RMI remote method invocation

Marco Danelutto
Lab. Programmazione di Rete :: A.A. 2007-2008

Concetto RPC/RMI

- invocazione di procedura/metodo
  - o.method(params);
  - eseguita su oggetto remoto:
    - metodo per ottenere un handle
    - metodi per passare param e restituire risultati

Implementazione classica

- Chiamata:
  - marshalling dei parametri (nome del metodo e parametri attuali in pacchetto)
  - spedizione
  - unmarshalling e chiamata del metodo

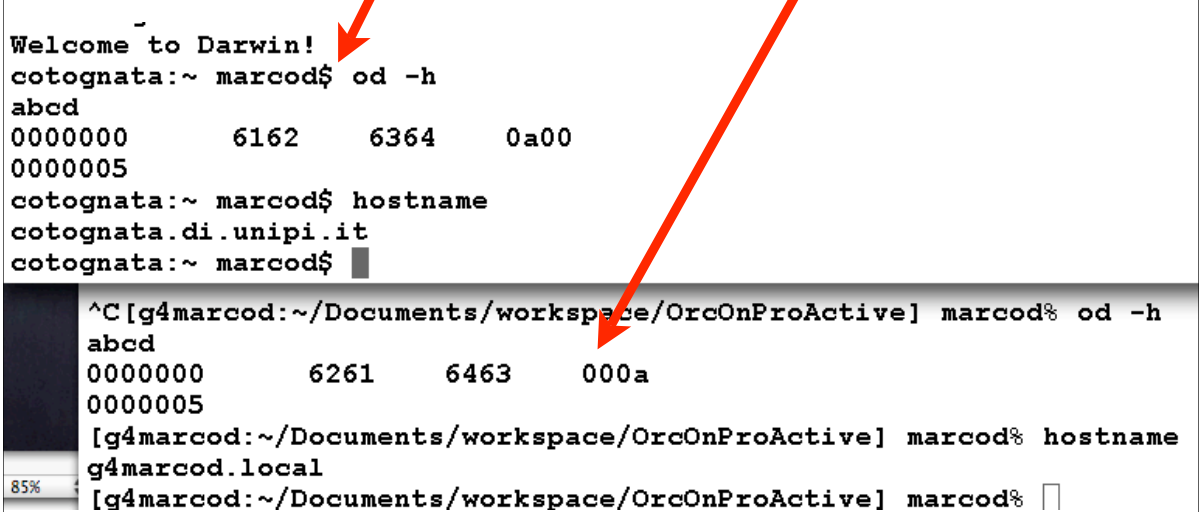
Implementazione classica

- Ritorno
  - marshalling dei risultati
  - spedizione
  - unmarshaling e setting del valore di ritorno

Rappresentazione dei dati

- XDR: eXternal Data Representation
- macchine eterogenee
  - trasformazione di messaggi in formato intermedio neutro rispetto all'architettura
  - e.g. rappresentazione di interi

Esempio: PowerPC vs Intel



Strumenti per il supporto di RPC/RMI

- IDL (Interface Description Language)
- Compilatore
  - IDL -> STUB
    - routine che sostituisce la procedura originale implementando marshalling e unmarshalling
  - IDL -> skeleton
    - server remoto

User view

- ProcedureHandle = lookup("nome");
  - generalmente risolto dal RTS
- poi
  - ris = ProcedureHandle(param1, param2, ...);

Sostituisce:

- apertura di una connessione con host remoto
- spedizione dei parametri
- ricezione del risultato
- e sua memorizzazione
  - oltre all'implementazione del server remoto ...

Java: RMI

- Remote Method Invocation
  - standard fin dalle prime versioni
  - componente server:
    - dichiarazione di oggetti accessibili da remoto
  - componente client:
    - lookup => oggetto -> chiamata di metodo

Lato server

- Interfaccia Oggetto remoto
  - pubblica per permettere accesso da third party remote
- Interfaccia:
  - extends Remote
    - marker interface :
      "sarà acceduto da remoto"

Lato server (2)

- Implementazione Interfaccia
  - oggetto da "pubblicare"
- Separazione interfaccia/implementazione è importante:
  - interfaccia è pubblica
  - implementazione non lo è

Lato server (3)

- Metodi pubblicati dall'oggetto remoto
  - devono prevedere il lancio dell'eccezione
    - RemoteException
- Esecuzione di metodi da client diversi
  - concorrente (a meno che non siano sincronizzati)

Lato server (4)

- Instaurazione di un security manager
  - controlla le operazioni relative all'RMI

java.rmi

Class RMISecurityManager

```
java.lang.Object
  └ java.lang.SecurityManager
      └ java.rmi.RMISecurityManager
```

```
public class RMISecurityManager
extends SecurityManager
```

A subclass of `SecurityManager` used by RMI applications that use downloaded code. RMI's class loader will not download any classes from remote locations if no security manager has been set. `RMISecurityManager` does not apply to applets, which run under the protection of their browser's security manager. `RMISecurityManager` implements a policy that is no different than the policy implemented by `SecurityManager`. Therefore an RMI application should use the `SecurityManager` class or another application-specific `SecurityManager` implementation instead of this class.

To use a `SecurityManager` in your application, add the following statement to your code (it needs to be executed before RMI can download code from remote hosts, so it most likely needs to appear in the `main` method of your application):

```
System.setSecurityManager(new SecurityManager());
```

Lato Server (5)

- Per rendere un oggetto che si pubblica "server RMI"
  - extends UnicastRemoteObject nella dichiarazione della classe Oggetto
  - oppure prima della pubblicazione si usa il metodo statico UnicastRemoteObject.exportObject(Object); per rendere "server" l'oggetto che poi si pubblica

Lato server (6)

- Pubblicazione dell'oggetto
  - binding
    - associazione oggetto::nome
    - realizzata mediante un registry
  - server che accetta richieste di chiamata dei metodi dell'oggetto pubblicato
    - realizzata dal supporto a run time

Lato client

- Instaurazione di un security manager
  - controlla le operazioni relative all'RMI
  - come per il lato server
    - serve in caso di download di codice/ classi dal server

Lato client (2)

- Ottenere un'handle per l'oggetto remoto
  - operazione di lookup
    - nome di pubblicazione/bind
    - nome della macchina con il registry
  - tipo dell'handle :: interfaccia Oggetto remoto
    - altrimenti devo poter accedere

Lato client (3)

- chiamata di metodo sull'handle
  - risultato = handle.metodo(params ...);
- trasparenza della chiamata rispetto alla locazione dell'oggetto
  - dettagli nascosti nella lookup

Sample code (server)

```
public interface OggettoRemotoInterface extends Remote {
  public int f(int x);
}

public class OggettoRemoto implements OggettoRemoteInterface {
  public int f(int x) {
     ...
  }
}

public class Server {
  public static void main(String [] args) {
    System.setSecurityManager(new RMISecurityManager());
    OggettoRemoto o = new OggettoRemoto();
    Naming.rebind("servizioF",o);
  }
}
```

Sample code (client)

```
public class Client {
  public static void main(String [] args) {

        System.setSecurityManager(new RMISecurityManager());
        OggettoRemotoInterface o = (OggettoRemotoInterface)
          Naming.lookup("rmi://hostregistry/servizioF");

          int i = o.f(1234);

  }
}
```

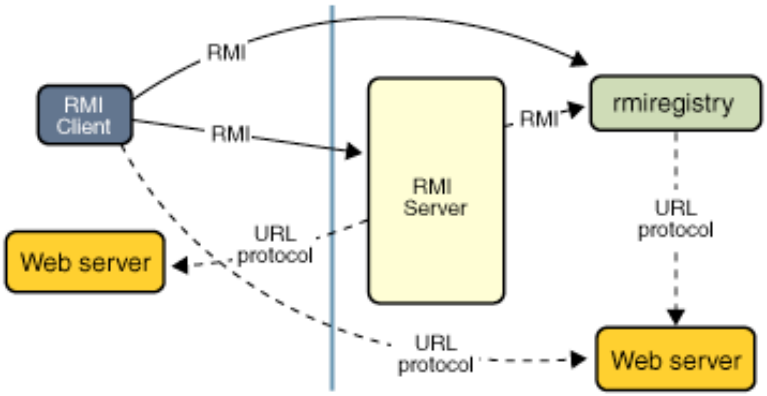
Tools (lato server)

- Compilazione server (javac)
  - precedentemente alla 1.5: rmic
- rmiregistry &
- lancio del server (java)

Tools (lato client)

- compilazione client (javac, serve interfaccia)
- run client (java)

Caricamento dinamico



http://java.sun.com/docs/books/tutorial/rmi/overview.html

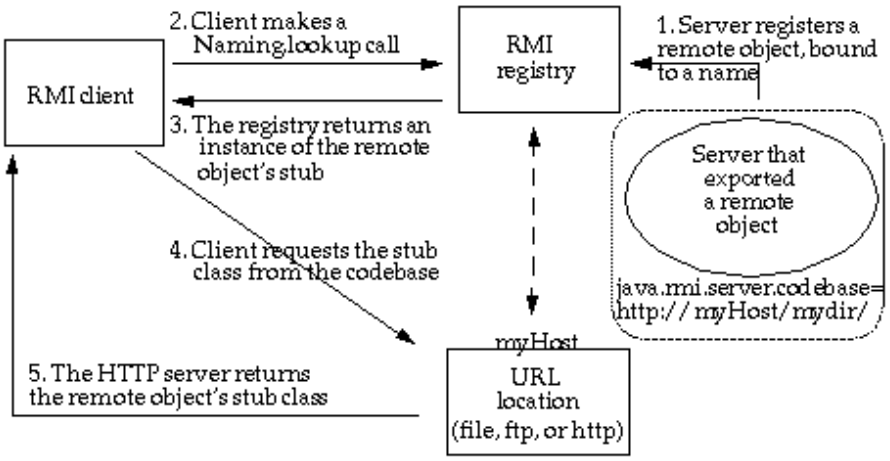
Concetto di codebase

- locazione dove si vanno a cercare le classi che debbono essere caricate a seguito di chiamate RMI
  - file, HTTP server, ...
- proprietà
  - -Djava.rmi.server.codebase=...

Codebase (2)

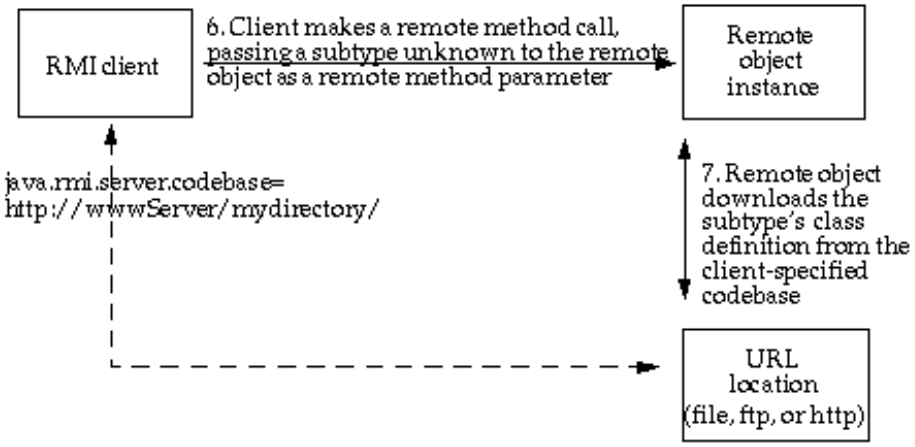
- file system (condiviso mediante NFS)
  - file://home/m/marcod/classiRmi
- HTTP server (accessibile via rete)
  - http://www.cli.di.unipi.it/~marcod/classes
    - => $HOME/public_html/classes

Funzionamento RMI



http://java.sun.com/j2se/1.4.2/docs/guide/rmi/codebase.html

Uso codebase



http://java.sun.com/j2se/1.4.2/docs/guide/rmi/codebase.html

Security policy

- modo per garantire capabilites al codice installato a seguito di operazioni RMI
- esempio: file "policy" garantisce tutto:

```
grant {
   permission java.security.AllPermissions;
};
```

- java -Djava.security.policy=policy

Security policy (2)

- capabilities associate a codebase:

```
grant codebase "file:/..." {
   permission java.security.XXX;
};
```

Registry

- comando: rmiregistry [portnumber]
- porta di default : 1099
  - significa che si può fare lookup solo se la porta è aperta
- normalmente lanciato da linea di comando
  - in background

Registry (2)

- ATTENZIONE:
  - registrazione di oggetti controllata da diritti utente
  - registry lanciato da utente marcod NON può accettare bind di oggetti da utente rossipaolo

Registry (3)

- si può lanciare, accedere da programma
- reg = LocateRegistry.getRegistry (host,porta);
  o = (OggettoRemotoInterface)
  reg.lookup("servizioF");
- getRegistry(...) trova registro esistente
- createRegistry(int port) ne crea uno

Registry (4)

- per evitare di lasciare processi attivi
  - LocateRegistry.createRegistry(porta)
  - quando il programma termina, termina anche il registry => libera la porta per un altro utente !!!

Terminazione server

- main del programma server:

```
public class Server {
  public static void main(String [] args) {
    System.setSecurityManager(new RMISecurityManager());
    OggettoRemoto o = new OggettoRemoto();
    Naming.rebind("servizioF",o);
  }
}
```

- non termina: thread RMI server attivi ...

Terminazione controllata

- reb.bind(nomeoggetto, oggetto);
- reg.unbind(nomeoggetto);
  - rimozione dell'associazione (ancora possibile utilizzare handle esistenti)
- UnicastRemoteObject.unexportObject (oggetto, true);
  - spegnimento thread server RMI

Filosofia generale

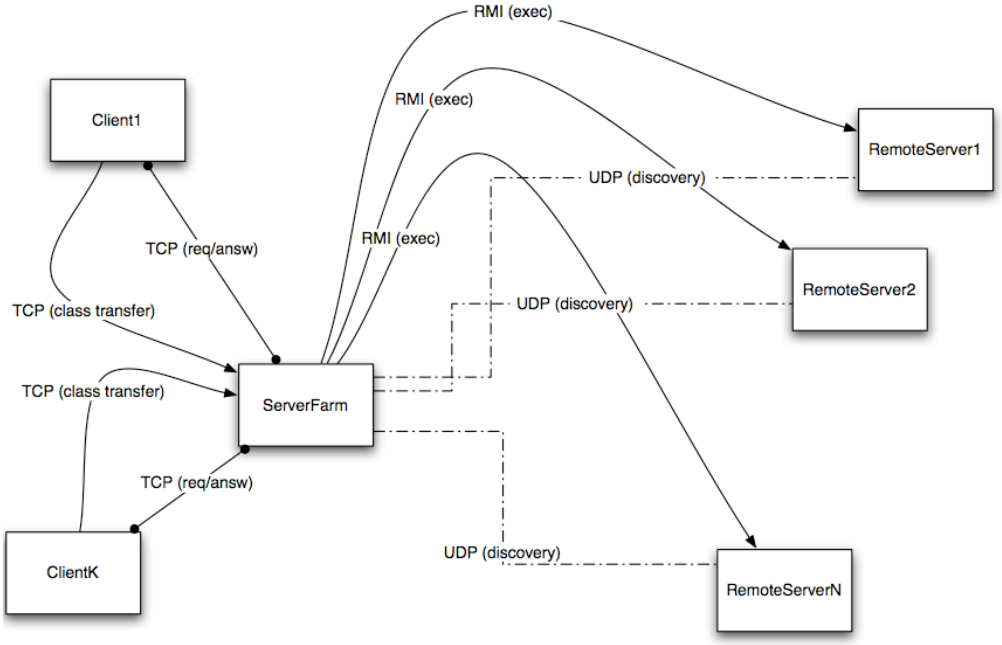
- metto a disposizione un servizio
- rendo pubblica l'interfaccia e la locazione del registry
- clienti in grado di accedere il servizio da remoto
- eventualmente utilizzando un codebase opportuno

Es: Servizi generici

-
-
-
-

http://java.sun.com/docs/books/tutorial/rmi/overview.html

Esempio: progetto 06/07



Risorse in rete

- Tutorial SUN su RMI
  - http://java.sun.com/docs/books/tutorial/rmi/index.html
- Caricamento dinamico di codice
  - http://java.sun.com/j2se/1.4.2/docs/guide/rmi/codebase.html