




# Tehnologii avansate de programare

## *Curs 3*

Cristian Frăsinaru

`acf@infoiasi.ro`

Facultatea de Informatică  
**Universitatea "Al. I. Cuza" Iași**





# Remote Method Invocation



# Cuprins

---

- Ce este RMI ?
- Tehnologii similare
- Arhitectura
- Identificarea serviciilor
- Componentele unui sistem RMI
- Compilarea și execuția
- Descărcarea dinamică a claselor

# Introducere

# Ce este RMI ?

- Programare de rețea la un nivel superior
- Tehnologie Java pentru implementarea aplicațiilor distribuite
- Oferă o sintaxă și semantică similare cu cele ale aplicațiilor ne-distribuite

# Caracteristici

- Permite colaborarea obiectelor aflate în mașini virtuale diferite.
- Permite unei aplicații să apeleze metode ale unui obiect aflat în alt spațiu de adrese.
- Implementează soluții (la nivel distribuit) pentru:
  - identificarea obiectelor externe (*remote*)
  - trimiterea parametrilor și primirea rezultatelor
  - tratarea excepțiilor
  - gestiunea memoriei
- Portabilitate

# Tehnologii similare

- **CORBA** (Common Object Request Broker Architecture - OMG)
- **Java IDL** (Interface Definition Language)
- **RMI-IIOP** (Internet Inter-ORB Protocol)
- **DCOM** - (Distributed Component Object Model - Microsoft)
- **DCE** - (Distributed Computing Environment - Open Group )
- **Web Services**

# Comparație cu alte tehnologii

	<b>RMI</b>	<b>Corba</b>	<b>DCE</b>	<b>WS</b>
<b>Mecanism invocare</b>	Java RMI	Corba RMI	RPC	JAX-RPC, .NET,...
<b>Format date</b>	Serializare	CDR	NDR	XML
<b>Format mesaje</b>	Flux	GIOP	PDU	SOAP
<b>Protocol transfer</b>	JRMP	IIOp	RPC CO	HTTP
<b>Interfețe</b>	Java	CORBA IDL	DCE IDL	WSDL
<b>Descoperire</b>	Java Registry	COS Naming	CDS	UDDI

JRMP = Java Remote Method Protocol. ORB = Object Request Broker. CDR = Common Data Representation. GIOP = General Inter-ORB Protocol. IIOp= Internet Inter-ORB Protocol. IDL = Interface Definition Language. COS = CORBA Object Services. RPC = Remote Procedure Call. NDR = Network Data Representation. PDU = Protocol Data Units. RPC CO = RPC Connect-Oriented protocol. IDL = Interface Definition Language. CDS = Cell Directory Service.



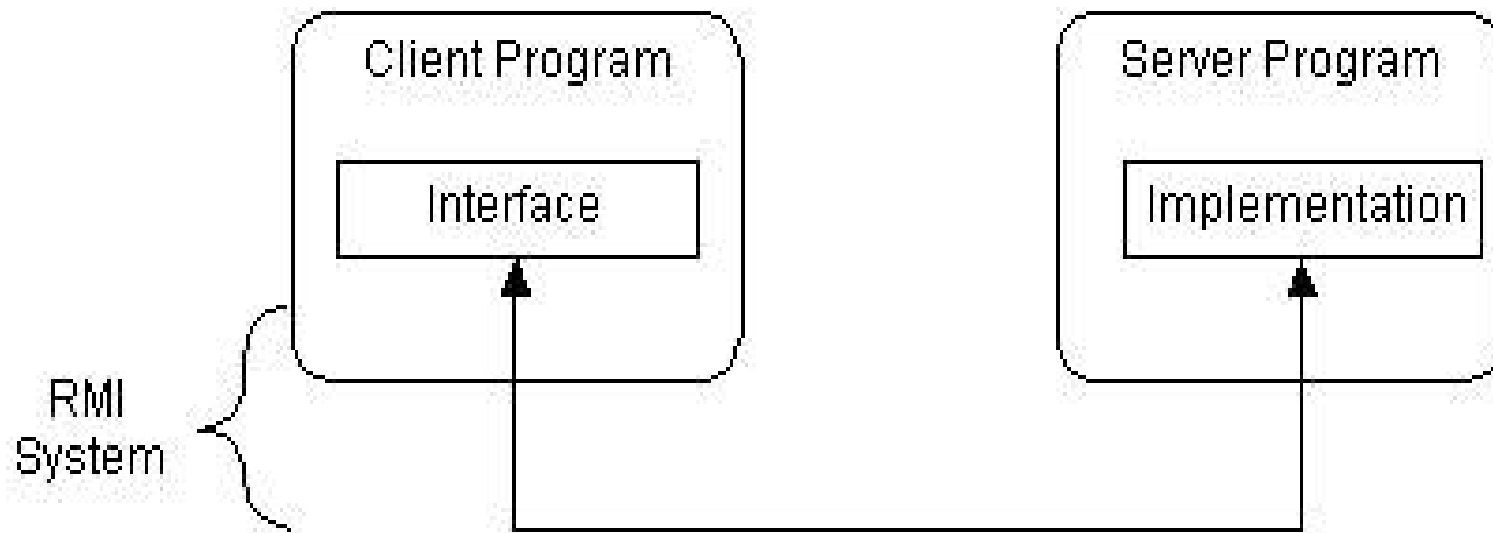
# Aplicații distribuite - nedistribuite

	<b>Local</b>	<b>La distanță</b>
Definiție	<code>interface</code>	?
Implementare	<code>class</code>	?
Creare	<code>new</code>	?
Acces	referință	?
Referință	obiect heap	?
Activ	$\geq 1$ ref.	?
Finalizare	<code>finalize</code>	?
Distrugere	<code>gc</code>	?
Excepții	<code>Exception</code>	?

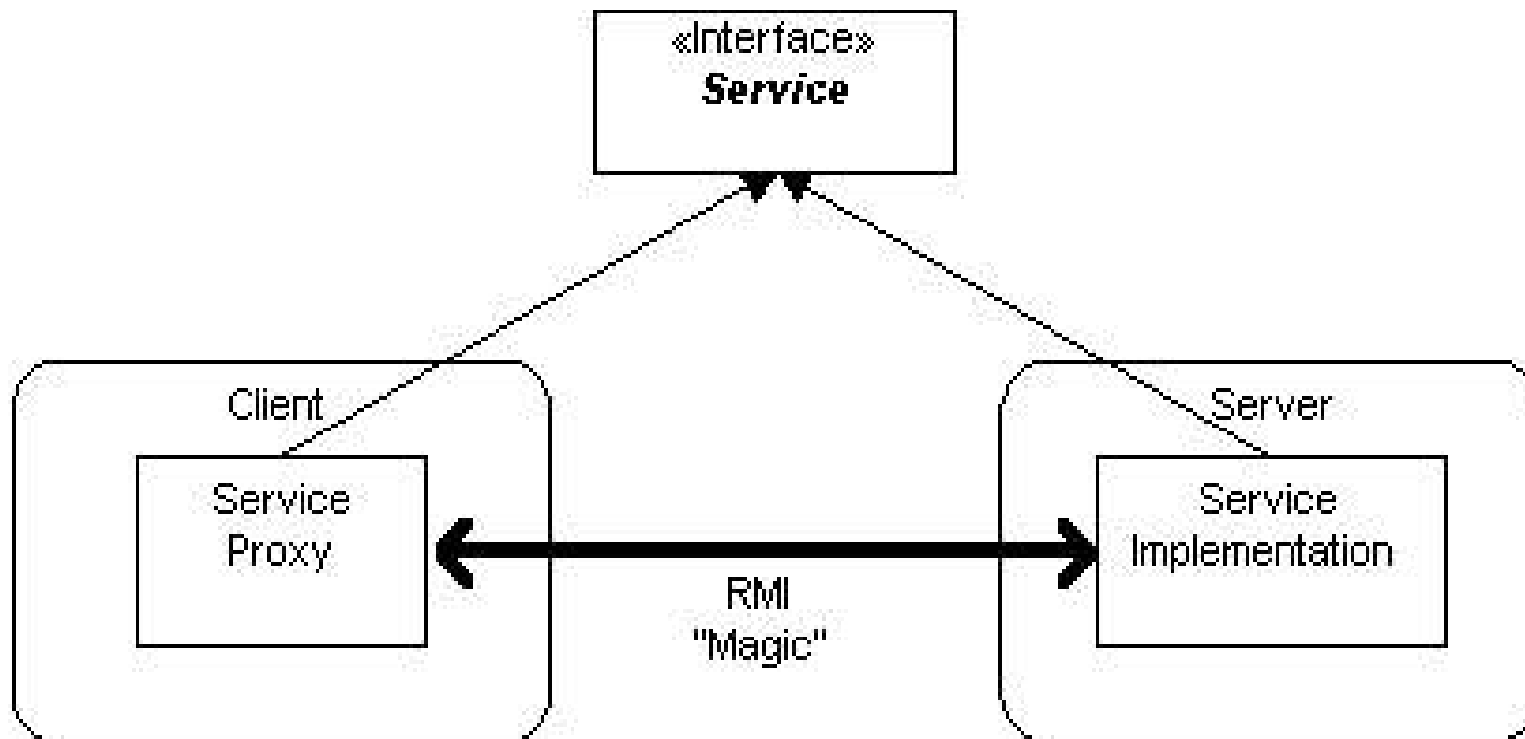
# Arhitectura RMI

# Principiul de bază

## Separarea conceptelor de comportament și implementare



# Clase și interfețe

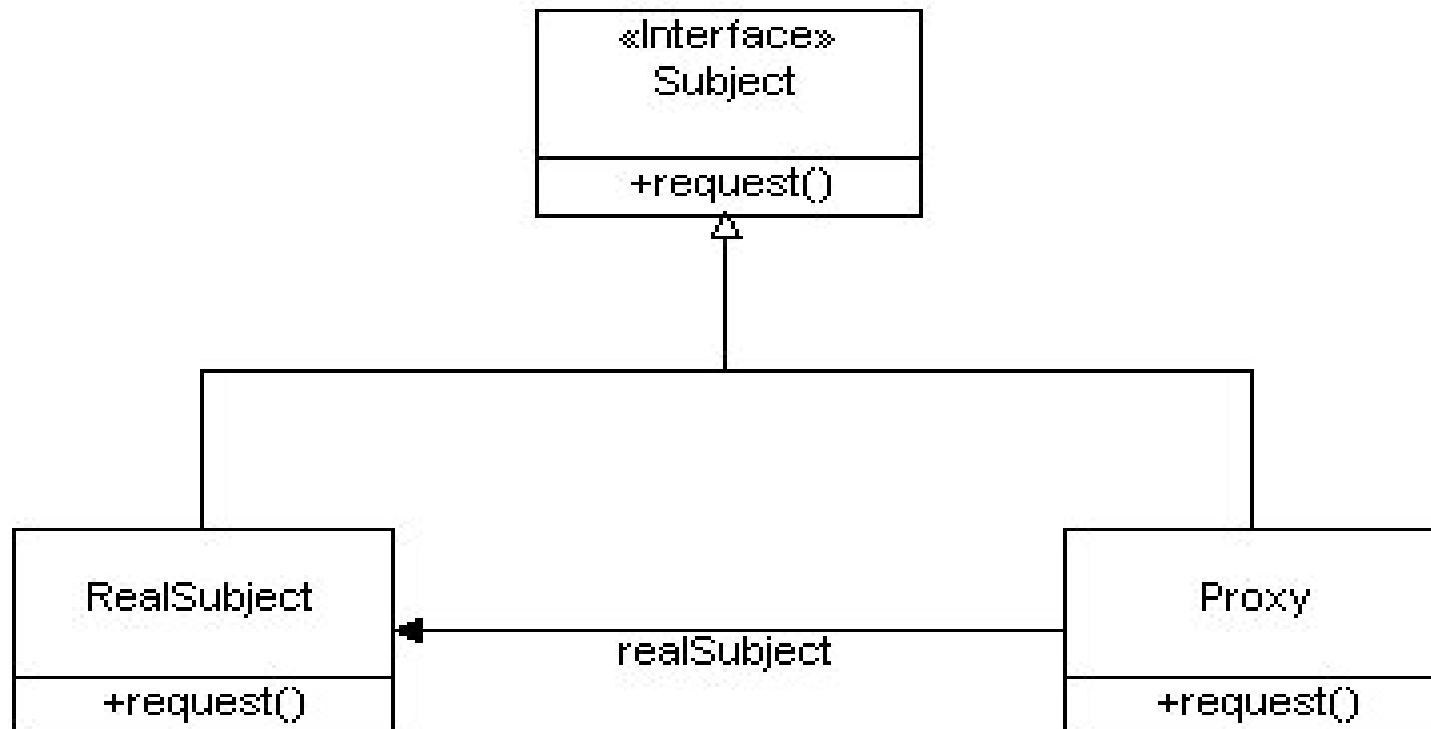


# Proxy Design Pattern (1)

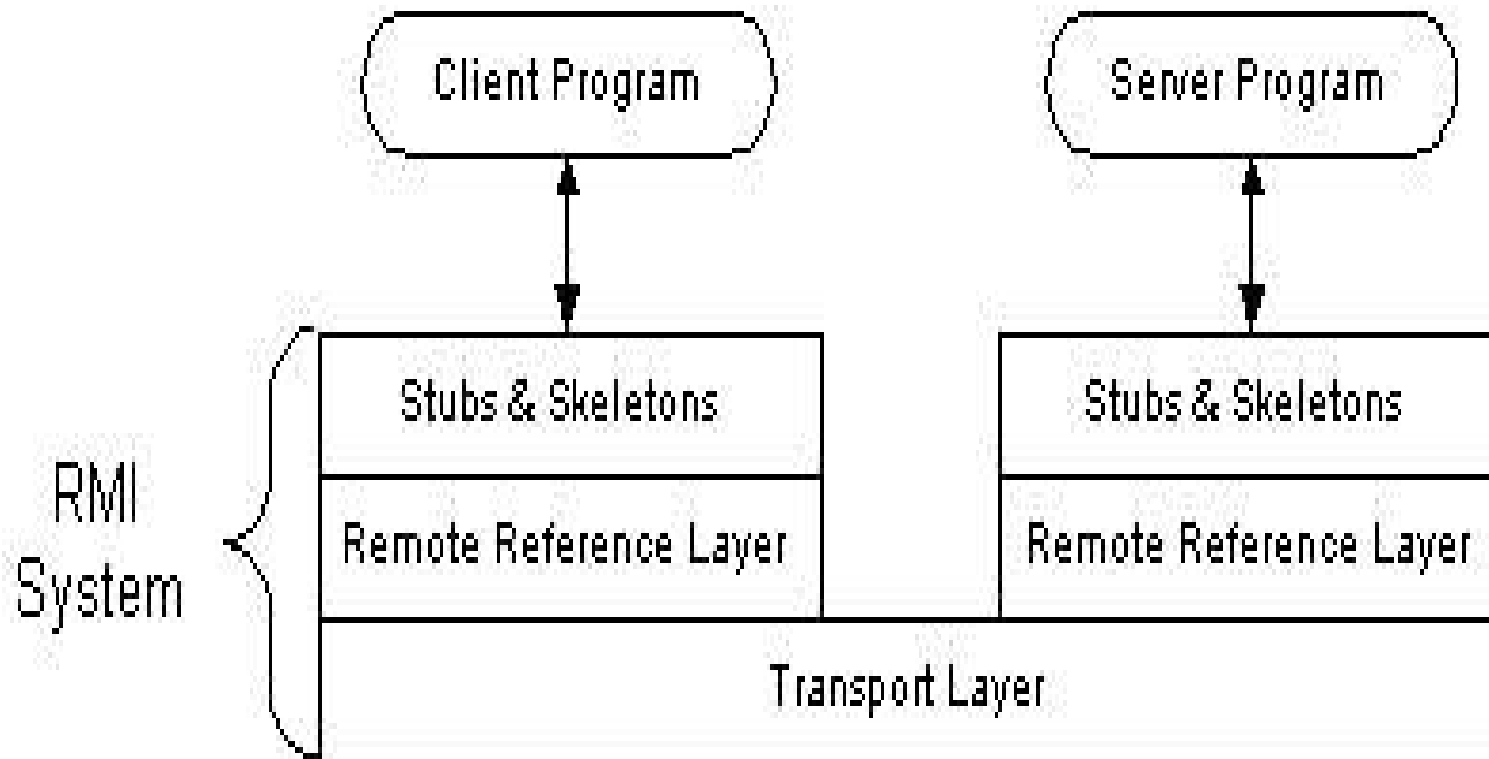
**Proxy**=obiect care funcționează ca o interfață către alt obiect (din rețea, memorie, etc.) Tipuri de proxy:

- Remote Proxy
- Virtual Proxy
- Protection (Access) Proxy
- Cache Proxy
- Synchronization Proxy
- Smart Reference Proxy

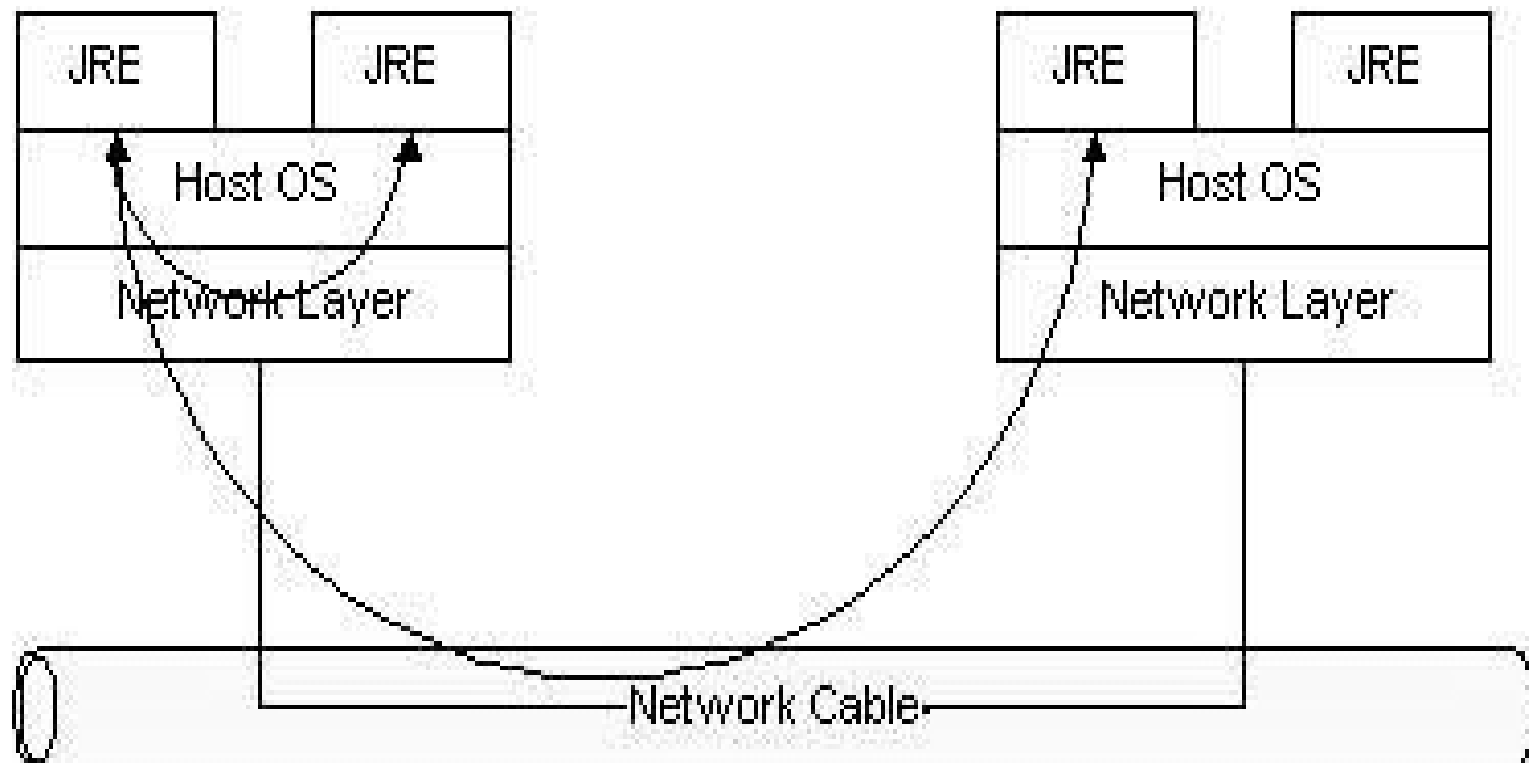
# Proxy Design Pattern (2)



# Nivele de funcționalitate



# Transportul datelor





# Identificarea serviciilor

- Cum poate fi identificat un obiect în rețea ?
- Prin servicii de nume.

- **JNDI** (Java Naming and Directory Interface)
- **RMI Registry**

**rmi://<numeServer> [:<port>] /<numeServiciu>**

# RMI Registry

- Utilitarul **rmiregistry** este inclus în distribuția standard.  
Trebuie pornit pe orice mașină pe care se găsesc obiecte ce oferă servicii.  
Așteaptă cereri de la clienți la un anumit port (implicit 1099).
- **Crearea unui serviciu** - se realizează pe server și presupune exportul unui obiect în regiștri sub un nume public.
- **Căutarea unui serviciu** - este realizată de client cu metoda **Naming.lookup**

# Folosirea RMI

# Componentele unui sistem RMI

- Interfețele serviciilor
- Clasele cu implementări concrete
- Fișierele *stub* și *skeleton*
- Mașina server care găzduiește serviciul
- Un serviciu de nume
- O aplicație client care solicită serviciul

# RMI API

- **java.rmi**

Remote, Naming

- **java.rmi.server**

RMIServer, UnicastRemoteObject

- **java.rmi.registry**

Registry, LocateRegistry

- **java.rmi.activation**

- **java.rmi.dgc**

# Interfața



Oferă **descrierea** unui serviciu.

## Hello.java

```
import java.rmi.Remote;  
import java.rmi.RemoteException;  
  
public interface Hello extends Remote {  
    String echo(String name) throws RemoteException;  
}
```



# Implementarea

Oferă implementarea unui serviciu.

## HelloImpl.java

```
import java.rmi.*;
import java.rmi.server.*;

public class HelloImpl extends UnicastRemoteObject
    implements Hello {
    public HelloImpl() throws RemoteException {
        super();
    }
    public String echo(String name) {
        return "Hello " + name;
    }
}
```

# Aplicația server



## Server.java

```
import java.rmi.*;
import java.rmi.server.*;

public class Server {
    public Server() throws RemoteException {
        super();
    }
    public static void main(String[] args) {
        String name = "rmi://localhost/Hello";
        try {
            HelloImpl obj = new HelloImpl();
            Naming.rebind(name, obj);
            System.out.println("Serviciul Hello activat!");
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```





# Aplicația client

## Client.java

```
import java.rmi.*;

public class Client {
    public static void main(String args[]) {
        try {
            String name = "rmi://localhost/Hello";
            Hello obj = (Hello) Naming.lookup(name);
            System.out.println(obj.echo("Duke"));

        } catch (Exception e) {
            System.err.println(e);
        }
    }
}
```

# Compilarea și execuția

Să presupunem că toate fișierele sunt în același director.

1. Compilăm sursele: **javac \*.java**
2. Generăm fișierele stub/skeleton: **rmic HelloImpl**
3. Pornim serviciul de nume: **rmiregistry**
4. Creăm serviciul: **java Server**
5. Apelăm serviciul: **java Client**

# Distribuirea componentelor



## Server

- Interfețele și implementările serviciilor
- Clasele *stub* și *skeleton*

## Client

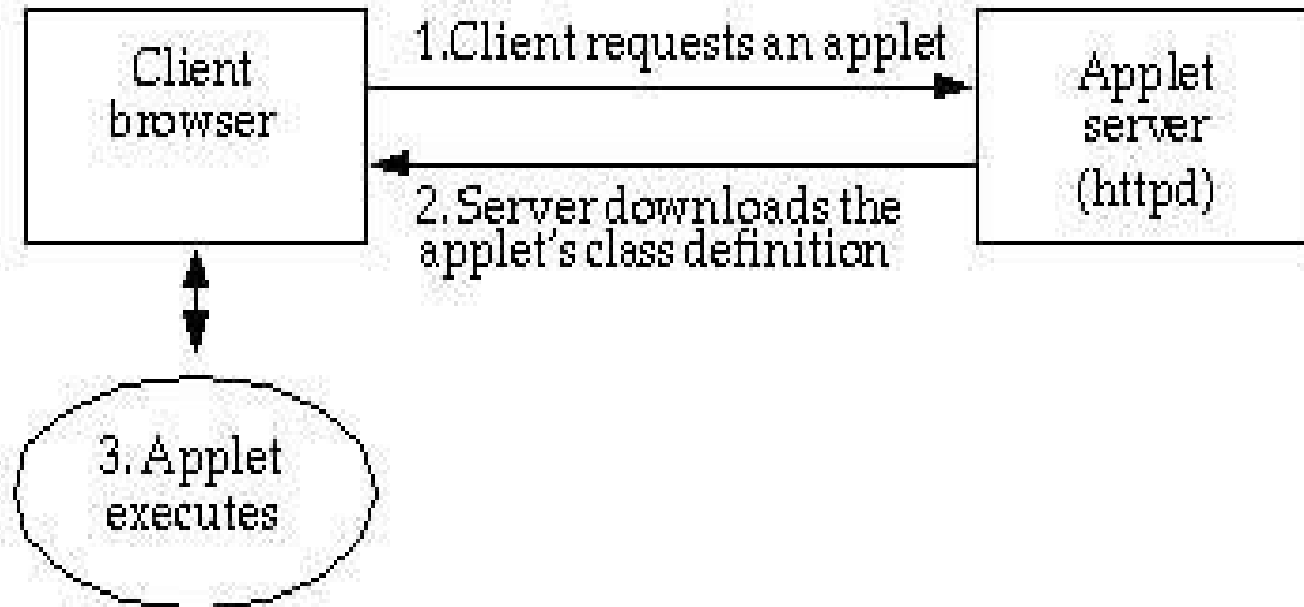
- Interfețele serviciilor necesare
- Clasele *stub*
- Clasele necesare pentru trimiterea parametrilor sau primirea rezultatelor



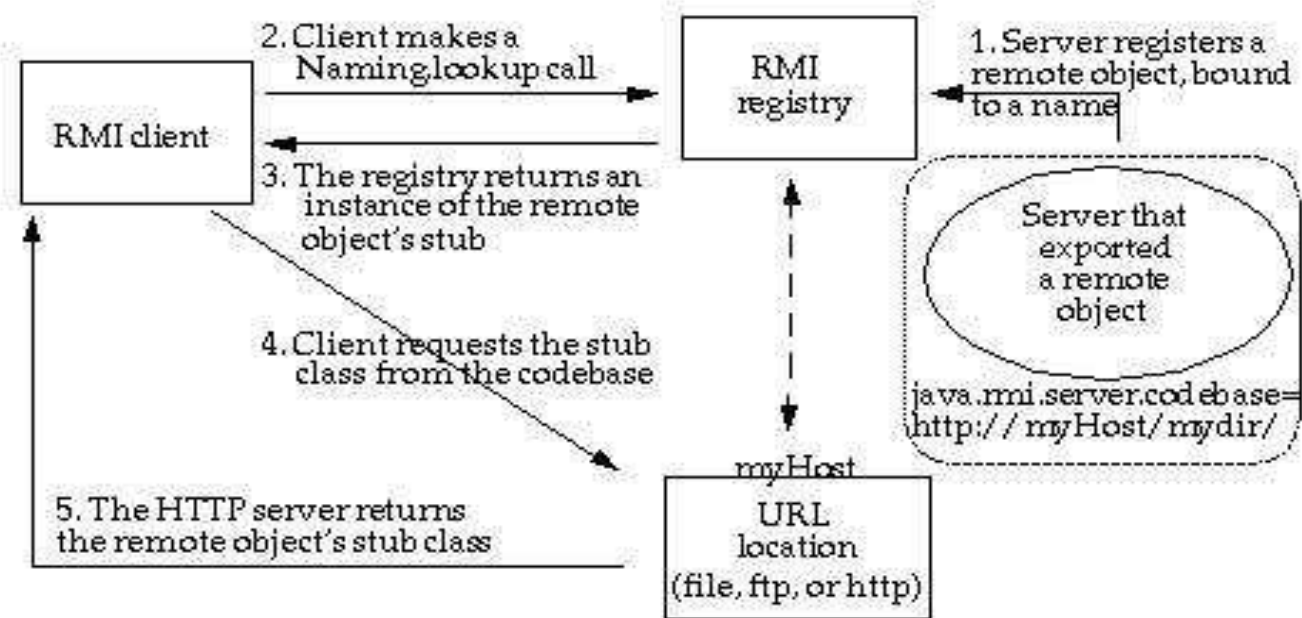


# Descărcarea dinamică a claselor

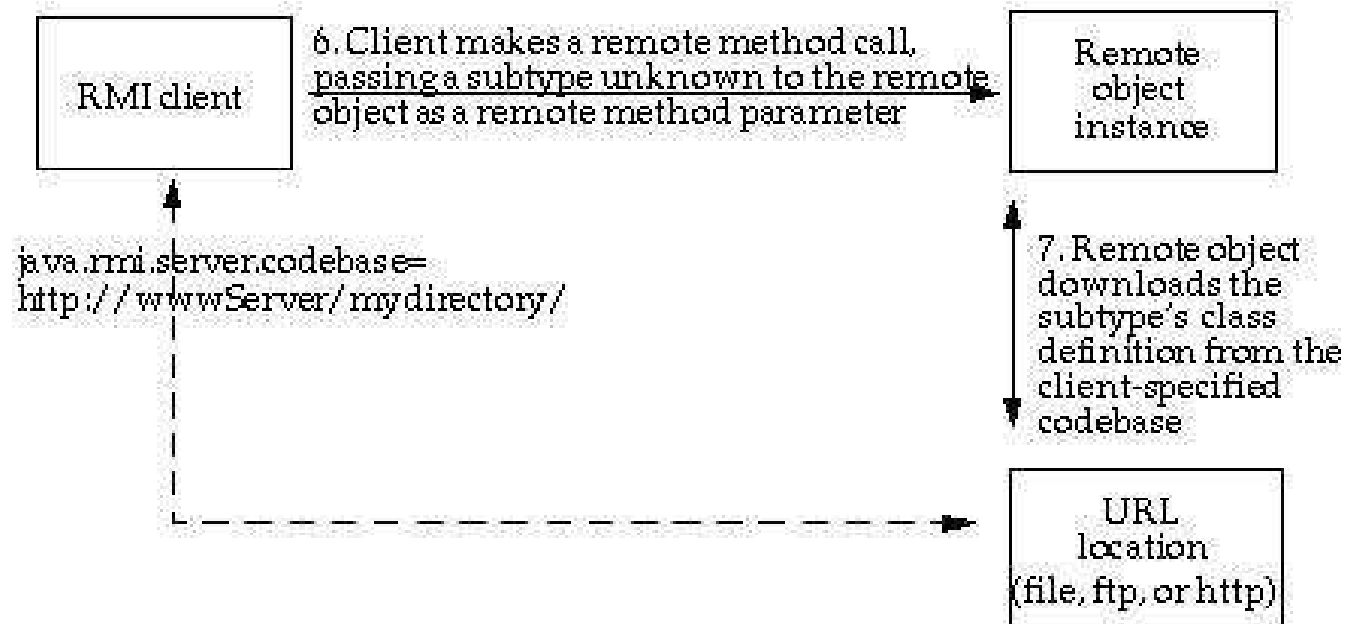
# Parametrul *codebase*



# Descărcarea dinamică *stub*-ului



# Descărcarea altor clase



# Execuția sistemului



## server

```
set classpath=  
java -Djava.rmi.server.codebase=file:/d:/java/work/rmi/server/  
      -Djava.security.policy=server.policy Server
```

## client

```
set classpath=  
java -Djava.rmi.server.codebase=file:/d:/java/work/rmi/client/  
      -Djava.security.policy=client.policy Client  
rem pause
```

```
java -Djava.rmi.server.codebase=http://81.196.96.113:8080/rmi/
```





# Transmiterea parametrilor

- **Tipuri primitive**

- **Tipuri referință**

Obiectele sunt transmise folosind mecanismul **serializării**.

- **Tipuri "remote"**

Tipul returnat de serviciu va fi transmis clientului prin intermediul unui obiect proxy.

# Parametrii 'remote'



```
//Client.java
Test t = remoteObj.test();
System.out.println(t.getInfo());
...
//Test.java
public interface Test extends java.io.Serializable

//TestImpl.java
public class TestImpl implements Test

//RemoteObjImpl
...
public Test test() {
    return new TestImpl("Salutare !");
}
}
```



# Exportul obiectelor



## UnicastRemoteObject.exportObject

```
import java.rmi.*;
import java.rmi.server.*;

public class RMIServer extends Object implements RMIInterface {

    public static void main(String[] args) throws Exception {
        RMIServer s = new RMIServer();
        UnicastRemoteObject.exportObject(s);
        Naming.rebind("MyServer", s);
    }
}
```



# Identificarea sursei unui apel

## getClientHost

```
public void myRemoteMethod() {  
    try {  
        String client = getClientHost();  
        System.out.println("Called by " + client);  
    } catch (ServerNotActiveException e) {  
        System.out.println("Server not active \n" + e);  
    }  
}
```

# Listarea obiectelor din regiștri

## Naming.list

```
import java.rmi.*;

public class RegList {
    public static void main(String[] args) throws Exception {
        String host;
        if (args.length == 0) host = "localhost:1099";
        else host = args[0];
        String[] names = Naming.list("//" + host + "/");
        for (int i = 0; i < names.length; i++)
            System.out.println(names[i]);
    }
}

...
rmi://localhost:1099/App1Server
rmi://localhost:1099/App2Server
```

# Distributed Garbage Collection

- Mecanism de **numărare** a clienților care accesează un serviciu.
- Obiectele sunt marcate: *dirty/clean*
- Mecanism de notificare când nu mai există clienți active: `Unreferenced`
- Timeout: `java.rmi.dgc.leaseValue` (10min)
- Un client trebuie să poată trata situații în care obiectul referit a "dispărut".