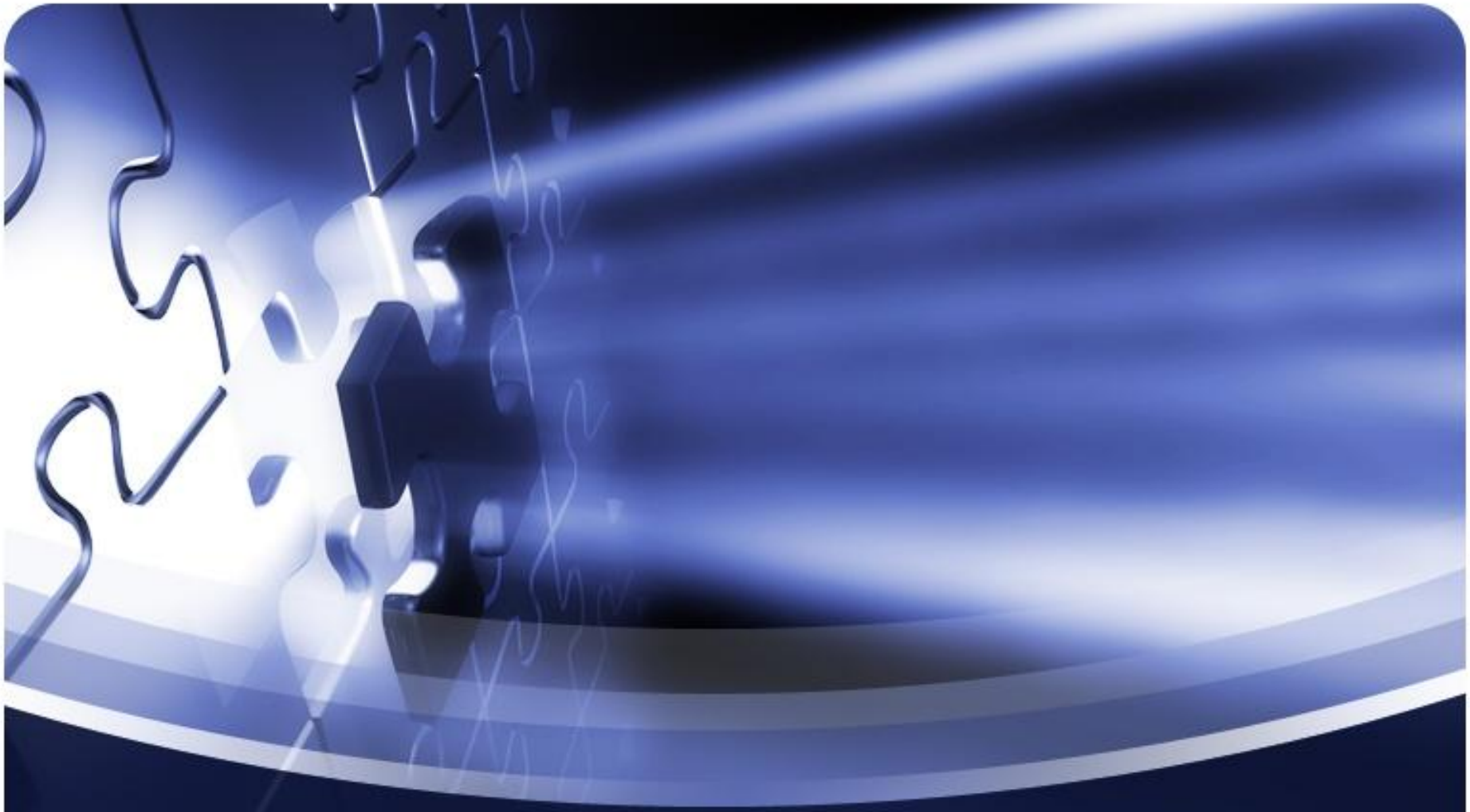


Distribuirano programiranje

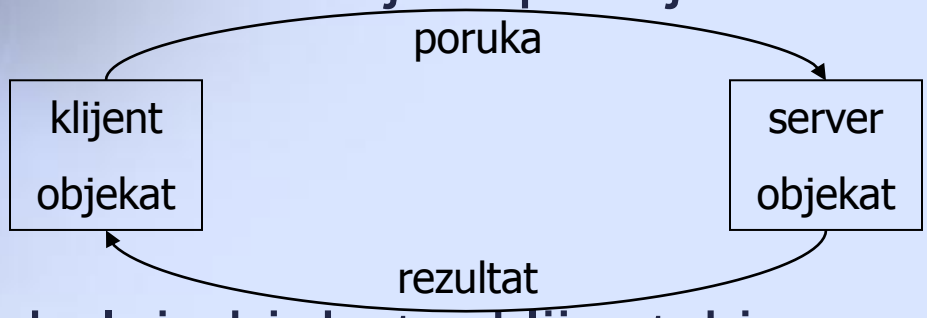
Programski jezici II

Tehnologije

- RMI
  - Remote Method Invocation
- CORBA
  - Common Object Request Broker Arhitecture

Distribuirani objekti

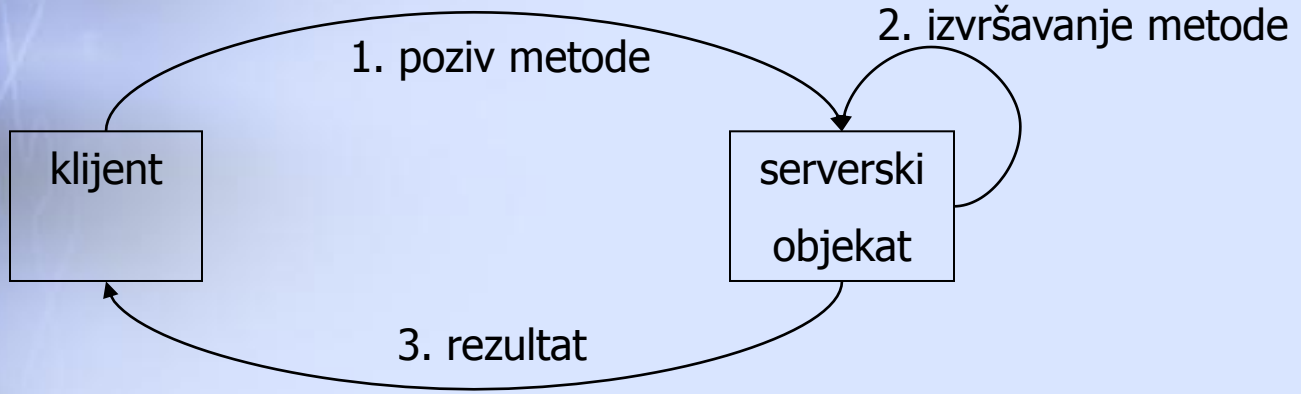
- posmatra se objekat koji “živi” na udaljenom računaru
  - kod Java tehnologija – unutar druge JVM
- njemu se mogu poslati poruke (tj. pozvati njegove metode) i preuzeti rezultati na isti način kao da je u pitanju lokalni objekat



- lokalni objekat – klijentski
- udaljeni objekat – serverski

Distribuirani objekti

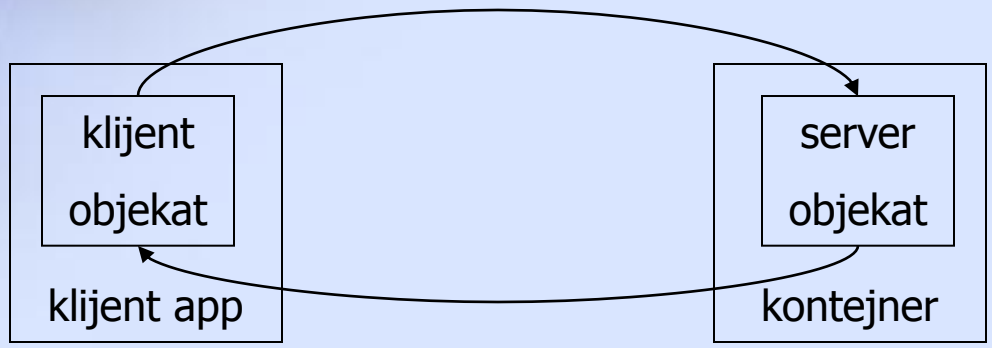
- poziv metode serverskog objekta – podrazumijeva izvršavanje te metode na onom računaru na kojem se nalazi sam objekat (udaljeni računar)



- klijentski program se izvršava na više računara:
  - inicijalno na onom na kojem je pokrenut
  - i na svim računarima na kojima se nalaze serverski objekti koje on koristi

Distribuirani objekti

- ovakva komunikacija može se posmatrati kao komunikacija 2 objekta:
  - klijent objekat je sastavni dio neke aplikacije
  - server objekat se nalazi u “kontejneru” koji obezbjeđuje mrežne i druge servise



Distribuirani objekti

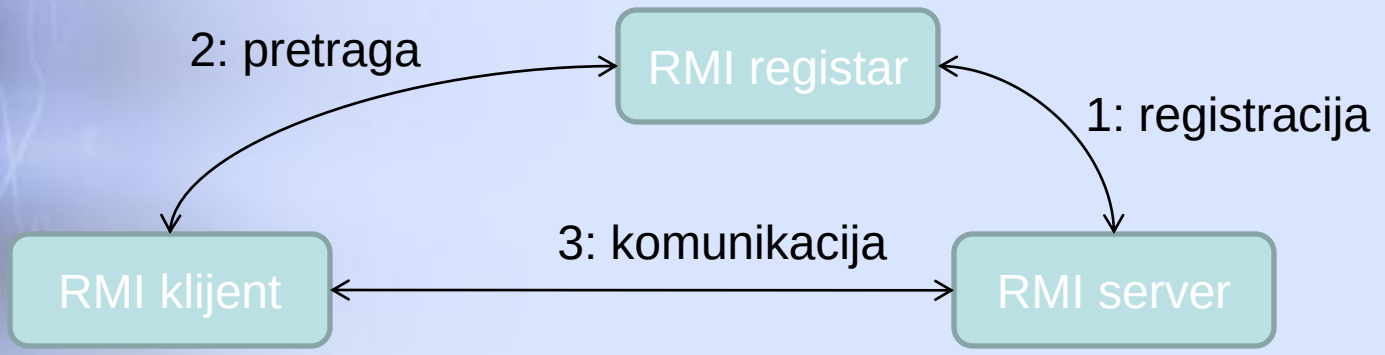
- sa stanovišta autora klijentske aplikacije, i klijentski i serverski objekat su sastavni dio jednog programa
- osobina ovakvog programa je da se izvršava na više računara u mreži
- logički posmatrano – serverski program ne postoji
- postoji serverski objekat koji se ponaša kao da je dio klijentskog programa
- važno: programer ne vodi računa o protokolu kojim komuniciraju klijentski i serverski objekti

RMI

- *Remote Method Invocation*
- osnovna Java tehnologija za pisanje distribuiranih aplikacija
- osnovne komponente: RMI klijent, RMI server i RMI registry
- klijentski objekat ne “vidi” direktno serverski objekat, nego Java interfejs koga ovaj implementira – RMI interfejs

RMI

- serverski objekat se registruje kod RMI registra

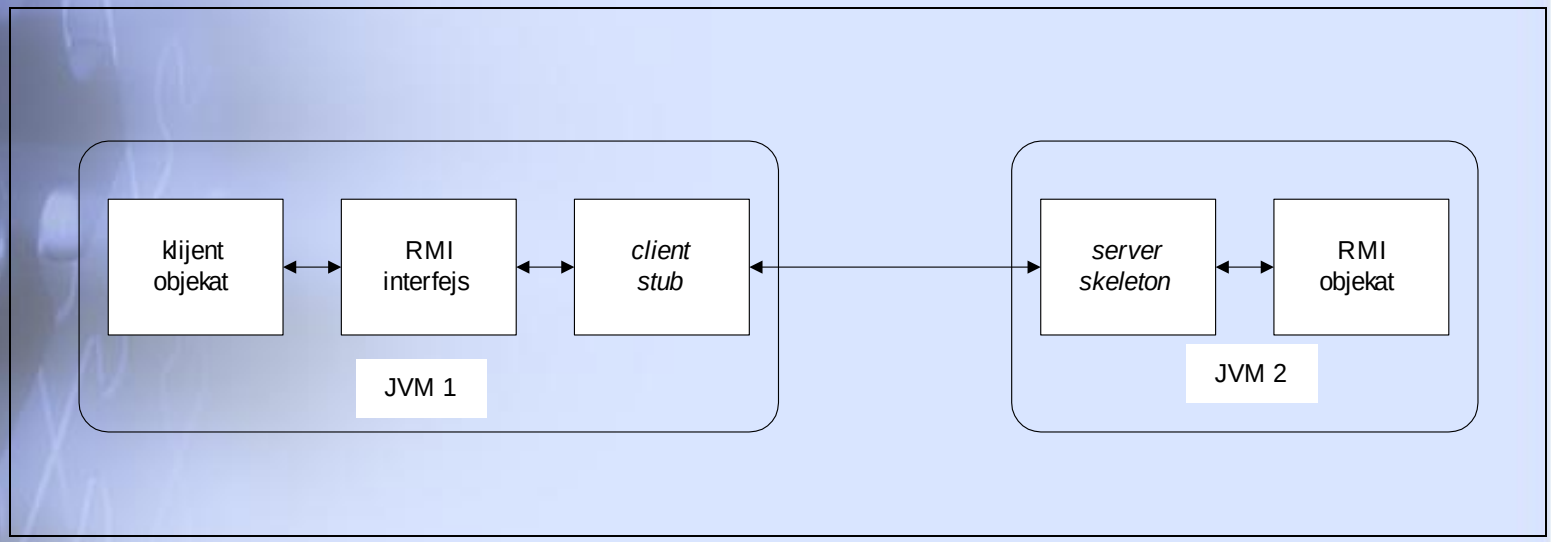


- RMI klijent pretražuje RMI registar i uzima referencu na serverski objekat
- RMI klijent koristi serverski objekat kao da je riječ o lokalno instanciranom objektu

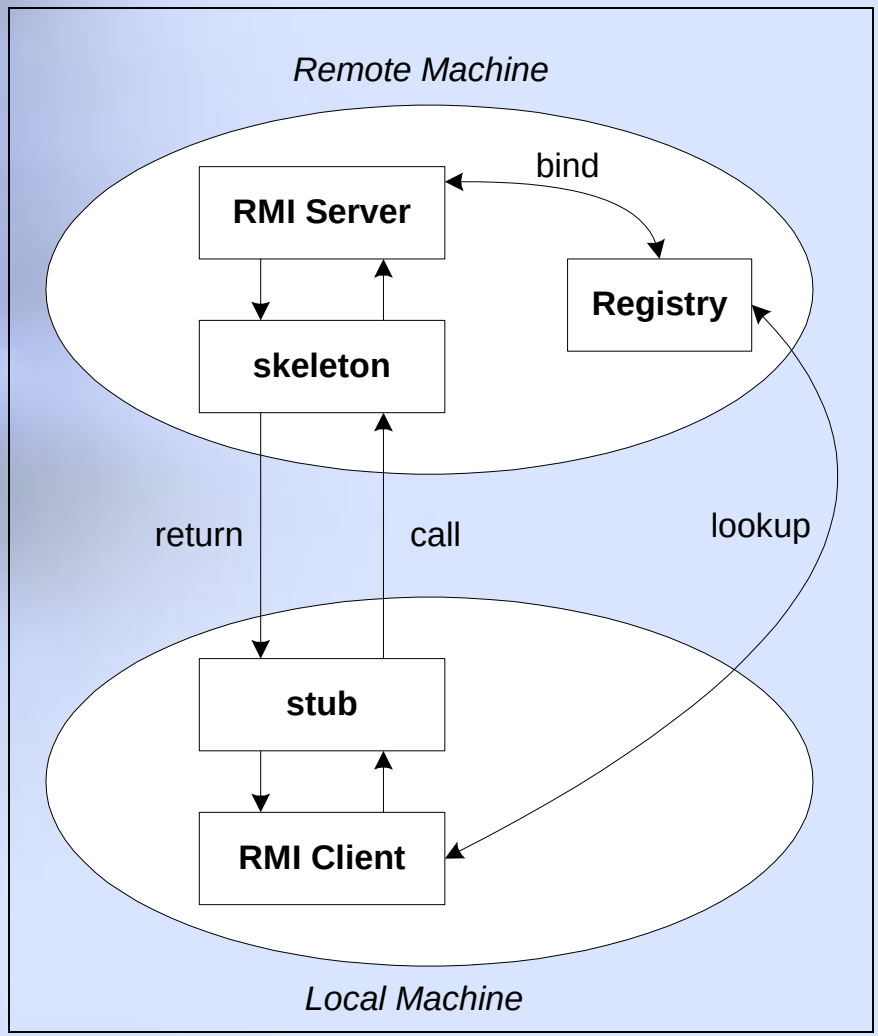
RMI

- komunikacija klijenta i servera odvija se po JRMP (*Java Remote Method Protocol*) protokolu
- za komunikaciju su zadužene posebne, automatski generisane klase:
  - *stub* klasa (za JRMP verzija 1.2), odnosno
  - *stub* i *skeleton* klase (za JRMP verzija 1.1)
- ove klase generišu se posebnim JDK alatom **rmic**
- osnovni zadatak *stub* klase – da pozive metoda serverskog objekta koje upućuje klijentski objekat proslijedi preko mreže – ova klasa je klijentski proxy za serverski objekat
- *skeleton* klasa – da pozive pristigle preko mreže pretvori u pozive stvarnog serverskog objekta

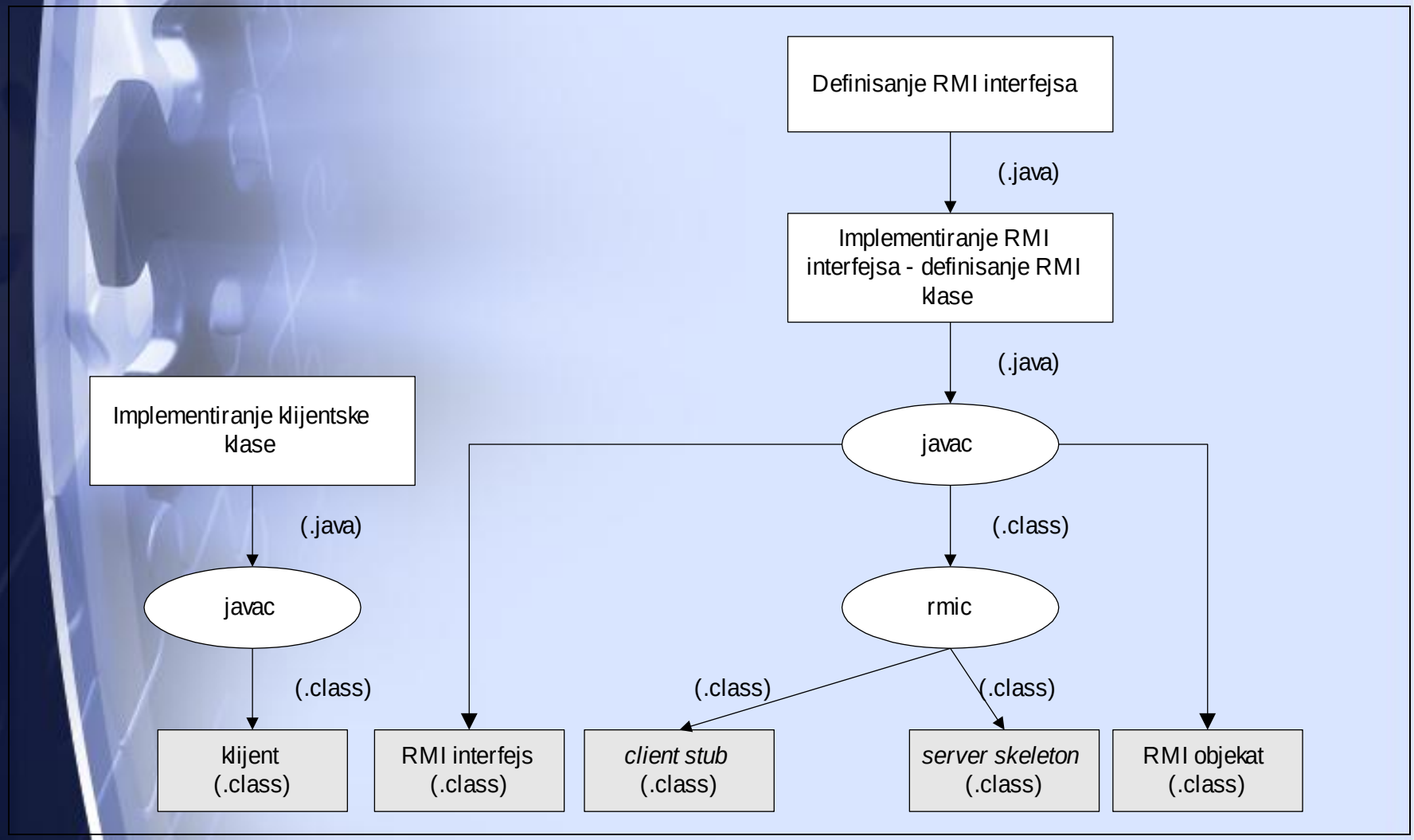
Učesnici u komunikaciji



RMI


Remote Machine
bind
RMI Server
Registry
skeleton
return
call
lookup
stub
RMI Client
Local Machine

Pisanje RMI programa



RMI interfejs

- Java interfejs koji ispunjava dva uslova:
  - nasljeđuje interfejs java.rmi.Remote
  - sve njegove metode moraju da bacaju izuzetak java.rmi.RemoteException

```
interface CalculatorInterface extends Remote {
   int add(int x, int y) throws RemoteException;
   int sub(int x, int y) throws RemoteException;
}
```

Metode RMI interfejsa

- argumenti i rezultat metoda se prenose po vrijednosti (pass-by-value), bez obzira da li je riječ o vrijednostima primitivnog tipa ili objektima

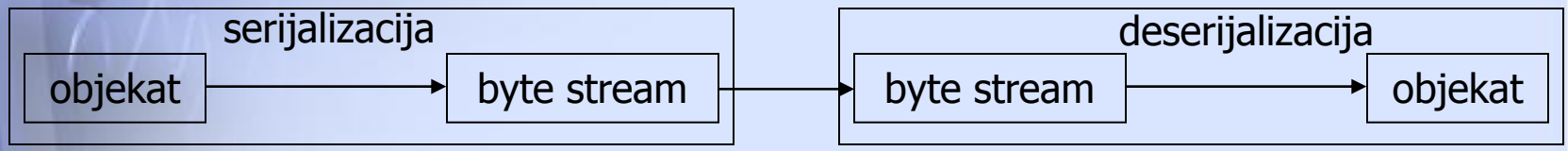
- primjer

```
interface ServerI extends Remote {
  SomeClass method(SomeClass x, int y)
    throws RemoteException;
}
```

parametri i rezultat se prenose po vrijednosti

Metode RMI interfejsa

- kod RMI-ja prenose se objekti, tj. njihov sadržaj se mora proslijediti – objekat se prenosi po vrijednosti, a ne reference
- parametri i rezultati metoda moraju biti serijalizabilni, tj. moraju da implementiraju standardni Serializable interfejs (markerski interfejs)



- primjer:

```
public class SomeClass implements Serializable {
...
}
```

RMI serverski objekat

- instanca klase koja mora da zadovolji sljedeće uslove:
  - implementira svoj RMI interfejs
  - nasleđuje java.rmi.UnicastRemoteObject
- primjer

```
public class CalculatorServer extends
    UnicastRemoteObject implements
            CalculatorInterface {
...
}
```

RMI serverski objekat

- inicijalizacija (tipično u main() metodi)
  - postavlja se novi SecurityManager

```
System.setSecurityManager(new
RMISecurityManager());
```

  - kreira se serverski objekat

```
CalculatorServer server = new
CalculatorServer();
```

  - registruje se serverski objekat

```
Naming.rebind(
  "//student.etfbl.net:1099/Server", server);
```

RMI registry

- serverski objekti se moraju registrovati pod nekim imenom kod servera koji služi kao katalog ovakvih objekata
- klijenti koji žele pristupati serverskom objektu će od ovog servera zahtijevati referencu na odgovarajući objekat putem imena pod kojim je registrovan
- rmiregistry - program koji obezbjeđuje RMI serverske usluge

RMI registry

- mora biti aktivan na serverskoj mašini
- očekuje klijente na TCP portu 1099 (default)
- pokreće se sa rmiregistry:
  - Windows – start rmiregistry
  - Unix – rmiregistry &
- može se pokrenuti i iz Java programa:

```
LocateRegistry.createRegistry(1099);
```

RMI klijent

- postavlja novi SecurityManager

```
System.setSecurityManager(new
RMISecurityManager());
```

- pronalazi svoj serverski objekat

```
CalculatorInterface server =
(CalculatorInterface)Naming.lookup(
  "//student.etfbl.net:1099/Server");
```

- koristi svoj serverski objekat

```
System.out.println("Count: " + server.add(1,2));
```

RMI i sigurnost

- potencijalni problem – kod koji se preuzima sa udaljene lokacije
- RMI class loader neće preuzeti kod sa udaljene lokacije, ako nije postavljen security manager i ako on ne dozvoljava preuzimanje koda sa date lokacije
- RMISecurityManager (nasljeđuje SecurityManager) – koristi se za RMI aplikacije

- RMISecurityManager se ne primjenjuje na aplete, koji se izvršavaju pod security manager-om Web čitača
- upotreba:

```
System.setSecurityManager(new RMISecurityManager());
```

- mora se izvršiti prije nego što RMI učitava kod sa udaljenog hosta – main metoda

RMI i sigurnost

- sigurnosna politika

```
grant codeBase "file:/E:/ETF BL/calculator" {
 permission java.security.AllPermission;
};

grant codeBase "file:/E:/ETF BL/calculator" {
 permission java.net.SocketPermission "localhost:1099",
    "connect";
 permission java.net.SocketPermission "localhost:1024-",
    "accept, resolve";
};

grant codeBase "file:/E:/ETF BL/calculator" {
permission java.net.SocketPermission "localhost:1024-",
    "connect, resolve";
};

java -Djava.security.policy=server_policyfile.txt
    CalculatorServer
java -Djava.security.policy=client_policyfile.txt
    CalculatorClient
```

- policytool

RMI i konkurentnost

- prilikom registracije servera registruje se konkretna instanca serverske klase – nju dijele svi klijenti koji su se sa serverom povezali preko istog imena
- istovremeni pozivi RMI objekta od strane više klijenata se obrađuju u posebnim programskim nitima
- potencijalni problem: sinhronizacija
- rješavanje problema: sinhronizovane metode, sinhronizovani blokovi, API visokog nivoa

CORBA

- *Common Object Request Broker Architecture*
- standardizovana arhitektura sistema distribuiranih objekata
  - nezavisna od jezika implementacije – omogućava da distribuirane i heterogene kolekcije objekata međusobno komuniciraju
  - nezavisna od proizvođača softvera: OMG (*Object Management Group*) standard
  - najsloženija tehnologija za rad sa distribuiranim objektima – podržava najširi spektar zahtjeva u okviru okruženja distribuiranih objekata

CORBA

- ista ideja kao kod RMI – klijentski objekat poziva metode serverskog objekta kao da je u pitanju lokalni objekat

- *Object Request Broker* (ORB) – softverski modul namjenjen za mrežnu komunikaciju
- *Internet Inter-ORB Protocol* (IIOP) – protokol kojim komuniciraju ORB-ovi

Interface Definition Language (IDL)

- jezik za specifikaciju interfejsa serverskih objekata (servanata) – univerzalan način
- na osnovu IDL interfejsa generiše se implementacija serverskog objekta u određenom programskom jeziku
- mapiranja IDL → <jezik_implementacije> postoje za mnoge jezike:
  - Java
  - C
  - C++
  - Ada
  - Lisp
  - Smalltalk
  - COBOL
  - Perl
  - Delphi
  - ...

IDL

- Korištenjem IDL jezika moguće je opisati:
  - modularizovane interfejse serverskih objekata,
  - operacije i atribute koje objekat podržava,
  - izuzetke koje operacije mogu baciti,
  - povratne tipove operacija i njihove parametre.
- IDL tipovi podataka su:
  - osnovni tipovi podataka – long, short, string, float i dr.,
  - konstruisani tipovi podataka – struct, union, enum sequence,
  - reference i
  - any tip.

IDL – mapiranja tipova

IDL	Java
boolean	boolean
char, wchar	char
octet	byte
short, unsigned short	short
long, unsigned long	int
long long, unsigned long long	long
float	float
double	double
string, wstring	String

IDL – mapiranja ključnih riječi

IDL	Java
module	package
operation	method
interface	interface
attribute	par metoda
exception	exception

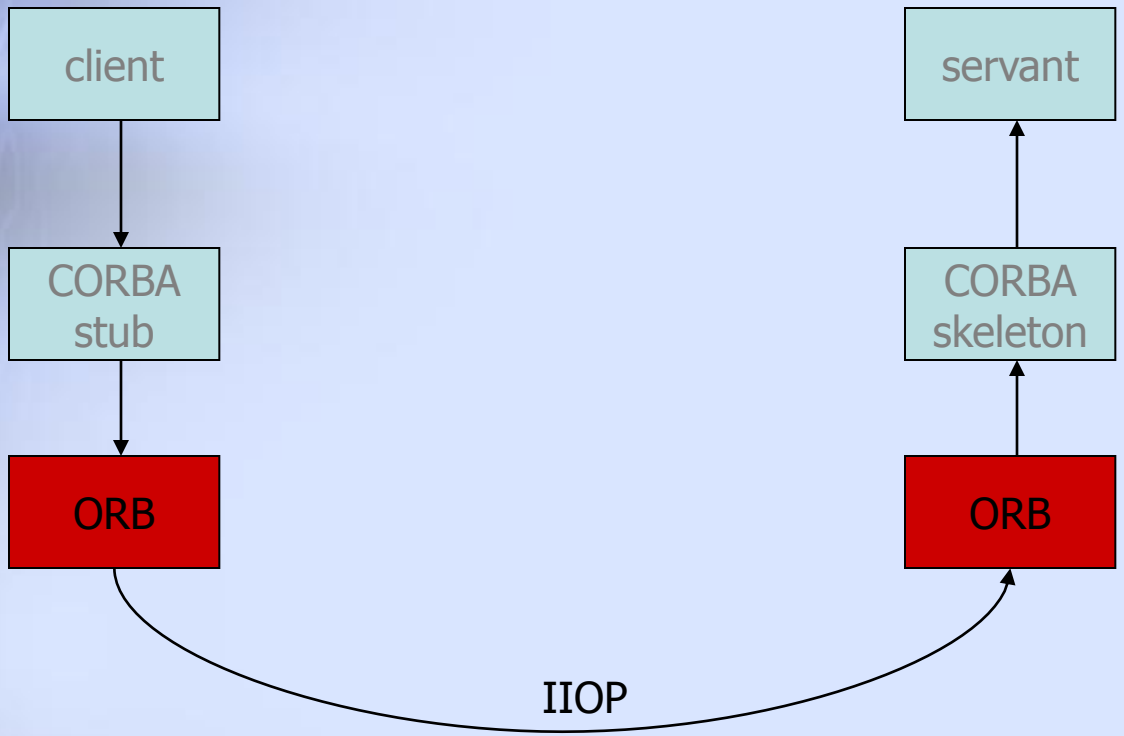
IDL → Java

- mapiranje definisano OMG standardom
- **idlj**: dio JDK paketa

```
module calculator {
  interface Calculator {
    long add(in long x, in long y);
    long sub(in long x, in long y);
  };
};
```

IDL → Java

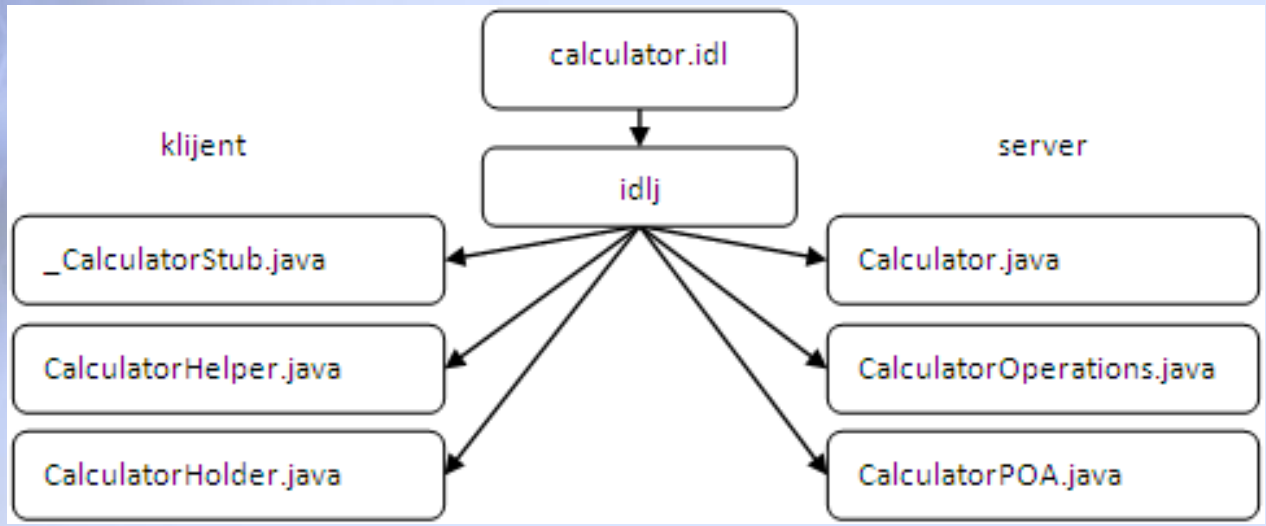
- rezultat rada **idlj** alata nije samo Java interfejs, već i *stub* i *skeleton* klase – one koriste ORB za komunikaciju sa objektom sa "druge strane"



IDL → Java

- mapiranje definisano OMG standardom
- **idlj**: dio JDK paketa

```
idlj -fall calculator.idl
```



- Implementiranje servera: **CalculatorServer.java**
- Implementiranje servanta: **CalculatorServant.java**
- Implementiranje klijenta: **CalculatorClient.java**

CORBA Naming Service

- poseban servis namijenjen za katalogizaciju serverskih objekata (slično rmiregistry-ju kod RMI)
- obraćaju mu se klijenti kada žele da pronađu serverski objekat
- registrovani objekti su organizovani u stablo, nalik direktorijumima i datotekama
- ovaj servis obezbjeđuje poseban server (*orbd*)
- analogija: DNS

CORBA Naming Service

- pokretanje orbd:

```
orbd -ORBInitialPort 900
```

- za pristup ovom servisu iz Jave, koristi se JNDI (*Java Naming and Directory Interface*) biblioteka – sastavni dio standardne Java biblioteke
- Server Manager za orbd:

```
servertool -ORBInitialPort 1050
```

Pokretanje CORBA aplikacije

- Pokretanje naming service-a

```
start orbd –ORBInitialPort 900
```

- Pokrenje servera

```
start java CalculatorServer -ORBInitialHost
localhost  –ORBInitialPort 900
```

- Pokretanje klijenta

```
java CalculatorClient –ORBInitialHost
localhost –ORBInitialPort 900
```

CORBA izuzeci

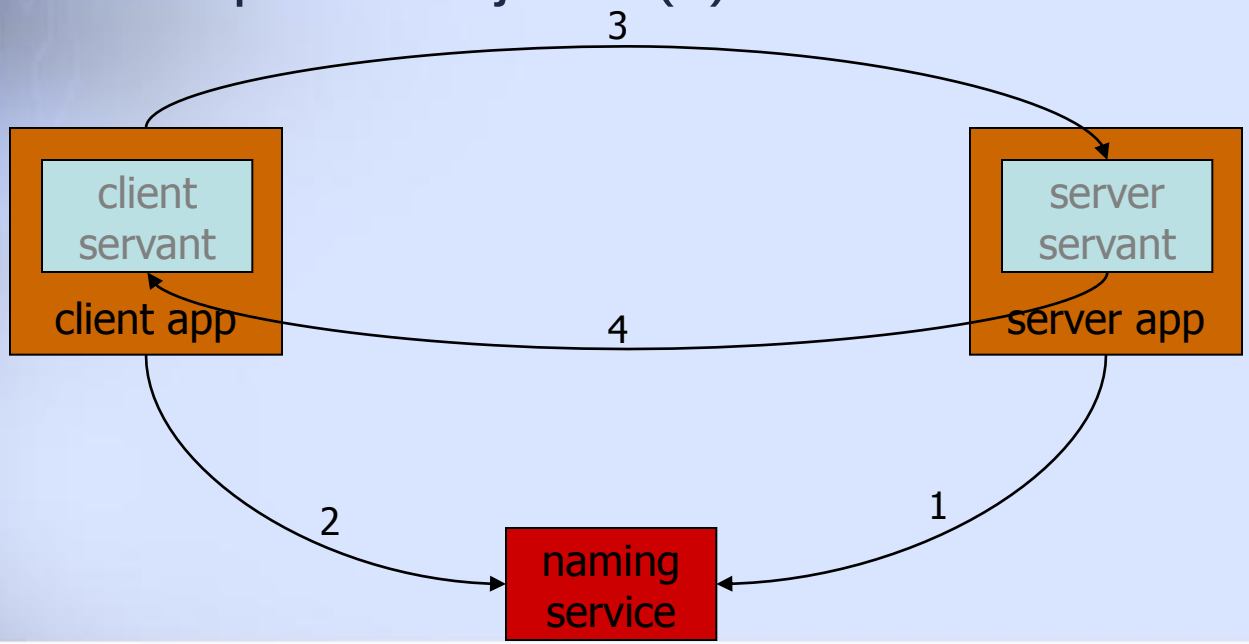
```
module calculator {
  interface Calculator {
    exception wrongArgument{};
    long add(in long x, in long y);
    long sub(in long x, in long y);
    long mul(in long x, in long y);
    long div(in long x, in long y) raises (wrongArgument);
  };
};
```

Callback

- kada je potrebno da server pošalje poruku klijentu
- i klijent i server moraju imati servant objekte koji će se pozivati
- dovoljno je samo servera registrovati kod *naming service*-a; klijent će mu poslati svoju referencu

Callback

- server registruje svog servanta (1)
- klijent traži servera po imenu (2)
- klijent poziva servera i šalje mu referencu na svog servanta (3)
- server poziva klijenta (4)



RMI ↔ CORBA

- RMI koristi JRMP
- CORBA koristi IIOP
- RMI-over-IIOP: premošćavanje razlike

- mogućnosti su svedene na “presjek skupa mogućnosti” dvaju tehnologija