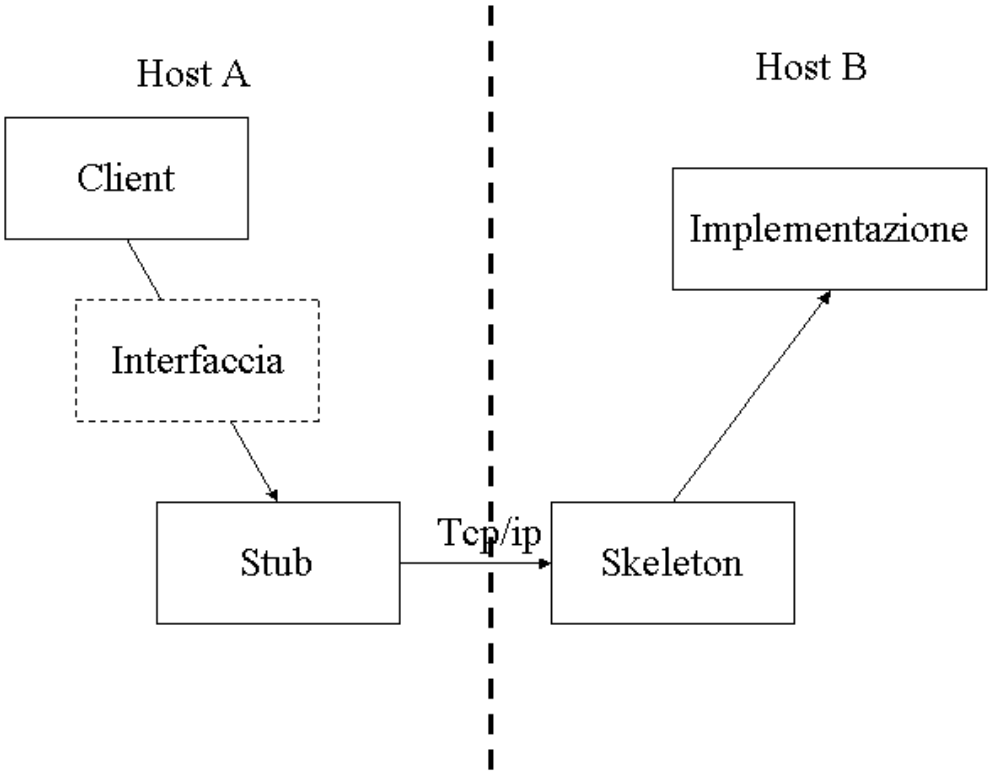
Lezione 14 - RMI

In questa lezioni vediamo una nuova API Java che permette lo scambio di messaggi (invocazione di metodi) tra due host. L'applicazione può essere una servlet, una applet o un'applicazione stand-alone.

RMI

La Remote Method Invocation è un'estensione del RPC (Remote Procedure Call). L'RPC permette a un processo che gira su una macchina A di invocare una procedura che gira su una macchina B. Allo stesso modo RMI nell'ambito della programmazione a oggetti, permette l'invio di un messaggio (o invocazione di un metodo) da una macchina A a un macchina B. Come al solito esiste un'opportuna API per semplificare il compito del programmatore che deve scrivere del codice per inviare i metodi.

In pratica, sia client che server possono quasi ignorare il fatto di risiedere su due macchine diverse.



Il minimo necessario per poter usare RMI è:

1) definire un'interfaccia con i metodi che devono essere invocati remotamente;
2) l'iterfaccia deve estendere l'interfaccia java.rmi.Remote;
3) ogni metodo dell'interfaccia deve dichiarare throws java.rmi.RemoteException;
4) bisogna implementare l'iterfaccia nell'oggetto remoto;
5) l'oggetto remoto deve avere un costruttore;
6) l'oggetto remoto deve implementare i metodi dell'interfaccia;
7) nel server bisogna creare e installare un security manager; //non serve sempre
8) nel server creare almeno un istanza dell'oggetto remoto;
9) nel server registrare uno degli oggetti remoti nel registro degli oggetti remoti;
10) nel client recuperare il riferimento all'oggetto remoto;

11) nel client invocare il metodo nell'oggetto remoto;
12) compliare i files;
13) creare gli stub e gli skeletons con rmic (RMI compiler);
14) far partire registro RMI;
15) far partire il server;
16) far partire il client;

Vediamo passo passo i punti raggruppati per colore che evidenzia funzionalità correlate.
:

Definire l'interfaccia:

```
package it.unive.dsi.labreti.rmi;

import java.rmi.*;
public interface Prova extends Remote
{
  String method(Integer i) throws RemoteException;
}
```

Implementazione dell'oggetto remoto e del server;

```
package it.unive.dsi.labreti.rmi;

import java.rmi.*;

public class ProvaImpl extends UnicastRemoteObject
implements Prova
{

Integer i;
  public ProvaImpl() throws RemoteException
  {
    super(); // esporta l'oggetto;
             // esportandolo si possono generare
eccezioni
  }

  public String method(Integer i)
  {

this.i=i;
    return "risultato;
  }

  public static void main(String[] args)
```

```
  {
    if ( System.getSecurityManager()==null)
      System.setSecurityMaanger(new
RMISecurityManager());

    try
    {
      ProvaImpl obj = new ProvaImpl();

       Naming.rebind("//host/ServerDiProva",obj);
     }
     catch(Exception e)
     {
       System.out.printl("ProvaImpl error:"+
e.getMessage())
       e.printStacjTrace();
     }
   }
}
```

Implementazione del client:

```
package it.unive.dsi.labreti.rmi;

import java.rmi.*;
public class ProvaClient
{

   public static void main(String[] args)
   {
       try
        {
       Prova obj = (Prova)
Naming.lookup("//host/ServerDiProva");
        String res = obj.method(4);
        System.out.printl(res);
       }
      catch (RemoteException re)
      {
        System.out.println("ProvaClient
error:"+re.getMessage());
         re.printStackTrace();
      }
    }
}
```

Compilazione:

```
javac Prova.java ProvaImpl.java ProvaClient.java;

rmic it.unive.dsi.labreti.rimi.ProvaImpl
```

Lanciare le applicazioni:

```
rmiregistry&
java it.unive.dsi.labreti.rmi.ProvaImpl &
java. it.unive.dsi.labreti.rmi.ProvaClient &
```

Considerazioni aggiuntive

Come abbiamo visto, i metodi invocati remotamente possono avere dei tipi di ritorno. Inoltre i metodi possono avere dei parametri e più in generale anche il tipo dell'oggetto remoto stesso può beneficiare del polimorfismo. Nasce quindi una domanda: cosa succede se la virtual machine della macchina client non conosce "il tipo reale" del valore di ritorno, diei parametri o dell'oggetto stesso. E' possibile configurare in maniera semplice il registry e il server in modo che il bytecode contente il necessario al client per conoscere il tipo del server venga "scaricato" dal client. In questo modo è possibile utilizzare il polimorfismo anche tra macchine diverse.

L'url con il quale il server collega l'oggetto al nome ha la seguente forma:

- Non è necessario specificare nessun protocollo;
- Se non viene specificato il nome o l'IP dell'host, viene assunto l'host locale;
- Si può specificare una porta se questa è diversa da la porta di default 1099;
- Per raqgioni di sicurezza un'applicazione puo collegare e scollegare (bind e unbind) oggetti solo su un registro che gira sulla stessa macchina, metre ovviamente la ricerca può avvenira anche da altri server.

Il SecurityManager è necessario solo se il client non può accedere alla classe (generata da rmic) degli stub.
Le caratteristiche offerte dalla classe base UnicastRemoteObject sono:

- usa il trasporto di default tramite socket;
- lo skeleton è sempre in esecuzione e in attesa di chiamate.

Notiamo che la sincronizzazione degli accessi non viene gestita da RMI: se ci sono due client che contemporaneamente invocano un metodo dell'oggetto remoto, nel server ci saranno due thread in esecuzione sul codice del metodo.