 **Sistemi distribuiti: il modello Java RMI**

Il modello Java RMI

SCD

Anno accademico 2008/9
Sistemi Concorrenti e Distribuiti

Tullio Vardanega, tullio.vardanega@math.unipd.it

Corso di Laurea Magistrale in Informatica, Università di Padova 1/23

 **Sistemi distribuiti: il modello Java RMI**

Architettura del modello – 1

- ❑ **Oggetto remoto come sola unità di distribuzione**
 - L'interfaccia può essere resa accessibile a processi remoti
 - Lo stato risiede sempre su un singolo nodo
- ❑ **Oggetto remoto ≠ oggetto locale**
 - 1) **Rispetto alla clonazione**
 - Solo il suo *servant* può clonare un oggetto remoto
 - L'oggetto viene creato nello spazio di indirizzamento dello stesso *servant*
 - Ma i *proxy* dell'oggetto remoto originale **non** vengono clonati
 - Il cliente che volesse utilizzare il clone deve localizzarlo come tale e connettersi esplicitamente


Corso di Laurea Magistrale in Informatica, Università di Padova 2/23

 **Sistemi distribuiti: il modello Java RMI**

Architettura del modello – 2

- ❑ **Oggetto remoto ≠ oggetto locale**
 - 2) **Rispetto alla mutua esclusione**
 - La presenza di metodi *synchronized* su un oggetto remoto **non** garantisce mutua esclusione tra processi clienti che risiedono su nodi **distinti**
 - Ogni *proxy* di oggetto remoto garantisce infatti mutua esclusione **solo** ai processi che risiedono sullo stesso nodo del *proxy*
 - 3) **Rispetto ai parametri passati ai metodi**
 - Il tipo dell'oggetto passato come parametro a RMI deve agevolare forme standard di *marshalling* e *unmarshalling* → parametro di tipo **serializable**
 - Quasi tutto in Java lo è tranne i tipi che dipendono dall'istanza di JVM (p.es. Thread, descrittori di *file, socket*) o che sono inerentemente "insicuri" (p.es. *FileInputStream*)
 - 4) **Rispetto al passaggio dell'oggetto come parametro**
 - Oggetto locale → per valore (con la modalità *deep copy* tipica di Java)
 - Oggetto remoto → per riferimento


Corso di Laurea Magistrale in Informatica, Università di Padova 3/23

 **Sistemi distribuiti: il modello Java RMI**

Architettura del modello – 3

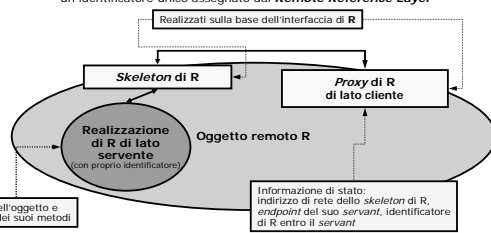
- ❑ **Oggetto remoto e locale sono indistinguibili a meno delle 4 differenze citate**
- ❑ **Riferimento all'oggetto remoto**
 - **Indirizzo di rete (IP), *end point* del servente, modalità di comunicazione (*protocol stack*)**
 - Usato dal *proxy* (chiamato *stub* dallo standard Java)
 - **Identificatore locale dell'oggetto nello spazio del servente**
 - Usato esclusivamente dal lato servente

Corso di Laurea Magistrale in Informatica, Università di Padova 4/23


 **Sistemi distribuiti: il modello Java RMI**

Architettura del modello – 4

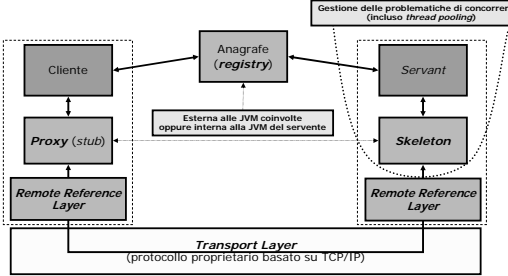
Il *proxy* converte ogni invocazione di metodo di R in un messaggio usando *socket* per creare una connessione TCP temporanea verso il nodo destinatario (oppure meglio utilizzandone una esistente) identificando l'oggetto remoto con un identificatore unico assegnato dal **Remote Reference Layer**



Corso di Laurea Magistrale in Informatica, Università di Padova 5/23

 **Sistemi distribuiti: il modello Java RMI**

Architettura del modello – 5



Corso di Laurea Magistrale in Informatica, Università di Padova 6/23

Sistemi distribuiti: il modello Java RMI

Architettura del modello – 6

- **Proxye skeleton** si fanno carico trasparentemente del *marshalling* e dell' *unmarshalling* tramite meccanismi nativi di "serializzazione"
 - `writeObject()` → metodo di `ObjectOutputStream`
 - `readObject()` → metodo di `ObjectInputStream`
- Solo per istanze di oggetti "serializzabili"
 - Non viene trasferito l'oggetto ma solo le informazioni che ne caratterizzano l'istanza così da poterla riprodurre a destinazione
 - Né stato (`static`, `transient`) né costanti né metodi
 - Serve poter accedere al `class` originario!
 - Oggetti parametro e tipi primitivi sono passati per *deep copy*
 - Tranne quelli strettamente legati al nodo di residenza (p.es. il *servant*) che sono passati esclusivamente per riferimento

Corso di Laurea Magistrale in Informatica, Università di Padova 7/23

Sistemi distribuiti: il modello Java RMI

Architettura del modello – 7

- Il **proxy** stesso è serializzabile
 - Può essere passato come parametro (per valore) e usato dal ricevente come riferimento all'oggetto remoto
 - Poiché tutto esegue su JVM standard non occorre copiare il codice del *proxy* presso il cliente → basta indicare le classi che occorrono per rigenerarlo a destinazione
 - Mobilità **forte** rispetto allo stato
 - Legame **debole** rispetto al codice come risorsa (*binding by value*)
- L'uso di RMI pertanto consente migrazione di copia del **proxy** verso il chiamante

Corso di Laurea Magistrale in Informatica, Università di Padova 8/23

Sistemi distribuiti: il modello Java RMI

Architettura del modello – 8

- Il **proxy** riceve la chiamata del cliente
 - La "reifica" (serializzandola)
 - E poi la invia al suo RRL tramite il metodo `invoke()` di `java.rmi.server.RemoteRef`
- Lo **skeleton** riceve la chiamata remota come parametro del metodo `dispatch()` invocato dal suo RRL
 - La deserializza
 - E poi la effettua localmente al (riferimento del) *servant*

Corso di Laurea Magistrale in Informatica, Università di Padova 9/23

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 1

- L'oggetto remoto deriva da una interfaccia pubblica che estende `java.rmi.Remote`

```
import java.rmi.*;
public interface Echo extends Remote {
    String call (String message) throws RemoteException;
}
```
- Ogni suo metodo può emettere eccezione `java.rmi.RemoteException`
 - Semantica *at-most-once*
- Ogni uso dell'oggetto come argomento o valore di ritorno ha il tipo dell'interfaccia e non della sua realizzazione concreta

Corso di Laurea Magistrale in Informatica, Università di Padova 10/23

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 2

- Il **servant** dell'oggetto remoto deve
 - Estendere `java.rmi.UnicastRemoteObject`
 - Realizzare i metodi dell'oggetto remoto
 - Definire esplicitamente un costruttore dell'oggetto che possa emettere eccezione `java.rmi.RemoteException`

```
import java.rmi.*;
import java.rmi.server.*;
public class EchoServer extends UnicastRemoteObject implements Echo {
    public EchoServer (String name) throws RemoteException {
        Naming.rebind (name, this);
    }
    public String call (String message) throws RemoteException {
        return "From EchoServer: " + message + ["* * message *"];
    }
    public static void main (String args[]) {
        // il main è nel servente, che può anche essere distinto dalla
        // classe che realizza l'oggetto remoto
    }
}
```

Invocabile solo localmente al nodo di residenza del registry!

Corso di Laurea Magistrale in Informatica, Università di Padova 11/23

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 3

- La logica del servente è specificata nel suo `main` che crea istanze dell'oggetto remoto
- Ogni istanza deve essere registrata presso l'anagrafe degli oggetti remoti del nodo che viene mantenuto da un processo dedicato (`rmiregistry`)
 - `Naming.bind` lega un nome (stringa URL) all'oggetto remoto (al suo riferimento) in una associazione unica e non modificabile
 - `Naming.rebind` crea una nuova associazione (nome, riferimento) anche sovrascrivendo quella precedente
- Il gestore del registro (*name server* locale) ascolta su una porta assegnata (*default*: 1099)

Corso di Laurea Magistrale in Informatica, Università di Padova 12/23

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 4

- Il *name server* può essere attivato a parte
 - `start rmiregistry [portnumber]` ← Win32
 - `rmiregistry [portnumber] &` ← GNU/Linux
- Una singola istanza di *NS* opera per conto di tutti i server di oggetti remoti del nodo
- Le varie componenti (interfaccia, oggetto remoto, server, cliente) vengono compilate in 2 fasi distinte → `javac` e `rmic`

Corso di Laurea Magistrale in Informatica, Università di Padova 13/23

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 5

- Il cliente dell'oggetto remoto
 - Fa *look-up* presso il NS locale tramite
 - `Naming.lookup (String name)`
 - *name* è l'URL della specifica dell'oggetto remoto (la sua interfaccia pubblica) presso il nodo e la porta dove è in ascolto il NS
 - Ottenendo un riferimento con il tipo dell'interfaccia e non della classe che lo realizza!
- Da ora in avanti l'oggetto remoto è indistinguibile da uno locale

Corso di Laurea Magistrale in Informatica, Università di Padova 14/23

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 6

- `rmic` (compilatore Java RMI) 1/2
 - Genera *stub* e *skeleton* per oggetti remoti a partire dalle classi compilate che ne contengono la realizzazione
 - Le classi compilate di partenza devono essere identificate rispetto ai *package* che le contengono

```
graph LR; A(EchoServer.java) -- javac --> B(EchoServer.class); B -- rmic --> C(EchoServer_Skel.class); B -- rmic --> D(EchoServer_Stub.class);
```

Corso di Laurea Magistrale in Informatica, Università di Padova 15/23

Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 7

- `rmic` (compilatore Java RMI) 2/2
 - Lo *skeleton* è una entità di lato server che contiene un metodo che recepisce le chiamate remote all'oggetto e le indirizza verso la sua istanza concreta
 - Il protocollo utilizzato è specifico di Java RMI (`JRMP`)
 - Lo *stub* è il *proxy* dell'oggetto remoto che indirizza le chiamate a esso verso il server corrispondente
 - Il riferimento all'oggetto remoto in possesso del cliente riferisce in realtà lo *stub* dell'oggetto ossia il *proxy* locale al cliente
 - Lo *stub* di lato server riproduce le chiamate remote in ingresso e le gira localmente allo *skeleton* corrispondente

Corso di Laurea Magistrale in Informatica, Università di Padova 16/23

Sistemi distribuiti: il modello Java RMI

Applicazione del modello – 1

- La JVM consente di caricare dinamicamente codice (in forma di *bytecode*) Java da qualsiasi URL
 - Capacità utilizzabile da RMI
- Le classi locali vengono normalmente caricate a partire dalla locazione `CLASSPATH`
- Le classi remote possono essere caricate a partire dall'URL `codebase`
 - Locazione configurata come proprietà
`java -Djava.rmi.server.codebase=file://<path>/`

Corso di Laurea Magistrale in Informatica, Università di Padova 17/23

Sistemi distribuiti: il modello Java RMI

Applicazione del modello – 2

- L'accesso a classi sconosciute può essere regolato da un gestore della sicurezza
 - Lato server → consentire la copia di proprie classi
 - Lato cliente → impedire l'accesso a siti non affidabili
- Accesso regolamentato da specifica politica configurata in un *file* passato come proprietà
 - `java -Djava.security.policy = <policy_file>`

```
grant {
  permission java.io.FilePermission "<<ALL FILES>>", "read";
  permission java.net.SocketPermission "*" : 1234, "accept, connect, listen, resolve";
  permission java.lang.RuntimePermission "accessClassInPackage.sun.jdbc.odbc";
  permission java.util.PropertyPermission "file.encoding", "read";
};
```

Corso di Laurea Magistrale in Informatica, Università di Padova 18/23

Sistemi distribuiti: il modello Java RMI

Esempio: *servant*

```

package echo;
public interface Echo extends java.rmi.Remote {
    String call (String message) throws java.rmi.RemoteException;
}

package echo; import java.rmi.*; import java.rmi.server.*;
public class EchoServer extends UnicastRemoteObject implements Echo {
    public EchoServer (String name) throws RemoteException {
        try { Naming.rebind (name,this); } catch (Exception e) {
            System.out.println ("Exception in EchoServer: " + e.getMessage());
            e.printStackTrace(); }
    }
    public String call (String message) throws RemoteException {
        System.out.println("Echo's method call invoked: [" + message + "]*");
        return "From EchoServer:- Thanks for your message: [" + message + "]*";
    }
    public static void main (String args[]) throws Exception {
        if (System.getSecurityManager() == null)
            System.setSecurityManager ( new RMISecurityManager() );
        String url = "rmi://" + args[0] + "/Echo";
        EchoServer echo = new EchoServer (url);
        System.out.println("EchoServer ready!");
    }
}
    
```

Corso di Laurea Magistrale in Informatica, Università di Padova 19/23

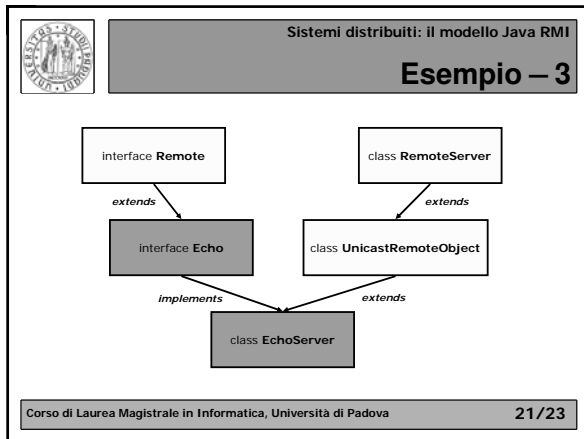
Sistemi distribuiti: il modello Java RMI

Esempio: *cliente*

```

package echo; import java.rmi.*; import java.rmi.server.*;
public class EchoClient {
    public static void main (String args[]) {
        int i;
        if (System.getSecurityManager() == null)
            System.setSecurityManager ( new RMISecurityManager() );
        try {
            System.out.println ("EchoClient ready!");
            String url = "rmi://" + args[0] + "/Echo";
            System.out.println ("Looking up remote object " + url + "...");
            Echo echo = (Echo) Naming.lookup (url);
            String toMsg = (String) args[1];
            for (i = 1; i < 6; i++) {
                toMsg = toMsg + "-" + i;
                System.out.println ("Message " + i + " to Echo: [" + toMsg + "]*");
                String fromMsg = echo.call (toMsg);
                Thread.sleep (2000);
                System.out.println ("Message from Echo: \n\t" + fromMsg + "\n");
            }
        } catch (Exception e) {
            System.out.println ("Exception in EchoClient: " + e.getMessage());
            e.printStackTrace(); }
    }
}
    
```

Corso di Laurea Magistrale in Informatica, Università di Padova 20/23



Sistemi distribuiti: il modello Java RMI

Esempio – 4

Generato staticamente per motivi di retrocompatibilità

```

javac -d . Echo.java
javac -d . EchoServer.java
javac -d . EchoClient.java
rmic -d . echo.EchoServer
    
```

Generato dinamicamente a partire JDK 1.2

Nel package echo

```

EchoServer_Skel.class
EchoServer_Stub.class
    
```

Attivazione del name server di lato servente sul nodo localhost alla porta 1234

```

rmiregistry 1234 &
    
```

Attivazione del servente con parametro indicante l'endpoint del name server

```

java -classpath . -Djava.security.policy=pol.policy
echo.EchoServer localhost:1234
    
```

Attivazione del cliente con parametro indicante l'endpoint del name server

```

java -classpath . -Djava.security.policy=pol.policy
echo.EchoClient localhost:1234 Initial_Message
    
```

Corso di Laurea Magistrale in Informatica, Università di Padova 22/23

