



Tehnologii Java

Curs -

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

	RMI	Corba	DCE	WS
Mecanism invocare	Java RMI	Corba RMI	RPC	JAX-RPC, .NET,...
Format date	Serializare	CDR	NDR	XML
Format mesaje	Flux	GIOP	PDU	SOAP
Protocol transfer	JRMP	IIOp	RPC CO	HTTP
Interfețe	Java	CORBA IDL	DCE IDL	WSDL
Descoperire	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.

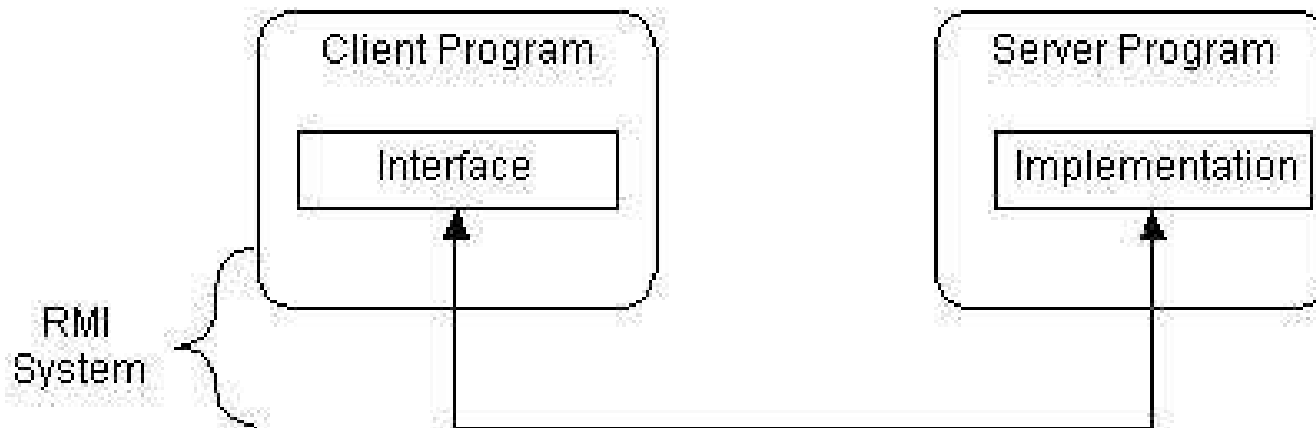
Aplicații distribuite - nedistribuite

	Local	La distanță
Definiție	<code>interface</code>	?
Implementare	<code>class</code>	?
Creare	<code>new</code>	?
Acces	referință	?
Referință	obiect heap	?
Activ	≥ 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

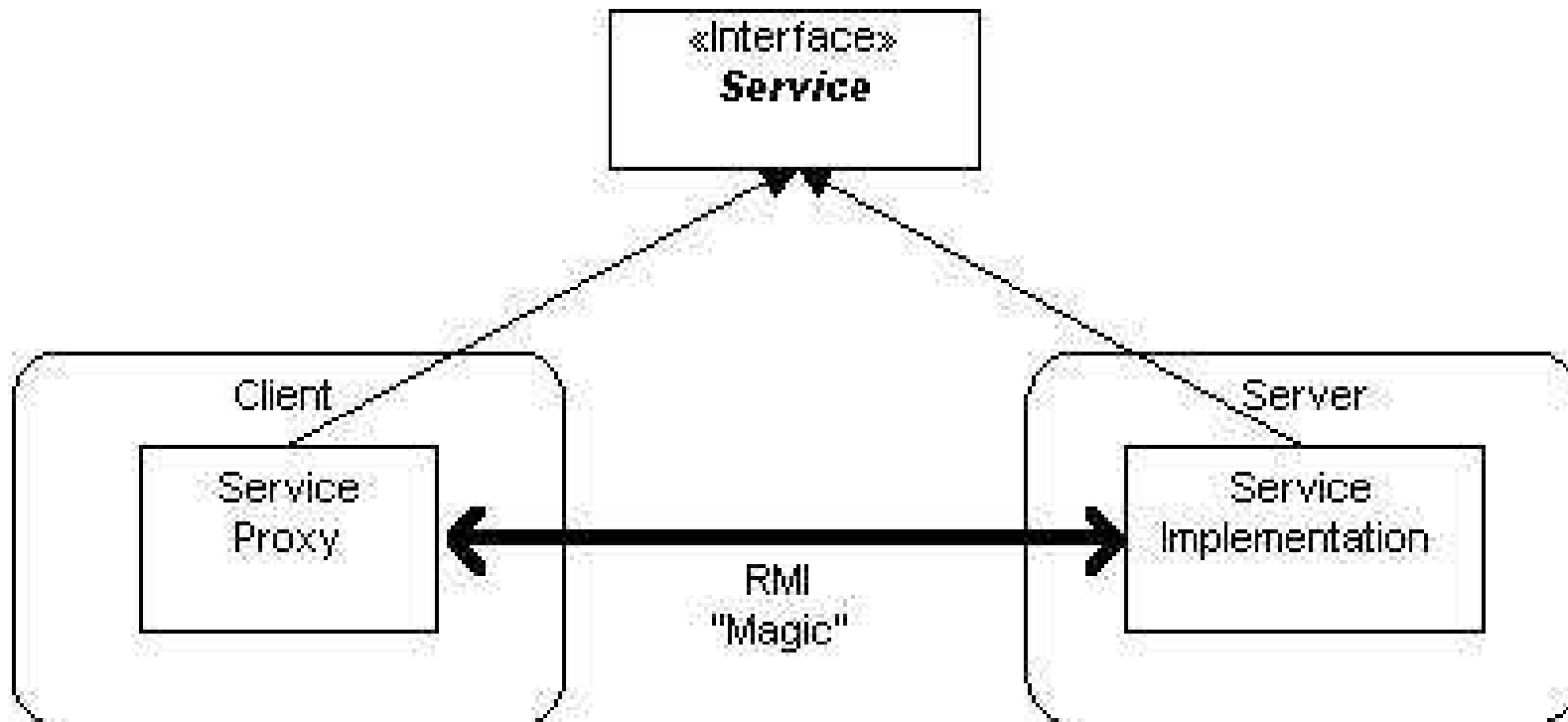


Proxy Design Pattern (1)

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)



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

• Server

- Interfețele serviciilor
- Clasele cu implementări concrete
- Un serviciu de nume
- Eventual, un server Web

• Client

- Interfețele serviciilor
- O aplicație client care solicită serviciul
- Eventual, un server Web



Exemplul 1

Hello World!

Interfață - Descriere

Hello.java

```
package service;
```

```
import java.rmi.Remote;
```

```
import java.rmi.RemoteException;
```

```
public interface Hello extends Remote {
```

```
    String sayHello(String name) throws RemoteException;
```

```
}
```

API → **Remote**

Clasă - Implementare



HelloImpl.java

```
package server;
import java.rmi.RemoteException;
import service.Hello;

public class HelloImpl implements Hello {

    public HelloImpl() throws RemoteException {
        super();
    }

    public String sayHello(String name) {
        return "Hello " + name;
    }
}
```



Aplicația server

Server.java

```
package server;
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;
import service.Hello;

public class HelloServer {
    public static void main(String[] args) throws Exception {
        // *
        Hello hello = new HelloImpl();
        Hello stub = (Hello) UnicastRemoteObject.exportObject(hello, 0);
        Registry registry = LocateRegistry.getRegistry();
        registry.rebind("Hello", stub);
        System.out.println("Serviciul Hello activat!");
    }
}
```

API → **UnicastRemoteObject, Registry**

Aplicația client

Client.java

```
package client;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import service.Hello;

public class HelloClient{
    public static void main(String[] args) throws Exception {
        // *
        Registry registry = LocateRegistry.getRegistry("localhost");
        Hello hello = (Hello) registry.lookup("Hello");

        System.out.println(hello.sayHello("Duke"));
    }
}
```



Exemplul 2

Compute Engine



Descrierea serviciului



Compute.java

```
package service;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote {

    <T> T executeTask(Task<T> t) throws RemoteException;
}
```



Descrierea unui *task*



Task.java

```
package service;

public interface Task<T> {

    T execute();
}
```



Implementare serviciului

ComputeImpl.java

```
package server;

import java.rmi.RemoteException;
import service.Compute;
import service.Task;

public class ComputeImpl implements Compute {
    public ComputeImpl() throws RemoteException {
        super();
    }
    public <T> T executeTask(Task<T> t) {
        return t.execute();
    }
}
```

Aplicația server

ComputeServer.java

```
package server;
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;
import service.Compute;

public class ComputeServer {
    public static void main(String[] args) throws Exception {
        // *
        Compute compute = new ComputeImpl();
        Compute stub =
            (Compute) UnicastRemoteObject.exportObject(compute, 0);
        Registry registry = LocateRegistry.getRegistry();
        registry.rebind("Compute", stub);
        System.out.println("Serviciul Compute activat!");
    }
}
```

Aplicația client

ComputeClient.java

```
package client;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import service.Compute;

public class ComputeClient{
    public static void main(String[] args) throws Exception {
        // *
        Registry registry = LocateRegistry.getRegistry("localhost");
        Compute compute = (Compute) registry.lookup("Compute");

        Task task = new Pi(1000);
        System.out.println(compute.execute(task));
    }
}
```

Implementarea unui *task*

Pi.java

```
package client;

import java.io.Serializable;
import java.math.BigDecimal;
import service.Task;

public class Pi implements Task<BigDecimal>, Serializable {
    private final int digits;
    public Pi(int digits) {
        this.digits = digits;
    }
    public BigDecimal execute() {
        return new BigDecimal(3.14);
    }
}
```

Pi.java → **DestroyAll.java**

Fișierele de permisiuni



—→ **Inlocuim * cu:**

```
if (System.getSecurityManager() == null) {  
    System.setSecurityManager(new SecurityManager());  
}
```

server.policy

```
grant codebase "file:/d:/java/RMIServer/build/classes" {  
    permission java.security.AllPermission;  
};
```

client.policy

```
grant codebase "file:/d:/java/RMIClient/build/classes" {  
    permission java.security.AllPermission;  
};
```



Descărcarea dinamică a claselor



Intrebări:

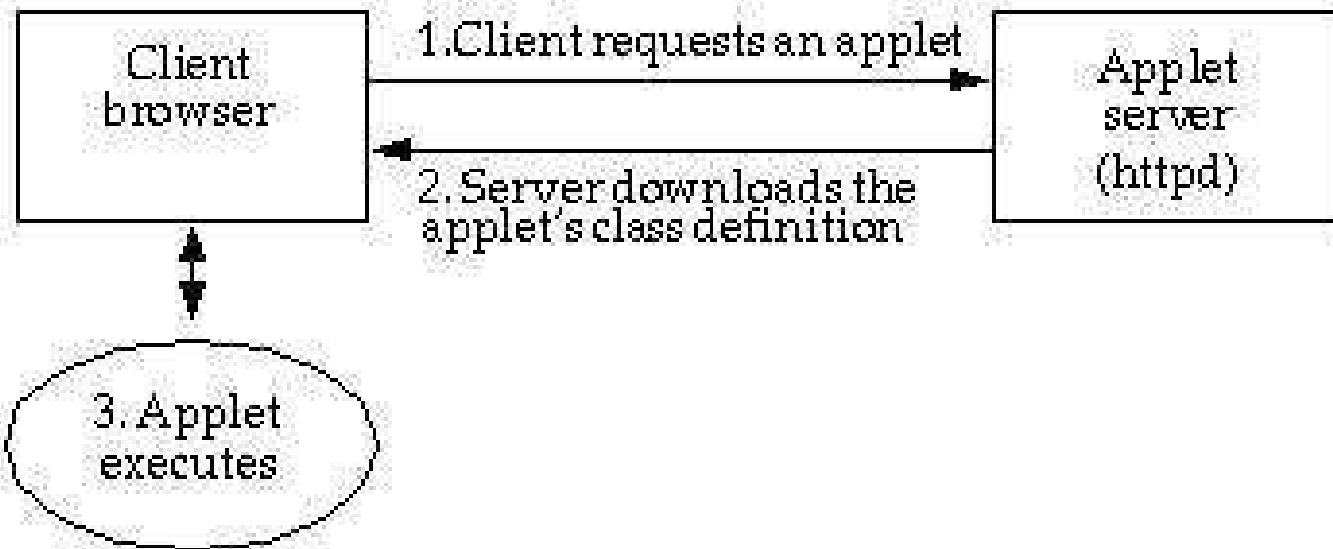
- Unde găsește clientul interfețele **Compute** și **Task** ?
- Unde găsește serverul definiția clasei **Pi** ?

Răspuns=**Codebase**

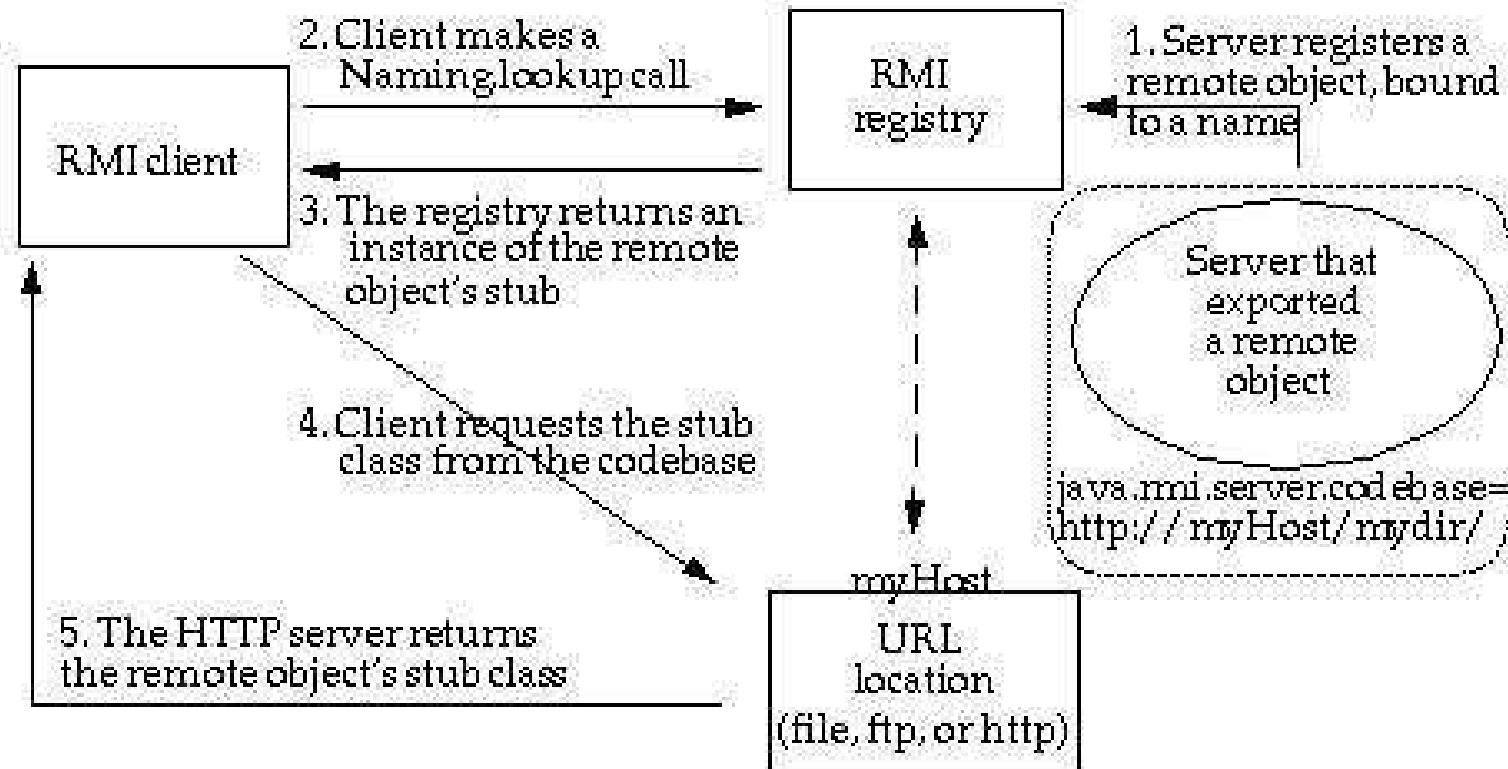
A codebase can be defined as a source, or a place, from which to load classes into a virtual machine You can think of your CLASSPATH as a "local codebase"



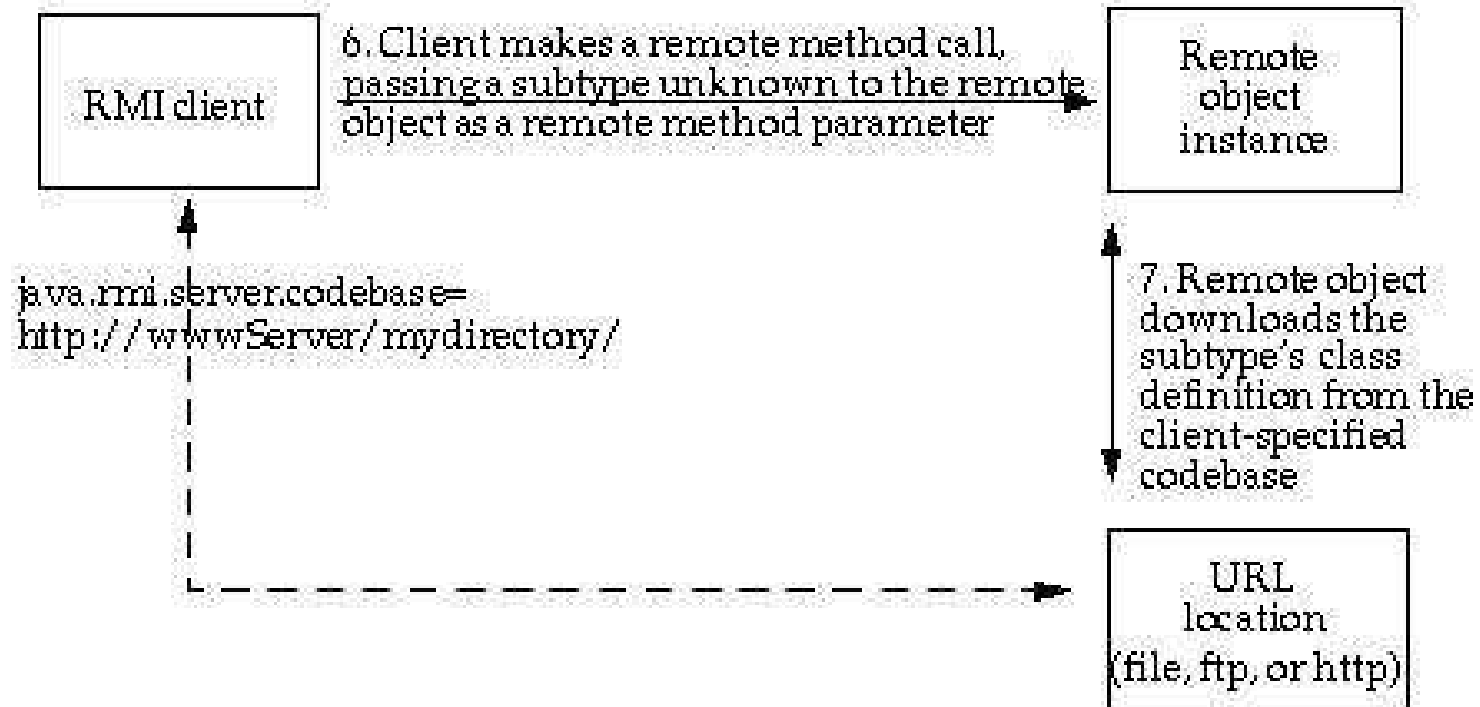
Codebase



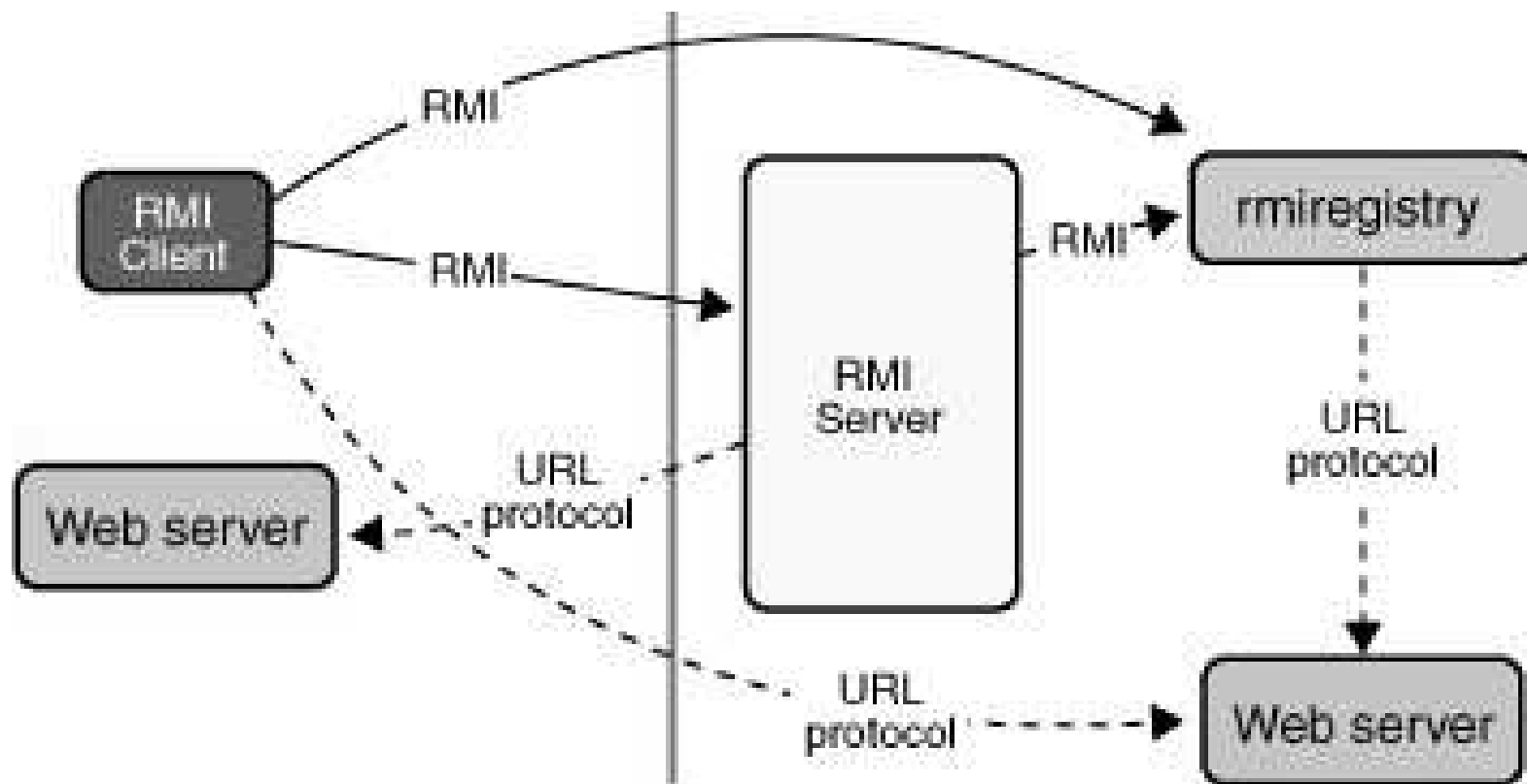
Descărcarea dinamică *stub*-ului



Descărcarea altor clase



Imagine de ansamblu



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.

Compilarea și execuția



La nivel de server

```
-Djava.security.policy=server.policy  
-Djava.rmi.server.codebase=file:/d:/java/RMIServer/dist/RMIServer.jar  
-Djava.rmi.server.hostname=localhost
```

La nivel de client

```
-Djava.security.policy=client.policy  
-Djava.rmi.server.codebase=file:/d:/java/RMIClient/dist/RMIClient.jar
```



Alte servicii RMI

Identificarea sursei unui apel



getClientHost

```
public class MyRemoteClass extends UnicastRemoteObject {
    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 RegistryList {
    public static void main(String[] args) throws Exception {
        String host = "localhost:1099";
        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".