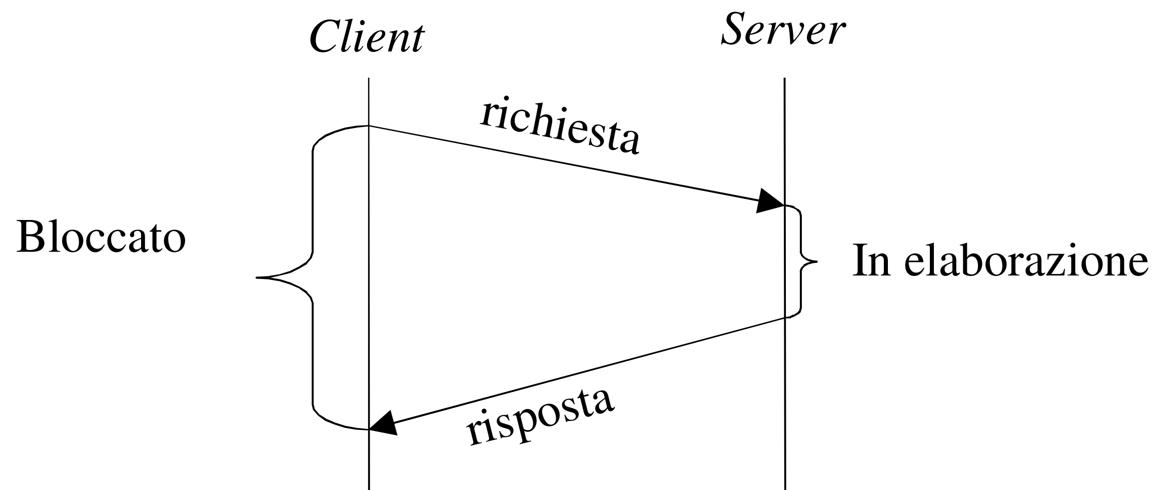
LPR 2005/2006
Lezione 7

- paradigma di interazione domanda/risposta
- remote procedure call
- RMI (Remote Method Invocation): API JAVA
- esercizio

PARADIGMA DI INTERAZIONE A DOMANDA/RISPOSTA

Paradigma di interazione basato su richiesta/risposta

- il client invia ad un server un *messaggio di richiesta*
- il server risponde con un *messaggio di risposta*
- il client rimane bloccato (sospende la propria esecuzion) finchè non riceve la risposta dal server



PARADIGMA DI INTERAZIONE DI TIPO DOMANDA/RISPOSTA

Esempio interazione domanda/risposta:
un client richiede ad un server la stampa di un messaggio. Il server restituisce al client un codice che indica l'esito della operazione. Il client attende l'esito della operazione

Implementazioni possibili:

- invio messaggi di *richiesta/risposta* su una connessione TCP. Soluzione costosa perché richiede l'apertura di una connessione per l'invio di pochi dati
- utilizzo di connessioni UDP. E' necessario definire un protocollo per la garantire l'affidabilità
- utilizzare un meccanismo di *remote procedure call*

REMOTE PROCEDURE CALL

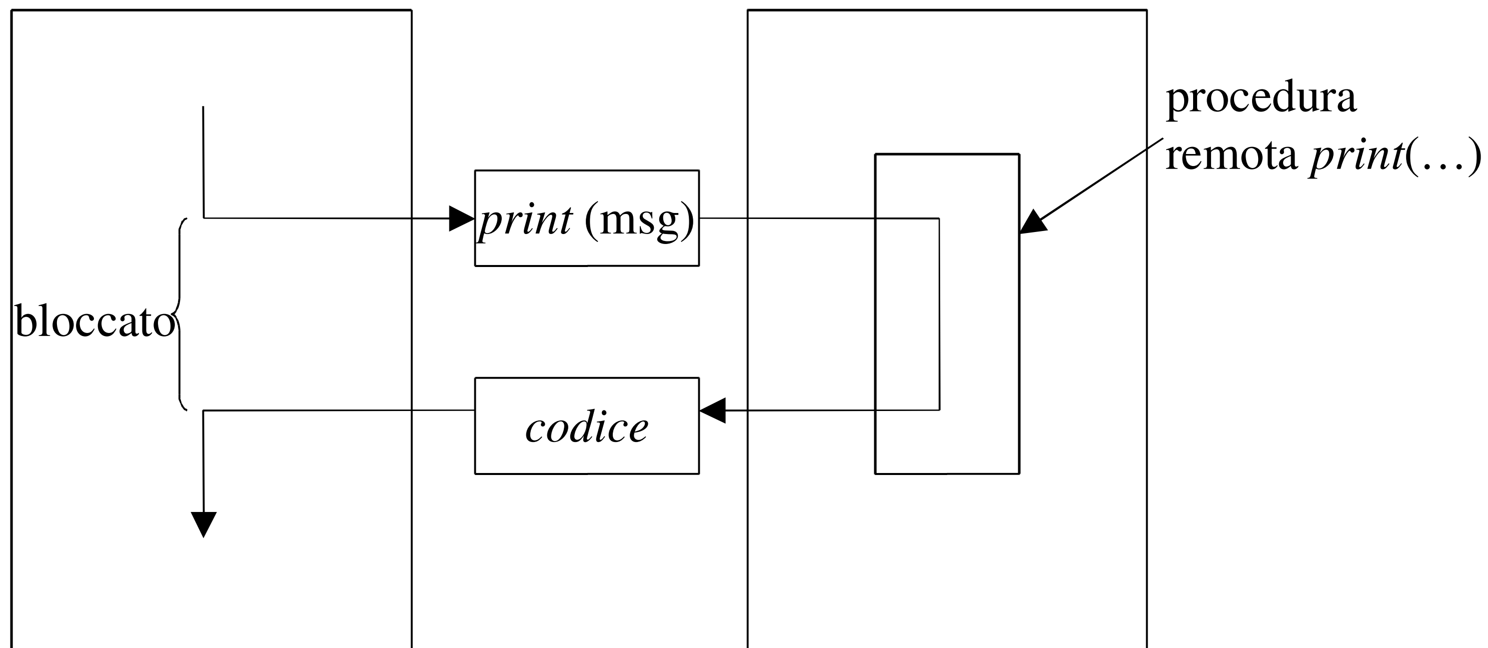
- Richiesta del client al server = *invocazione di una procedura* definita sul server

- Il client invoca una *procedura remota RPC (Remote Procedure Call)*

- I meccanismi utilizzati dal client sono gli stessi utilizzati per una normale invocazione di procedura, ma …

  - l’invocazione di procedura avviene sull’host su cui è in esecuzione il client
  - la procedura viene eseguita sull’host su cui è in esecuzione il server
  - i parametri della procedura vengono automaticamente sulla rete dal supporto all’RPC

REMOTE PROCEDURE CALL



Esempio: richiesta stampa di messaggio e restituzione esito operazione

REMOTE METHOD INVOCATION

implementazioni RPC

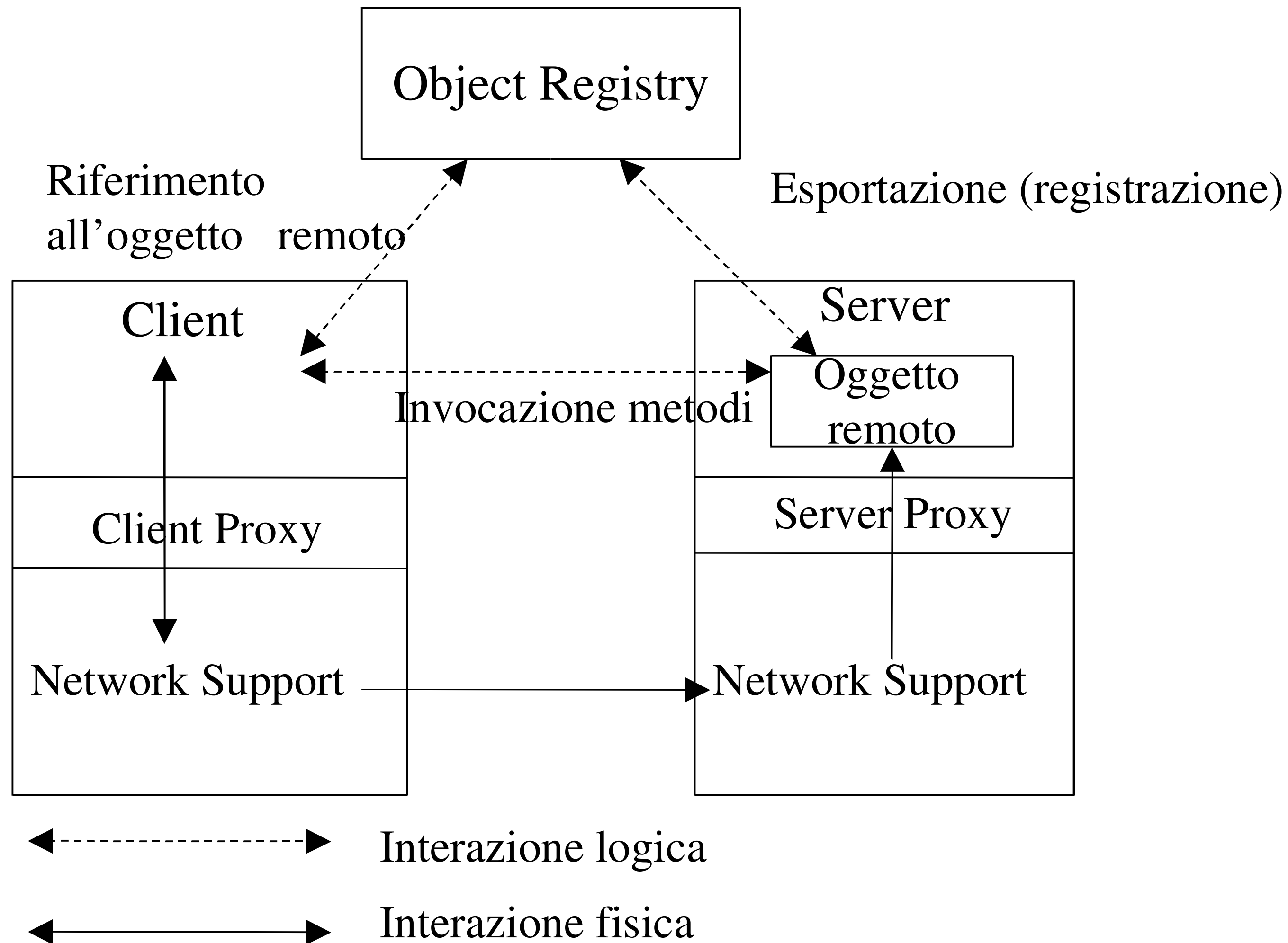
- Open Network Computing Remote Procedure Call (Sun)
- Open Group Distributed Computing Environment (DCE)
- …

Evoluzione di RPC = paradigma di interazione basato su *oggetti distribuiti*

remote method invocation (RMI): evoluzione del meccanismo di invocazione di procedura remota al caso di oggetti remoti

JAVA RMI: JAVA API per la programmazione ad oggetti distribuita

OGGETTI DISTRIBUITI: ARCHITETTURA GENERALE



OGGETTI DISTRIBUITI: ARCHITETTURA GENERALE

Il server che definisce *l'oggetto remoto*:

- definisce un *oggetto distribuito* = un oggetto i cui metodi possono essere invocati da parte di processi in esecuzione su hosts remoti)

- *esporta (pubblica)* l'oggetto ⇒

  crea un mapping

  nome simbolico oggetto/ riferimento all'oggetto

  e lo *pubblica* mediante un servizio *di tipo registry* (simile ad un DNS per oggetti distribuiti)

OGGETTI DISTRIBUITI: ARCHITETTURA GENERALE

Quando il client che vuole accedere all'oggetto remoto

- ricerca *un riferimento* all'oggetto remoto mediante i servizi offerti *dal registry*

- invoca i metodi definiti dall'oggeto remoto (*remore method invocation*).

- invocazione dei metodi di un oggetto remoto
  - *a livello logico:* identica all'invocazione di un metodo locale
  - a livello di supporto: è gestita da un *client proxy* che provvede a trasformare i parametri della chiamata remota in dati da spedire sulla rete.

    Il network support provvede quindi all'invio vero e proprio dei dati sulla rete

OGGETTI DISTRIBUITI: ARCHITETTURA GENERALE

Quando il server che gestisce l'oggetto remoto riceve un'invocazione per quell'oggetto

- il network support passa i dati ricevuti al *server proxy*
- il *server proxy* trasforma i dati ricevuti dal network support in una invocazione ad un metodo locale
- il *server proxy* trasforma i dati ricevuti dal network support nei parametri del metodo invocato

RMI: API JAVA

I metodi definiti dall'oggetto remoto ed i rispettivi parametri (le segnature dei metodi) devono essere noti:

- al client, che richiede un insieme di servizi mediante l'invocazione di tali metodi
- al server, che deve fornire *una implementazione* di tali metodi

Il client non è interessato *alla implementazione* di tali metodi

In JAVA:

- definizione di una *interfaccia* contiene le segnature di un insieme di metodi, ma non il loro codice

- Definizione di una classe che *implementa l'interfaccia:* contiene il codice dei metodi elencati nella interfacccia

JAVA INTERFACE

public interface solido {
public abstract Double area ( );
public abstract Double volume( );
public abstract String getName( );
}

non posso istanziare oggetti dell'interfaccia, posso definire una classe che *implementa* l'interfaccia e poi istanziare oggetti di quella classe

public class cilindro *implements* solido
{*public Double* area ( )
{*return* …..}

Le interfacce possono essere utilizzate per definire una forma di eredità multipla

JAVA: REMOTE INTERFACE

L'interfaccia di un oggetto remoto deve essere definita come *un'interfaccia Remota*

- estende l'interfaccia *Remote*
- i metodi definiti possono sollevare *eccezioni remote.* Una eccezione remota indica un generico fallimento nella comunicazione remota dei parametri e dei risultati al/da il metodo remoto

```
import java.rmi.*;
public interface EchoInterface extends Remote{
        String getEcho (String Echo) throws RemoteException;
                                                        }
```

Remote è una interfaccia che *non definisce alcun metodo*. Il solo scopo è quello di *identificare* gli oggetti che possono essere utilizzati in remoto

JAVA: REMOTE INTERFACE

Esempio: un Host è connesso ad una stazione metereologica che rileva temperatura, umidità,….mediante connessioni a diversi strumenti di rilevazione. sull'host è in esecuzione un server che fornisce queste informazioni agli utenti interessati.

```
import java.rmi.*;
public interface weather extends Remote{

    public double getTemperature( ) throws RemoteExceptions;
    public double getHumidity( ) throws RemoteExceptions;
    public double getWindSpeed ( ) throws RemoteExceptions;
    …….
}
```

JAVA: IMPLEMENTAZIONE DELL' INTERFACCIA

- definizione di una classe che *implementa* l'interfaccia remota
- la classe deve *estendere* la classe *UnicastRemoteObject*
- la classe può definire *ulteriori metodi pubblici*, ma solamente quelli definiti nella interfaccia remota possono essere invocati da un altro host

```
import java.rmi.*;
import java.rmi.server.*;
public class EchoRMIImplextends UnicastRemoteObject
                                        implements EchoInterface
        { public EchoRMIImpl( ) throws RemoteException
                { super ( );}
         public String getEcho (String echo) throws RemoteException
                { return echo}
        }
```

JAVA: IMPLEMENTAZIONE DELL' INTERFACCIA

public class UnicastRemoteObject *extends* RemoteServer

la classe UnicatRemoteObject definisce un insieme di metodi che consentono il corretto funzionamento di RMI, in particolare metodi per

- Il *marshalling* dei peramentri , cioè la trasformazione dei parametri in uno stream di bytes
- l'*unmarshalling* dei risultati cioè la trasformazione di uno stream di bytes nei valori restituiti

Se la classe estende un'altra classe, si può creare l'oggetto ed esportarlo passandolo al metodo *UnicastRemoteObject.exportObject( )*

JAVA: IMPLEMENTAZIONE DELL' INTERFACCIA

```
public class EchoRMIImpl extends UnicastRemoteObject
                                    implements EchoInterface
        { public EchoRMIImpl ( ) throws RemoteException
                { super ( );}
         public String getEcho (String echo) throws RemoteException
                { return echo}
        }
```

- il costruttore EchoRMIServer invoca il costruttore della super classe UnicastRemoteObject
- il costruttore della superclasse *esporta* l'oggetto = crea l'oggetto e si mette in attesa di richieste di invocazioni di metodi su una porta
- il costruttore può sollevare *RemoteException*, perché il costruttore della super classe *UnicastRemoteObject* può sollevare queste eccezioni

JAVA : GENERAZIONE DEI PROXY

- le applicazioni RMI richiedono la generazione di proxy, che consentano la trasformazione dei parametri in streams di bytes e viceversa.

- *Generazione dei proxy:* utilizzo dell'RMI compiler (*rmic*), ditribuito con il JDK.

  rmic EchoRMIImpl

- *rmic*
  - legge il file .class di una classe che implementa una interfaccia *remote*
  - produce *il proxy* in un file *EchoRMIImpl_Stub.class*

- Lo stub va passato al client che deve invocare i metodi remoti

JAVA : DEFINIZIONE DEL SERVER

L’oggetto remoto deve essere *istanziato* ed *esportato* da un server

```
import java.net.*;
import java.rmi.*;
public class EchoRMIServer{
   public static void main(String args [])
   { try
            { EchoRMIImpl server= new EchoRMIImpl();
            Naming.rebind (“echo”, server);
            System.out.println(“l’Echo Server è pronto!!”);
            }catch (RemoteException ex)
                        {System.out.println(“Eccezione nell’EchoServer”+ex)}
            catch (MalformedURLException ex)
                        {System.out.println(“URL  mal formata”+ex)}
}}
```

JAVA : DEFINIZIONE DEL SERVER

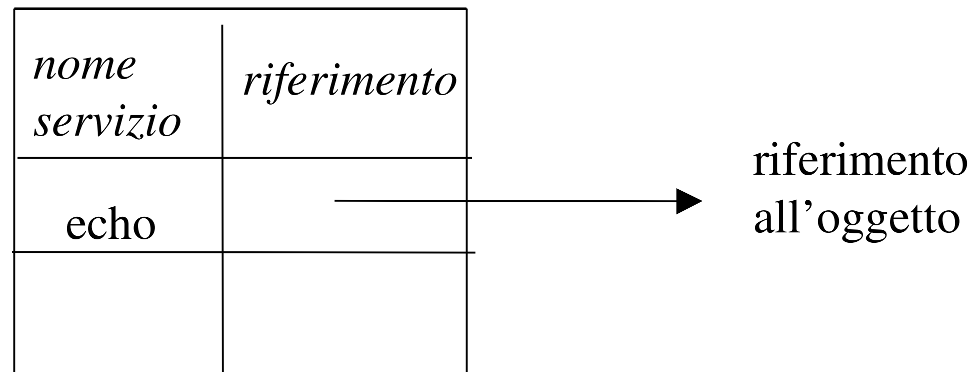
EchoServer

- crea *un'istanza* dell'oggetto remoto (*new*)

- inserisce un riferimento ad all'oggetto creato nel *registry locale (*che deve essere attivo su local host sulla porta di *default 1099*)

  Naming.rebind ("echo", server);

- *Registry* = simile ad un DNS per oggetti remoti, contiene legami tra il nome simbolico dell'oggetto remoto ed il riferimento all'oggetto



JAVA : DEFINIZIONE DEL SERVER

- la classe *Naming* contiene metodi per la gestione dei registry

- nel caso più semplice l'oggetto viene inserito in un registry locale, attivato sullo stesso host su cui è in esecuzione il server

- *Naming. rebind*(….) crea un collegamento tra un nome simbolico (qualsiasi) ed un riferimento all'oggetto. Se esiste già un collegamento per lo stesso oggetto all'interno dello stesso registry, tale collegamento *viene sovrascritto*

- è possibile inserire più istanze dello stesso oggetto remoto nel registry, con nomi simbolici diversi

- in generale un registry può essere individuato mediante *una URL completa*

rmi: //localhost:2048/echo

JAVA: ATTIVAZIONE DEL SERVER

Per rendere disponibile i metodi dell'oggetto remoto, è necessario attivare due tipi di servizi

- il server che fornisce il servizio di registrazione di oggetti remoti (*registry*)
- EchoServer: fornisce accesso ai metodi remoti

Attivazione del registry in background:

% rmiregistry &

- viene attivato un registry associato per default alla porta 1099

- Se la porta è già utilizzata, viene sollevata un'eccezione. Si può anche scegliere esplicitamente una porta

  % rmiregistry 2048 &

JAVA: INDIVIDUAZIONE DEL SERVIZIO DI NAMING

In generale il servizio di naming si individua con:

Naming.bind("rmi://fujih3.cli.di.unipi.it:2048/echo");

La URL specifica:

- nome del protocollo a cui si fa rifermiento: *rmi://*
- l'host e la porta su cui è in esecuzione il registry (può essere localhost)
- Il nome dell'oggetto remoto che voglio registrare

Per default:

host = Local host
porta =1099

JAVA: ATTIVAZIONE DEL SERVER

- Se la porta è già utilizzata, viene sollevata un'eccezione. Si può anche scegliere esplicitamente una porta

  % rmiregistry 2048 &

- ATTEZIONE: in questo caso nella rebind occorre specificare la URL completa

  rmi: //localhost:2048/echo

- In generale la URL specificata nella rebind deve individuare host e porta su cui è attivo il servizio di naming

JAVA: L'RMI CLIENT

Il client:

- ricerca un riferimento all'oggetto remoto
- invoca i metodi dell'oggetto remoto come fossero metodi locali (unica differenza: occorre intercettare RemoteException)

Per ricercare il riferimento all'oggetto remoto, il client

- deve accedere al registry attivato sul server.
- il registry viene individuato mediante la sua URL (simile ad URL HTTP)
- Il riferimento restituito dal registry è un riferimento ad un oggetto di tipo *Object* $\Rightarrow$ è necessario effettuare il *casting dell'oggetto* restituito al tipo definito nell'interfaccia remota

JAVA: L'RMI CLIENT

```
import java.rmi.*;
import java.net.*;
public class EchoRMIClient
    {public static void main (String Args[])
    Scanner in= new Scanner (System.in);
    try
            {EchoInterface serverRMI =
            (EchoInterface) Naming.lookup("rmi://fujih3.cli.di.unipi.it:2048/echo");
            String message;
            message=in.next( );
            String echo = serverRMI.getEcho(message);
            System.out.println(echo)}
      catch (Exception e)
    }
}
```

JAVA: L’RMI CLIENT

L’esecuzione del client richiede

- la classe EchoRMIClient.class, risultante della compilazione del client

- La classe EchoInterface.class

- Il proxy *EchoRMIImpl_Stub.class,* risultante dalla compilazione della classe che implementa l’oggetto remoto , mediante rmic

METODI DELLA CLASSE NAMING

- *bind* ( )
- *rebind(*)
- *unbind ()*
- *lookup* ( )
- *list* ()

ESERCIZIO

Sviluppare una applicazione RMI per la gestione di una elezione. Il server esporta un iniseme di metodi

- *public void* vota (*String* nome). Accetta come parametro il nome del candidato. Non restituisce alcun valore. Registra il voto di un candidato in una struttura dati opportunamente scelta

- *public int* risultato (*String* nome). Accetta come parametro il nome di un candidato e restituisce i voti accumulati da tale candidato fino a quel momento.

- un metodo che consenta di ottenere i nomi di tutti i candidati, con i rispettivi voti, ordinati rispetto ai voti ottenuti