



Il modello Java RMI

SCD

Anno accademico 2013/14
Sistemi Concorrenti e Distribuiti

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

Laurea Magistrale in Informatica, Università di Padova1 / 31



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 2

❑ Oggetto remoto oggetto locale / I

○ Rispetto alla clonazione

- Solo il *servant* (servente di oggetto) può clonare un oggetto remoto
- L'oggetto viene creato nello spazio di indirizzamento del *servant*
- Ma i *proxy* dell'oggetto remoto originale non vengono clonati con esso
- Il cliente che volesse utilizzare il clone deve localizzarlo come tale e connettersi esplicitamente con esso

○ Rispetto alla mutua esclusione

- L'accesso concorrente a metodi di oggetto remoto è sempre intrinsecamente possibile: ogni *proxy* di oggetto remoto garantisce infatti mutua esclusione solo ai chiamanti residenti sul suo stesso nodo
- Se il metodo remoto non è protetto da **synchronized** l'esecuzione è esposta a rischio di *data race*

Laurea Magistrale in Informatica, Università di Padova3 / 31



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 1

❑ L'oggetto remoto è l'unità di distribuzione

○ L'interfaccia è accessibile a clienti remoti

- Tramite l'implementazione in un particolare oggetto

○ Lo stato risiede sempre su un singolo nodo

- Presso l'oggetto che implementa l'interfaccia remoto

❑ Ma il modello non riesce a garantire trasparenza totale!

Laurea Magistrale in Informatica, Università di Padova2 / 31



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 3

❑ Oggetto remoto oggetto locale / II

○ Rispetto ai parametri passati ai metodi

- Il tipo dell'oggetto passato come parametro a RMI deve permettere *marshalling* e *unmarshalling* → deve essere di tipo *serializable*
- Quindi no ai tipi dipendenti dall'istanza di JVM (p.es. *Thread*, descrittori di *file*, *socket*) o quelli inerentemente "insicuri" (p.es. *FileInputStream*)

○ Rispetto al passaggio dell'oggetto come parametro

- Oggetto locale → passaggio per valore in modalità *deep copy* (non by-ref !)
- Oggetto remoto → per riferimento

❑ L'oggetto remoto è uguale all'oggetto locale a meno di queste 4 differenze

Laurea Magistrale in Informatica, Università di Padova4 / 31



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 4

❑ Riferimento all'oggetto remoto

○ Indirizzo IP del nodo di residenza dello *skeleton*

○ *Endpoint* 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

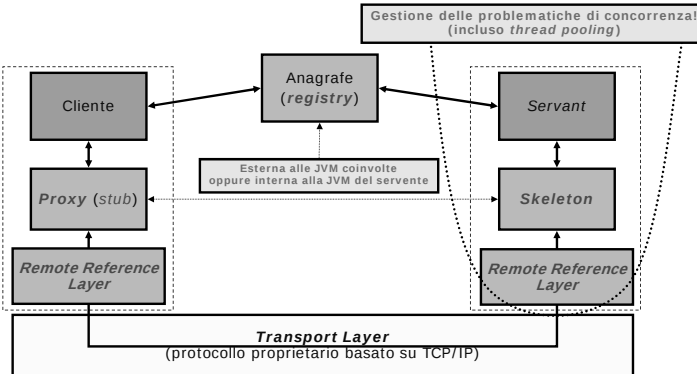
Laurea Magistrale in Informatica, Università di Padova

5 / 31



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 5



Laurea Magistrale in Informatica, Università di Padova

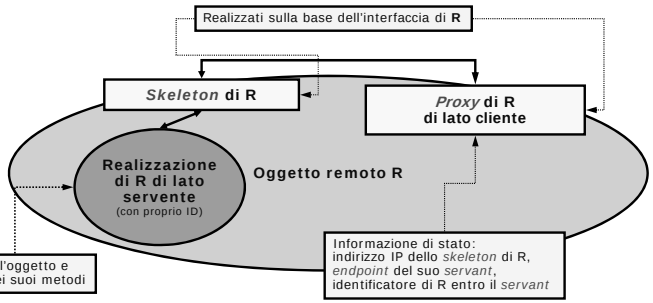
7 / 31



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 4

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



Realizzati sulla base dell'interfaccia di R

Skeleton di R

Proxy di R di lato cliente

Realizzazione di R di lato servente (con proprio ID)

Oggetto remoto R

Stato dell'oggetto e codice dei suoi metodi

Informazione di stato: indirizzo IP dello *skeleton* di R, endpoint del suo *servant*, identificatore di R entro il *servant*

Laurea Magistrale in Informatica, Università di Padova

6 / 31



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 6

❑ *Proxy* e *skeleton* si fanno carico trasparentemente del *marshalling* e dell'*unmarshalling* tramite meccanismi nativi di “serializzazione”

○ `writeObject()` – metodo di `ObjectOutputStream`

○ `readObject()` – metodo di `ObjectInputStream`

Laurea Magistrale in Informatica, Università di Padova

8 / 31

UniPD - SCD 2013/14 - Sistemi Concorrenti e Distribuiti

2



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 7

❑ Tutto questo quindi solo per oggetti “serializzabili”

- Non viene trasferito l'oggetto ma solo le informazioni che ne caratterizzano l'istanza così da poterla riprodurre a destinazione
- Passaggio «*by-value*» tramite il `.class` originario → come farlo passare?
 - Oggetti parametro e tipi primitivi sono passati per *deep copy*
- Passaggio «*by reference*» per quelli legati al nodo di residenza

Laurea Magistrale in Informatica, Università di Padova

9 / 31



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 9

❑ 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*

Laurea Magistrale in Informatica, Università di Padova

11 / 31



Sistemi distribuiti: il modello Java RMI

Architettura del modello – 8

❑ 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 consente migrazione di copia del *proxy* verso il chiamante

Laurea Magistrale in Informatica, Università di Padova

10 / 31



Sistemi distribuiti: il modello Java RMI

A look under the hood – 1

1. The servant creates an instance of the remote object which extends *UnicastRemoteObject*
2. The constructor for *UnicastRemoteObject* makes the remote object able and available to service incoming RMI calls
 - A TCP socket bound to an arbitrary port number is created
 - A thread is also created to listen for connections on that socket
3. The servant registers the remote object with the RMI registry handing it the corresponding stub (*proxy*)
 - The stub contains the information needed to “call back” to the servant when it is invoked

Laurea Magistrale in Informatica, Università di Padova

12 / 31



Sistemi distribuiti: il modello Java RMI

A look under the hood – 2

4. A client obtains the stub by calling the RMI registry

- If the server specified a "codebase" for clients to obtain the classfile for the stub, that information will also be passed to the client via the registry
- The client can then use the codebase to resolve the stub class to load the stub classfile
- All the RMIRegistry does is to hold onto remote object stubs to hand them off to clients when requested

Laurea Magistrale in Informatica, Università di Padova

13 / 31



Sistemi distribuiti: il modello Java RMI

A look under the hood – 4

- 8. When a client connects to the servant's socket a new thread is forked on the servant's side to service the incoming call
 - The original thread can continue listening to the original socket so that new calls from other clients can be made
- 9. The servant reads the RMI header information and creates a RemoteCall to unmarshal the incoming RMI arguments
- 10. The servant calls the "dispatch" method of the skeleton class which calls the target method on the object and pushes the result back to the socket
- 11. On the client side the return value of the RMI is unmarshaled and returned from the proxy back to the client code

Laurea Magistrale in Informatica, Università di Padova

15 / 31



Sistemi distribuiti: il modello Java RMI

A look under the hood – 3

- 5. When the client issues an RMI to the servant the stub class creates a "RemoteCall"
 - This opens a socket to the servant on the port specified in the stub and sends the RMI header information to it
- 6. The stub class marshals the arguments over the connection by using RemoteCall methods which serializes them into a Java object
- 7. The stub class calls RemoteCall.executeCall to cause the RMI to happen

Laurea Magistrale in Informatica, Università di Padova

14 / 31



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



Laurea Magistrale in Informatica, Università di Padova

16 / 31



Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 2

□ Il *servant* dell'oggetto remoto deve

- Estendere `java.rmi.UnicastRemoteObject`
- Fornire implementazione dei metodi dell'oggetto
- Definire esplicitamente un costruttore che possa emettere eccezione `java.rmi.RemoteException`

Invocabile solo localmente al nodo di residenza del registry!

```
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 }
    }
```

Laurea Magistrale in Informatica, Università di Padova

17 / 31



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 serventi di oggetti remoti del nodo

□ Le varie componenti (interfaccia, oggetto remoto, servente, cliente) vengono compilate in 2 fasi distinte → `javac` e `rmic`

Laurea Magistrale in Informatica, Università di Padova

19 / 31



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)

Laurea Magistrale in Informatica, Università di Padova

18 / 31



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'URI 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!
- Il NS ha un `ObjID` riservato → *closure* implicita

□ Da ora in avanti l'oggetto remoto è indistinguibile da uno locale

Laurea Magistrale in Informatica, Università di Padova

20 / 31



Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 6

- ❑ *For security reasons an application can only bind, unbind, or rebind remote object references with a registry running on the same host*
 - *This restriction prevents a remote client from removing or overwriting any of the entries in a server's registry*
- ❑ *A look-up however can be requested from any host, local or remote*
 - “Niente impedisce di esporre esternamente una interfaccia remota che registri localmente oggetti realizzati in altri nodi”

Laurea Magistrale in Informatica, Università di Padova

21 / 31



Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 8

- ❑ **rmic (compilatore Java RMI) / I**
 - Lo *skeleton* è una entità di lato servente 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 servente 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 servente riproduce le chiamate remote in ingresso e le gira localmente allo *skeleton* corrispondente

Laurea Magistrale in Informatica, Università di Padova

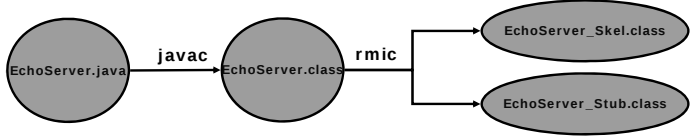
23 / 31



Sistemi distribuiti: il modello Java RMI

Utilizzo del modello – 7

- ❑ **rmic (compilatore Java RMI) / I**
 - 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);
```

Laurea Magistrale in Informatica, Università di Padova

22 / 31



Sistemi distribuiti: il modello Java RMI

Applicazione del modello – 1

- ❑ La JVM consente di caricare dinamicamente codice Java (in forma di *bytecode*) 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>/

Laurea Magistrale in Informatica, Università di Padova

24 / 31



Sistemi distribuiti: il modello Java RMI

Applicazione del modello – 2

❑ L'accesso a classi sconosciute può essere regolato da un gestore della sicurezza

○ Lato servente → consentire la copia di proprie classi

○ Lato cliente → impedire l'accesso a siti non affidabili

❑ Accesso regolato da 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 "accessClassPackage.sun.jdbc.odbc";
    permission java.util.PropertyPermission "file.encoding", "read";
};
```

Laurea Magistrale in Informatica, Università di Padova

25 / 31



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();
        }
    }
}
```

Laurea Magistrale in Informatica, Università di Padova

27 / 31



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!");
    }
}
```

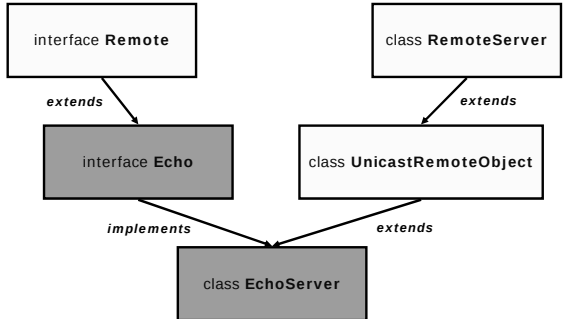
Laurea Magistrale in Informatica, Università di Padova

26 / 31



Sistemi distribuiti: il modello Java RMI

Esempio – 3



```
graph TD
    Remote[interface Remote] -- extends --> Echo[interface Echo]
    RemoteServer[class RemoteServer] -- extends --> UnicastRemoteObject[class UnicastRemoteObject]
    EchoServer[class EchoServer] -- implements --> Echo
    EchoServer -- extends --> UnicastRemoteObject
```

Laurea Magistrale in Informatica, Università di Padova

28 / 31

UniPD - SCD 2013/14 - Sistemi Concorrenti e Distribuiti

7



Sistemi distribuiti: il modello Java RMI

Esempio – 4

Generato staticamente per motivi di retrocompatibilità

Generato dinamicamente a partire JDK 1.2

Nel package echo

EchoServer_Skel.class

EchoServer_Stub.class

Compilazione

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

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
```

Laurea Magistrale in Informatica, Università di Padova

29 / 31



Sistemi distribuiti: il modello Java RMI

RMI e concorrenza – 2

❑ L'invocazione di lato cliente è chiaramente bloccante

○ Quindi uno stesso cliente non può inviare richieste concorrenti allo stesso oggetto remoto

○ Quindi il proxy di quel cliente non rischia data race

❑ Ogni invocazione in arrivo sul nodo del servente è ricevuta da un thread distinto

○ Quindi una implementazione thread-safe dell'oggetto remoto deve gestire la sincronizzazione

Laurea Magistrale in Informatica, Università di Padova

31 / 31



Sistemi distribuiti: il modello Java RMI

RMI e concorrenza – 1

❑ RMI Spec @ 3.2 Thread Usage in RMI

○ A method dispatched by the RMI runtime to a remote object implementation may or may not execute in a separate thread

○ The RMI runtime makes no guarantees with respect to mapping invocations to threads

○ Since remote method invocation on the same remote object may execute concurrently, a remote object implementation needs to make sure its implementation is thread-safe

- "it's your problem, baby"

Laurea Magistrale in Informatica, Università di Padova

30 / 31

UniPD - SCD 2013/14 - Sistemi Concorrenti e Distribuiti

8