- Un **remote proxy** potrebbe nascondere il reale indirizzo di un oggetto
- Un **virtual Proxy** può ottimizzare il sistema creando oggetti solo su richiesta, anche grazie alla variante copy-on-write
- Versioni come **Protection Proxy** o **funzionalità aggiuntive** (ad es. riferimento intelligente) possono aggiungere proprietà particolarmente complesse e interessanti secondo le necessità del programmatore

Copy-on-Write

Copy-On-Write. Se un oggetto risulta particolarmente oneroso da creare e, quindi, gestire non si è obbligati a crearne un'istanza ogni volta che ne viene richiesta una copia.

Il proxy creerà una nuova istanza solamente nel caso in cui il client voglia **modificare** il contenuto della vecchia versione.



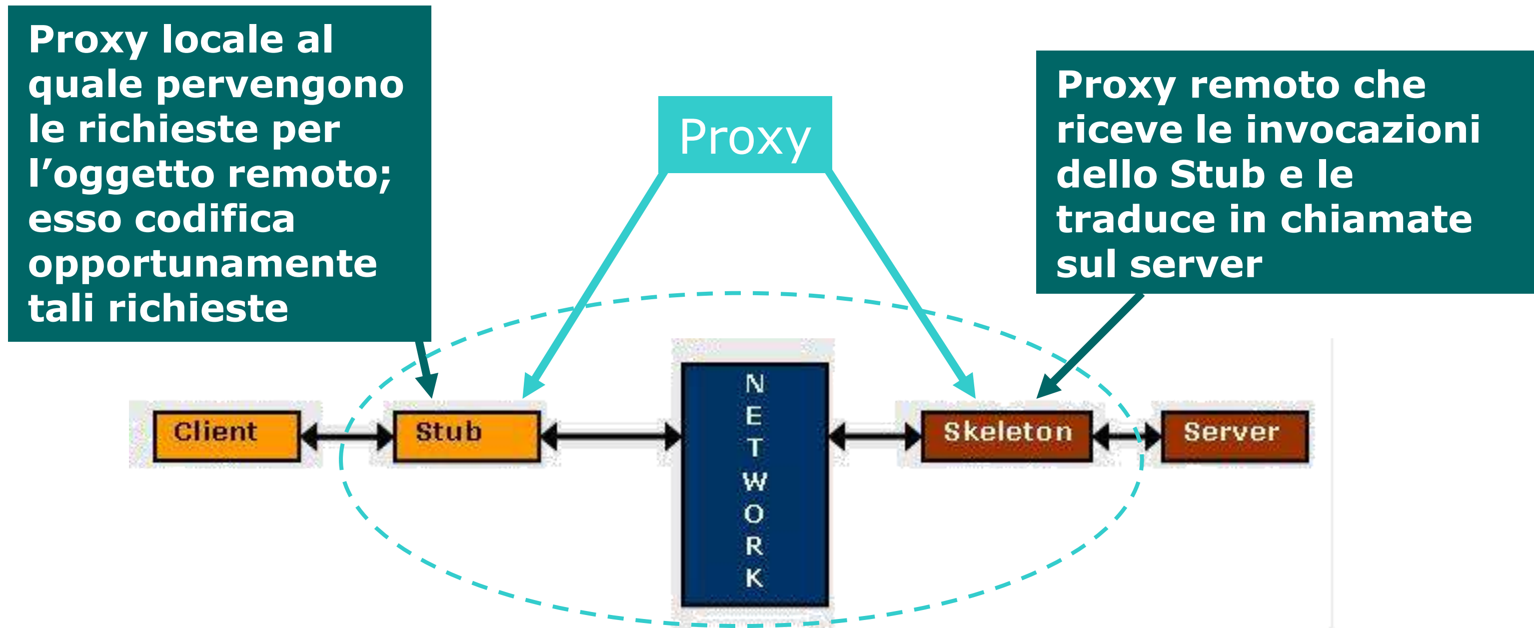
Java RMI (Remote Method Invocation)

RMI è uno dei modi che Java offre nativamente per utilizzare oggetti remoti. Tale tecnologia sfrutta proprio il **pattern Proxy** per implementare tali funzionalità

Consente di invocare un metodo di un oggetto remoto come se tale oggetto fosse locale (*ovvero appartenente allo stesso processo in cui viene eseguita l'invocazione*).

In questo senso, la tecnologia **RMI** può essere ricondotta, da un punto di vista concettuale, all'idea di **RPC** (chiamata di procedura remota) riformulata per il paradigma *object-oriented* (in cui, appunto, le *procedure* sono sostituite da *metodi*).

L’architettura RMI



Il client utilizza l’interfaccia creata da **STUB+NETWORK+SKELETON** come proxy per accedere alla risorsa presente sul server

Pattern correlati

- Decorator
  - Può avere un’implementazione molto simile a proxy anche se con differenti scopi. Decorator aggiunge man mano decorazioni ad un oggetto mentre proxy si focalizza sull’accesso allo stesso
- Adapter
  - Condivide, sicuramente l’interesse per l’accesso e l’interfaccia agli oggetti, tuttavia Adapter adatta l’interfaccia a quella dell’oggetto interessato, mentre proxy fornisce la stessa interfaccia d’accesso.

Bibliografia

- C#
- Gang of Four - Design Patterns, Elements of Reusable Object Oriented Software, Addison Wesley
- Il Pattern Proxy, Capuccini Simone 60578
- javastaff.com, soprattutto per la parte riguardante RMI
- Wikipedia