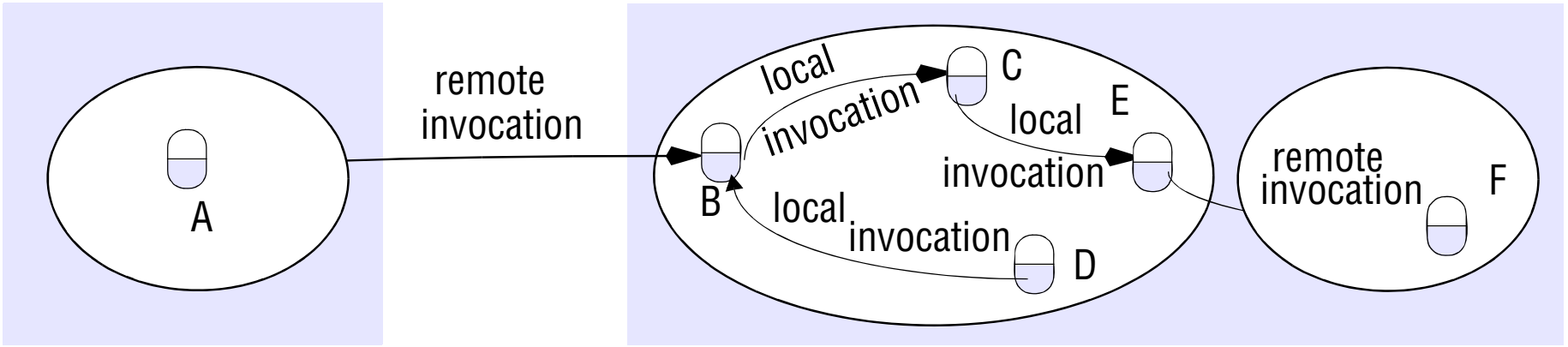
Remote Method Invocation - RMI

Java RMI

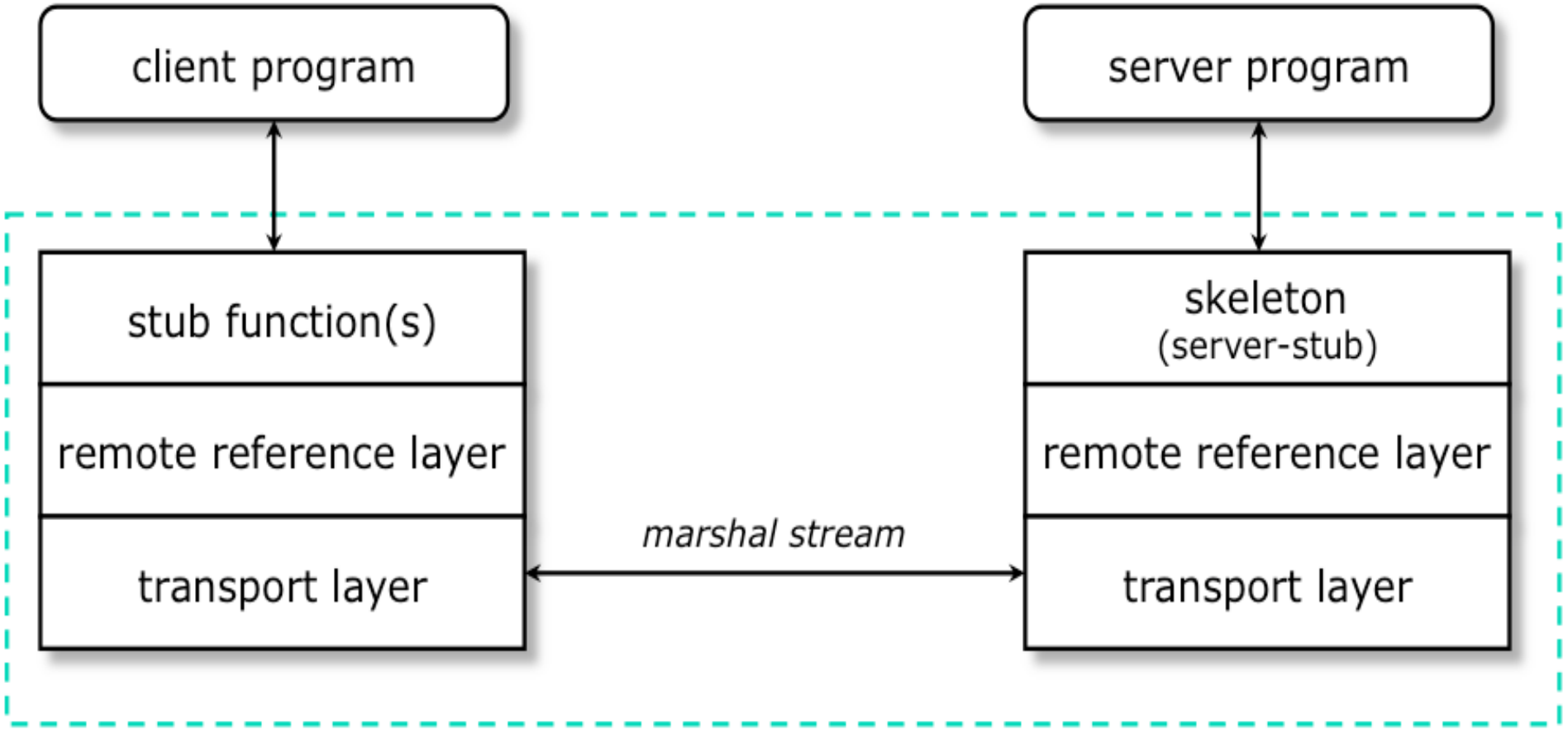
- Java RMI – sistem de obiecte distribuite integrate in Java
- Permite invocari ale metodelor obiectelor aflate in alte JVMs (Java Virtual Machines), deci in alte spatii de adrese (remote objects)
- Cu Java RMI un client poate obtine referinta unui obiect remote si poate invoca metodele acestuia ca si cum ar invoca metodele unui obiect local (din aceeasi masina virtuala)



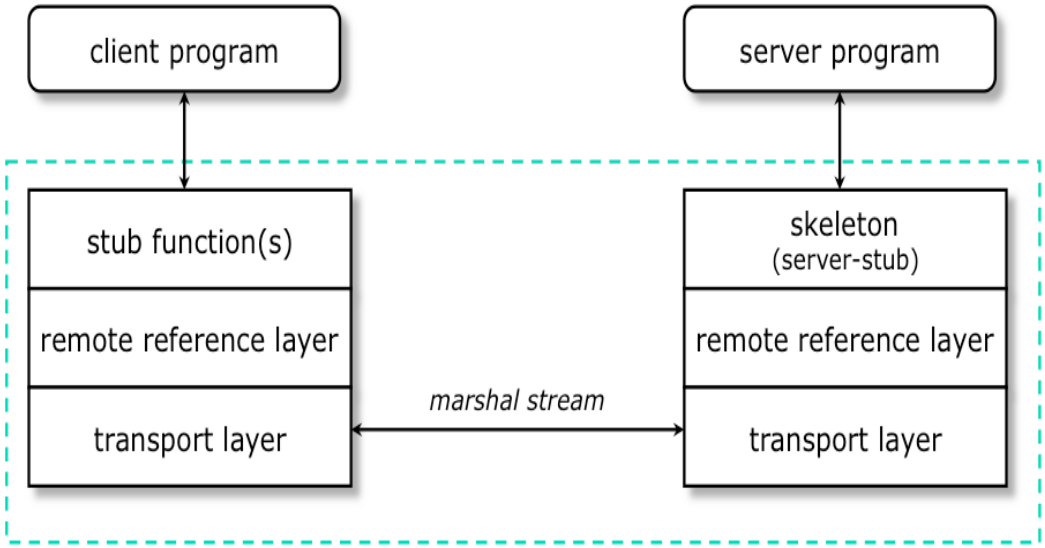
Arhitectura Java RMI

Include stub-urile client si server si modulele de comunicare cu functii similare RPC

si module pentru gestiunea referintelor remote



Rolurile Componentelor

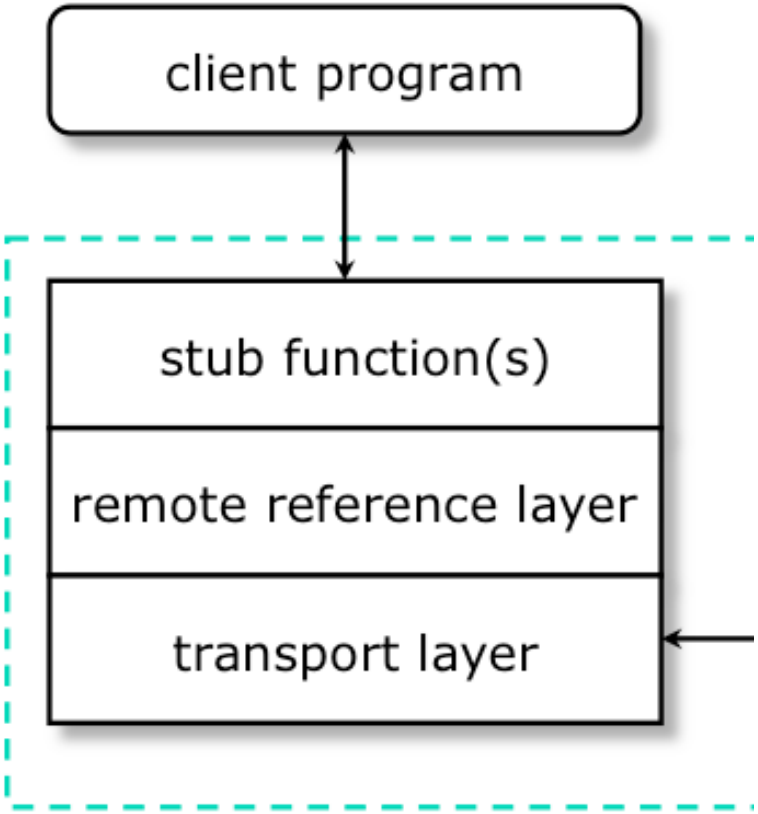


stub client

- stub-ul este un obiect local clientului, care implementeaza aceleasi metode ca obiectul remote
- un client initiaza o RMI apeland o metoda a obiectului stub
- stub-ul converteste (marshal) argumentele metodei intr-o forma serializata
- cere managerului de referinte remote sa transmita invocarea de metoda si argumentele obiectului remote
- asteapta apoi mesajul de raspuns, il converteste si returneaza rezultatul catre client

managerul de referinte remote

- o referinta remote contine
  - adresa de retea si portul (endpoint) serverului
  - un identificator local al obiectului remote in spatiul de adrese al serverului
- la client, managerul de referinte stabileste o conexiune de transfer cu serverul (folosind adresa de retea si portul)

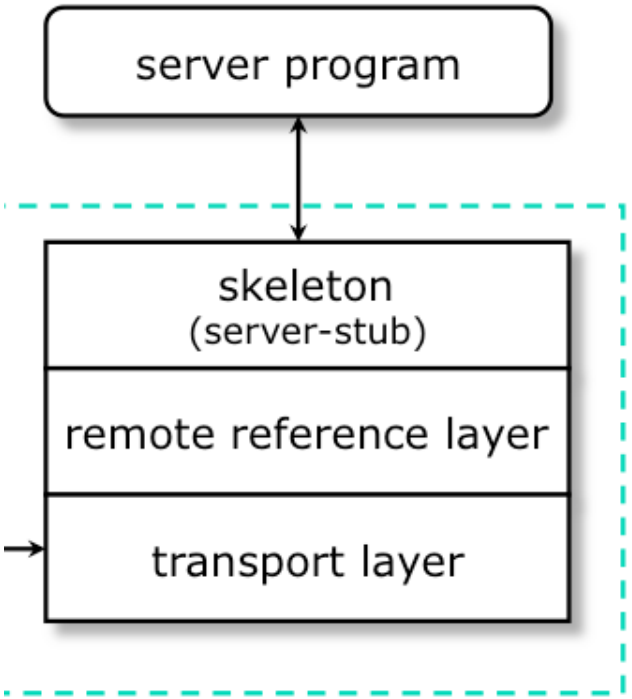


- construieste o cerere de transport pentru server, incluzand identificatorul local al obiectului remote
- paseaza cererea nivelului transport

managerul de referinte remote – la server

•la server, managerul de referinte primeste cererea de la nivelul transport

•o paseaza skeleton-ului corespunzator
identificatorului local al obiectului



skeleton la server

•converteste cererea remote intr-un apel de metoda corect (unmarshaling)

•invoca metoda obiectului remote

•Daca metoda genereaza un raspuns, trimiterea lui pe calea inversa la client reclama transformari similare la nivelele stub / skeleton si manager de referinte

Clase serializable

- Obiectele non-remote sunt pasate prin valoare; conditia este ca obiectul sa fie serializabil (sa implementeze interfata *Serializable*)
  - pentru transmitere, obiectul este serializat (marshaled) – reprezentat intr-o forma fara pointeri, ca o serie de octeti
  - forma originala a continutului se obtine prin de-serializare
  - nu toate obiectele sunt serializabile
    - ex. obiectele legate de platforma (descriptori de fisiere, socketi)
- Cand un obiect este pasat prin valoare, se creaza un nou obiect in procesul receptor
  - metodele obiectului creat pot fi invocate local
  - starea obiectului se poate schimba, putand deveni diferita de cea a obiectului din procesul transmitator

Descrierea unei clase remote

Descrierea unei clase remote are doua parti

- definirea interfetei
  - care sta la baza definirii clasei remote
- definirea clasei propriu zise – de fapt doua clase diferite
  - una la server - descrie implementarea interfetei
    - obiectele clasei exista in spatiul de adrese al serverului
    - un obiect poate contine si metode care nu apar in interfata si pot fi invocate local
  - alta la client
    - descrie functionalitatea proxy

Clase remote

- Un obiect remote este pasat ca parametru referinta
  - obiectele sunt identificate printr-o referinta (object handle) care poate fi
    - (ineficient ca volum transferat) un proxy (care reprezinta local obiectul remote - ale carui metode pot fi invocate local de un client) – trebuie sa fie serializabil
    - (mai eficient) o specificatie a claselor necesare pentru a construi stubul (proxy-ul) la client
  - obiectele trebuie sa ie serializabile

Un exemplu

Definitia interfetei HelloInterface.java

```
import java.rmi.*;
public interface HelloInterface extends Remote {
        // interfata HelloInterface extinde interfata Remote
        // implica includerea unei RemoteException
    public String say () throws RemoteException;
}
```

Clasa Remote Hello.java

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

public class Hello
 extends UnicastRemoteObject
                                  // permite obtinerea unei referinte
 implements HelloInterface {
  private String message;
  public Hello (String msg) throws RemoteException{
     message = msg;          // mesaj standard
   }
   public String say() throws RemoteException {
      return message;  // intoarce mesajul standard
   }
 }
```

Serverul (HelloServer.java)

```
import java.rmi.*;
import java.rmi.server.*;
public class HelloServer {
 public static void main (String argv[]) {
 try {
Naming.rebind ("Hello", new Hello ("Hello, world!"));
      System.out.println ("Hello Server is ready.");
    } catch (Exception e) {
      System.out.println ("Hello Server failed: " + e);
    }
 }
}
```

instantiaza obiectul server cu mesajul “Hello, world!”

inregistreaza serviciul cu RMIregistry

asociaza numele “Hello” cu obiectul remote

Clientul (HelloClient.java)

```
import java.rmi.*;
public class HelloClient {
 public static void main (String argv[]) {
  try {
// clientul apeleaza RMI registry pentru a afla referinta obiectului remote
// specifica un nume (Hello) in format URL:  //NumeMasina:port/NumeObiect
// Naming.lookup returneaza referinta (proxy hello)
    HelloInterface hello =
    (HelloInterface) Naming.lookup("//localhost/Hello");
// clientul invoca metoda “say” si afisaza rezultatul
    System.out.println (hello.say());
  }
  catch (Exception e) {
  System.out.println ("HelloClient exception: " + e);}
 }
}
```

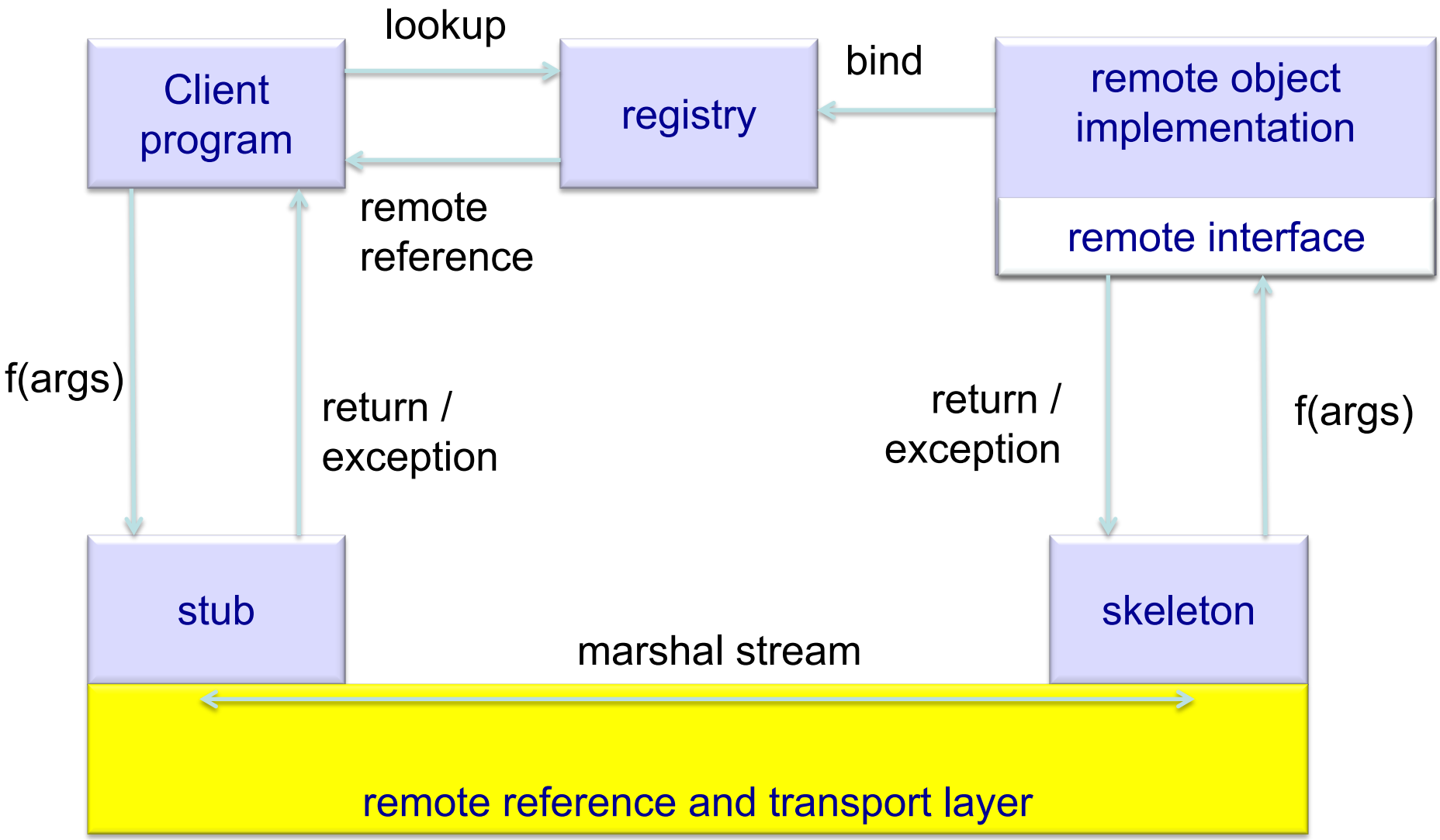
Pasii de compilare si executie

- Compilarea interfetei si implementarii
- Generarea stub client si skeleton server
- Porneste in background object registry (daca nu este in executie)
- Porneste in background serverul
- Executa clientul

Obs. in RMI, o interfata este implementata de doua clase

  - clasa remote – implementarea propriu zisa a serviciului
  - stub-ul client – care este un proxy de server
- Proxy include toate informatiile ce permit clientului sa invoce metodele obiectului remote
  - este serializabil (deci poate fi transmis de la un proces la altul)
- Cand se paseaza referinta obiectului server, daca clientul nu detine clasa proxy-ului, codul ei este descarcat automat

Configuratia de componente si invocari



Invocari dinamice de metode

- Invocarile pot fi generate si dinamic, la executie
  - utilitate – adaugarea unor servicii noi nu reclama re-compilarea aplicatiilor
  - apropie RMI de arhitectura orientata pe servicii
- Invocarile dinamice folosesc o forma generica

  invoke (obiect, metoda, in_param, out_param)

care include

referinta la obiectul remote

numele metodei si

parametrii

- Invocarile se genereaza la executie pe baza informatiilor primite de la un director de servicii

Despre interoperabilitatea RMI

RMI construit initial doar pentru Java

Nu a avut obiective legate de interoperabilitatea

- cu diverse sisteme de operare (precum CORBA)
- cu alte limbaje (precum SUN, DCE, CORBA)
- cu alte arhitecturi

Nu are nevoie de o reprezentare externa a datelor

- toate partile participante ruleaza o JVM

Beneficiu: simplitatea si designul clar

Versiuni Java RMI

- RMI initial ca parte a nucleului Java API – JDK 1.1, 1997
  - foloseste protocolul Java Remote Method Protocol (JRMP) care ruleaza peste TCP
- RMI-IIOP (RMI peste IIOP - Internet Inter-ORB Protocol) este o versiune ulterioara care elimina limitarea la JVMs
  - de regula, Java remote objects pot comunica cu obiecte CORBA scrise in alte limbaje de programare
  - IIOP este protocolul folosit pentru trimiterea invocarilor de metode prin ORBs (Object Request Brokers), dezvoltat in standardul CORBA (Common Object Request Broker Architecture)
- Alte modificari ale Java RMI includ posibilitatea de a folosi HTTP ca transport

Despre CORBA

CORBA - Common Object Request Broker Architecture

- construit ca standard independent de limbaj si masina pentru sisteme distribuite (C, C++, Lisp, Java, COBOL, Pyton …)
- scop – sa poata integra aplicatii bazate pe tehnologii mai vechi
- interfetele serviciilor sunt descrise intr-un limbaj neutru IDL (Interface Definition Language) si pot fi implementate in orice limbaj (orientat sau nu pe obiect) pentru care exista o mapare cu IDL
- pentru invocari de metode s-a definit ORBs (Object Request Brokers) care comunica prin IIOP (Internet Inter-ORB Protocol)
  - foloseste o sintaxa de transfer unica pentru a facilita independenta de limbaj si masina
  - tranzactiile IIOP se pot face prin TCP sau alte protocoale (ex. HTTP)

Bibliografie

- Ethan Cerami, Web Services Essentials. Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. O'Reilly 2002
- George Coulouris, Jean Dollimore,Tim Kindberg, Gordon Blair. Distributed Systems Concepts and Design, ed. 5, Addison Wesley
- Java RMI Tutorial

  http://www.eg.bucknell.edu/~cs379/DistributedSystems/rmi_tut.html

  file:///Users/vnpcristea/Documents/___curs%20sprc%202014/docs/Java%20RMI%20Tutorial.html
- Chapter 3. Remote Method Invocation

  http://docstore.mik.ua/orelly/java-ent/jenut/ch03_01.htm