

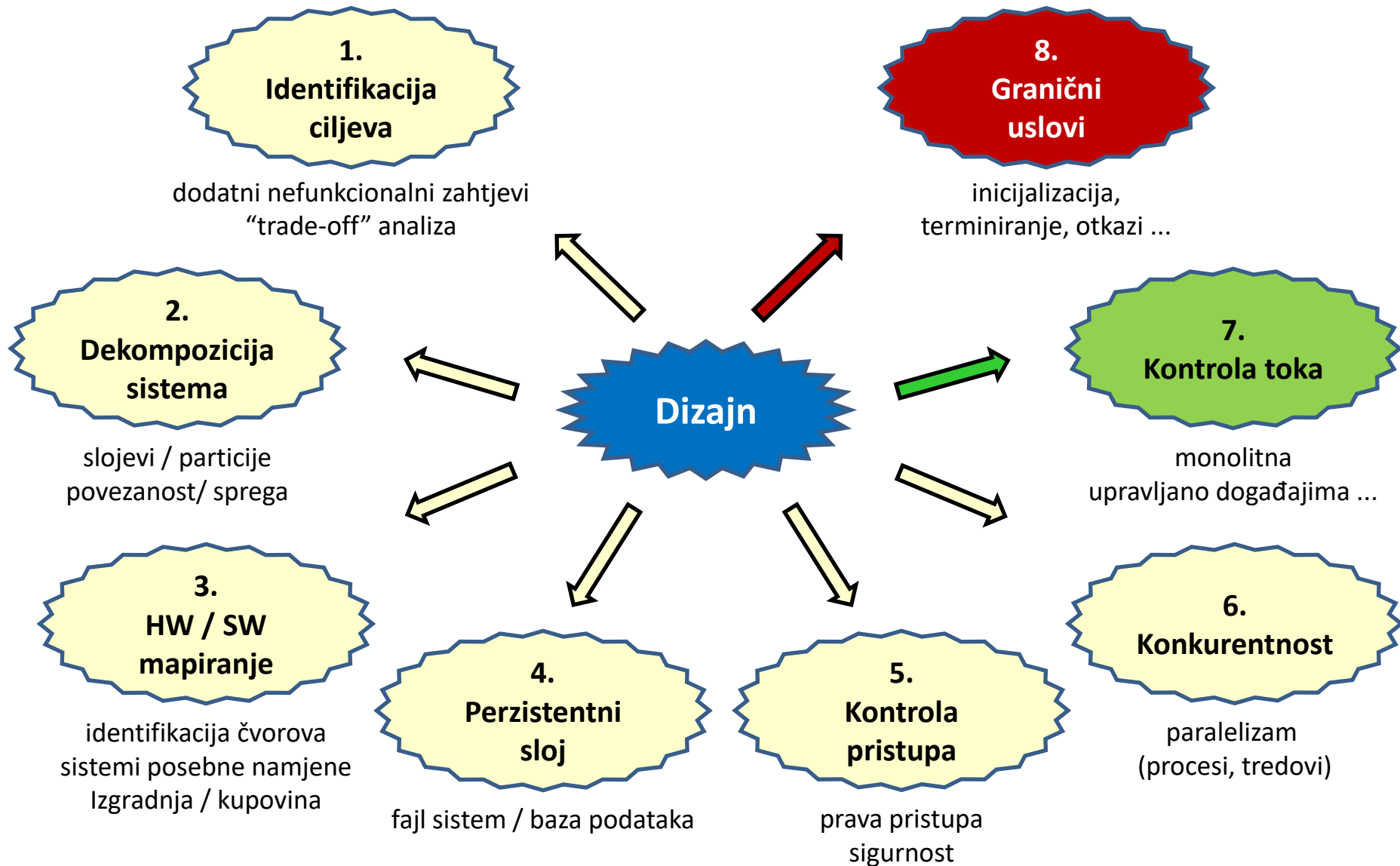
**UNIVERZITET U BANJOJ LUCI**  
**ELEKTROTEHNIČKI FAKULTET**

**Prof. dr Dražen Brđanin**

**PROJEKTOVANJE SOFTVERA**  
**/kontrola toka/**

**Banja Luka**  
**2024.**

# 8 bitnih aktivnosti u projektovanju



# 7. Kontrola toka

## Osnovni oblici kontrole toka

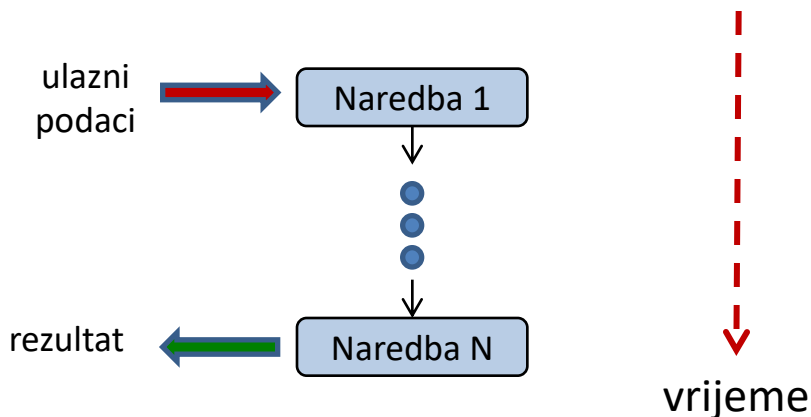
- Kontrola toka predstavlja redoslijed izvršavanja akcija/naredbi
- U O-O sistemima:
  - kontrola toka uključuje redoslijed izvršavanja metoda
  - kontrola toka zavisi od:
    - eksternih događaja (poruka) koje generišu učesnici,
    - vremenskih događaja (istek intervala, ...)
- Kontrola toka u OOA&D:
  - Analiza:
    - kontrola toka nije predmet istraživanja
    - podrazumijeva da svi objekti simultano izvršavaju operacije kad god je potrebno
  - Dizajn:
    - U obzir mora da se uzme činjenica da svaki objekat nema procesor 100% za sebe
- Dva oblika kontrole toka:
  - **centralizovana**
  - **distribuirana (decentralizovana)**

# 7.1. Centralizovana kontrola toka

## Centralizovana kontrola toka

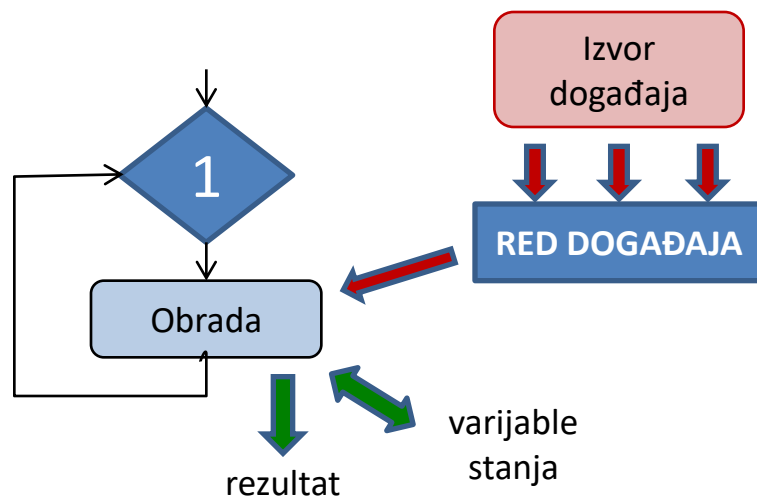
### proceduralna (*procedure-driven*)

- kontrola je u programskom kodu
  - Operacije čekaju na unos podataka kad god trebaju podatke od učesnika
  - Uglavnom u starijim sistemima implementiranim u proceduralnim jezicima
  - Nije pogodna za implementaciju konkurentnih aplikacija (koji je redoslijed sekvenciranja u slučaju većeg broja istovremenih događaja?)



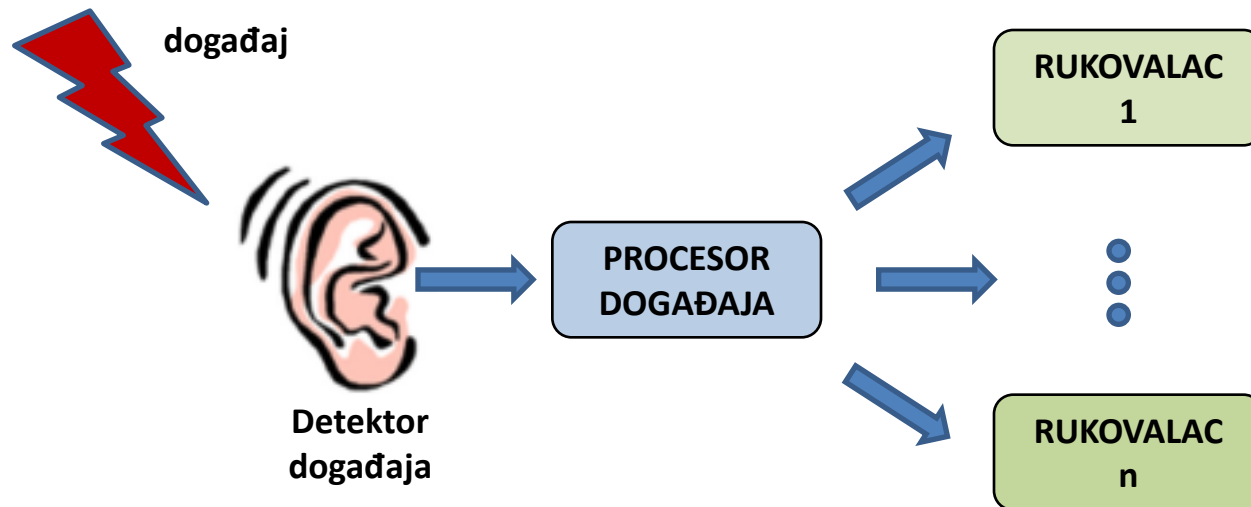
### upravljana događajima (*event-driven*)

- kontrolu ima dispečer koji poziva funkcije (*event handler*)
  - Glavna petlja čeka na spoljne događaje – kad se pojavi neki spoljni događaj, on se prosljeđuje odgovarajućem objektu, koji ga potom obrađuje
  - Ovakva glavna kontrolna petlja ima veoma jednostavnu strukturu



# 7.1. Centralizovana kontrola toka

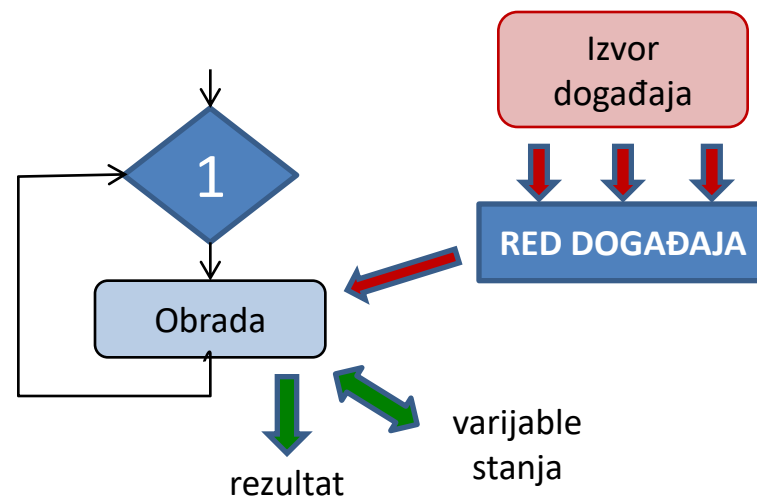
## Kontrola toka upravljana događajima (*event-driven*)



Izvori događaja su autonomni (npr. klijent na ATM, senzori, ...)

Događaji se dešavaju asinhrono

```
// upravljacka nit
while (true)
{
    e = red.getNew();
    if (e == quit) break;
    proces(e);
}
```



# 7.1. Centralizovana kontrola toka

## Kontrola toka upravljana događajima (*event-driven*)

### Tipična primjena: GUI-bazirane aplikacije

Korisnici imaju interakciju sa kontrolnim elementima (npr. dugme) i generišu događaje.

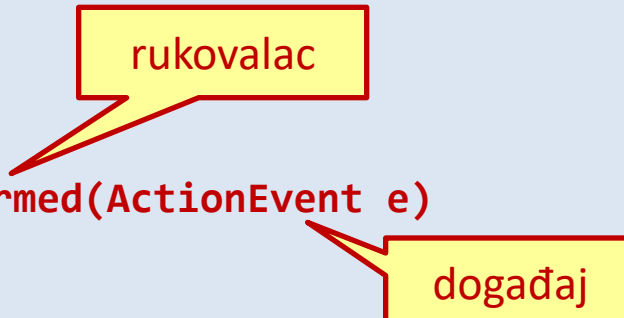
Događaje obrađuju rukovaoci događajima (***event handler***, u javi ***event listener***).

Rukovalac je “zakačen” na odgovarajuću komponentu.

Rukovaoci implementiraju odgovarajući interfejs.

```
class Panel extends JPanel implements ActionListener
{
    public Panel()
    {
        super();
        JButton btn = new JButton("OK");
        btn.addActionListener(this);
        this.add(btn);
    }

    @Override
    public void actionPerformed(ActionEvent e)
    { // obrada dogadjaja }
}
```



# 7.1. Centralizovana kontrola toka

## Kontrola toka upravljana događajima (*event-driven*)

### Rukovanje događajima

**Događaj** (*event*) se prenosi od **izvora događaja** (npr. dugme) do odgovarajućeg **rukovaoca događajima** (*event listener*)

**Izvor događaja** je objekat koji **zna kada/kako** se dogodio događaj i o tome **obavještava** zainteresovane objekte – **rukovaoce objektima**

**Rukovalac događajima** je objekat koji **treba biti obaviješten** da se desio neki događaj da bi reagovao na događaj, tj. izvršio odgovarajuću akciju

**Informacija o događaju** inkapsulirana je u odgovarajućem objektu (***ActionEvent***)

**Izvor događaja** mora biti u mogućnosti da **registruje rukovaoce** i da im **šalje objekte događaja**

**Rukovalac događajima** implementira odgovarajući **interfejs** (***ActionListener***)

Kad se desi događaj, **izvor obavještava** sve **registrovane rukovaoce** pozivom odgovarajuće metode (***actionPerformed***)

Kad **rukovalac dobije poruku** da se desio događaj, on **izvršava** odgovarajuće **akcije** ili **ignoriše** događaj

# 7.1. Centralizovana kontrola toka

## Primjer: (implementacija)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ButtonCounter
{
    public static void main(String[] args)
    {
        ButtonCounterFrame frame = new ButtonCounterFrame();
        frame.setVisible(true);
    }
}

class ButtonCounterFrame extends JFrame
{
    public ButtonCounterFrame()
    {
        // ... set up the frame and center it on the screen
        // ... create panels for the button and label

        ClickCountingLabel label = new ClickCountingLabel();
        labelPanel.add(label);

        JButton button = new JButton("Click Me!");
        button.addActionListener(label);
        buttonPanel.add(button);

        add(buttonPanel, BorderLayout.NORTH);
        add(labelPanel, BorderLayout.CENTER);

        // ... add panels to frame (to its content pane)
    }
}
```

```
class ClickCountingLabel extends JLabel implements ActionListener
{
    private static final String LABEL_TEXT_PREFIX =
        "Number of button clicks = ";

    private int numClicks = 0;

    public ClickCountingLabel()
    {
        super();
        setText(LABEL_TEXT_PREFIX + numClicks);
    }

    public void actionPerformed(ActionEvent event)
    {
        ++numClicks;
        setText(LABEL_TEXT_PREFIX + numClicks);
    }
}
```

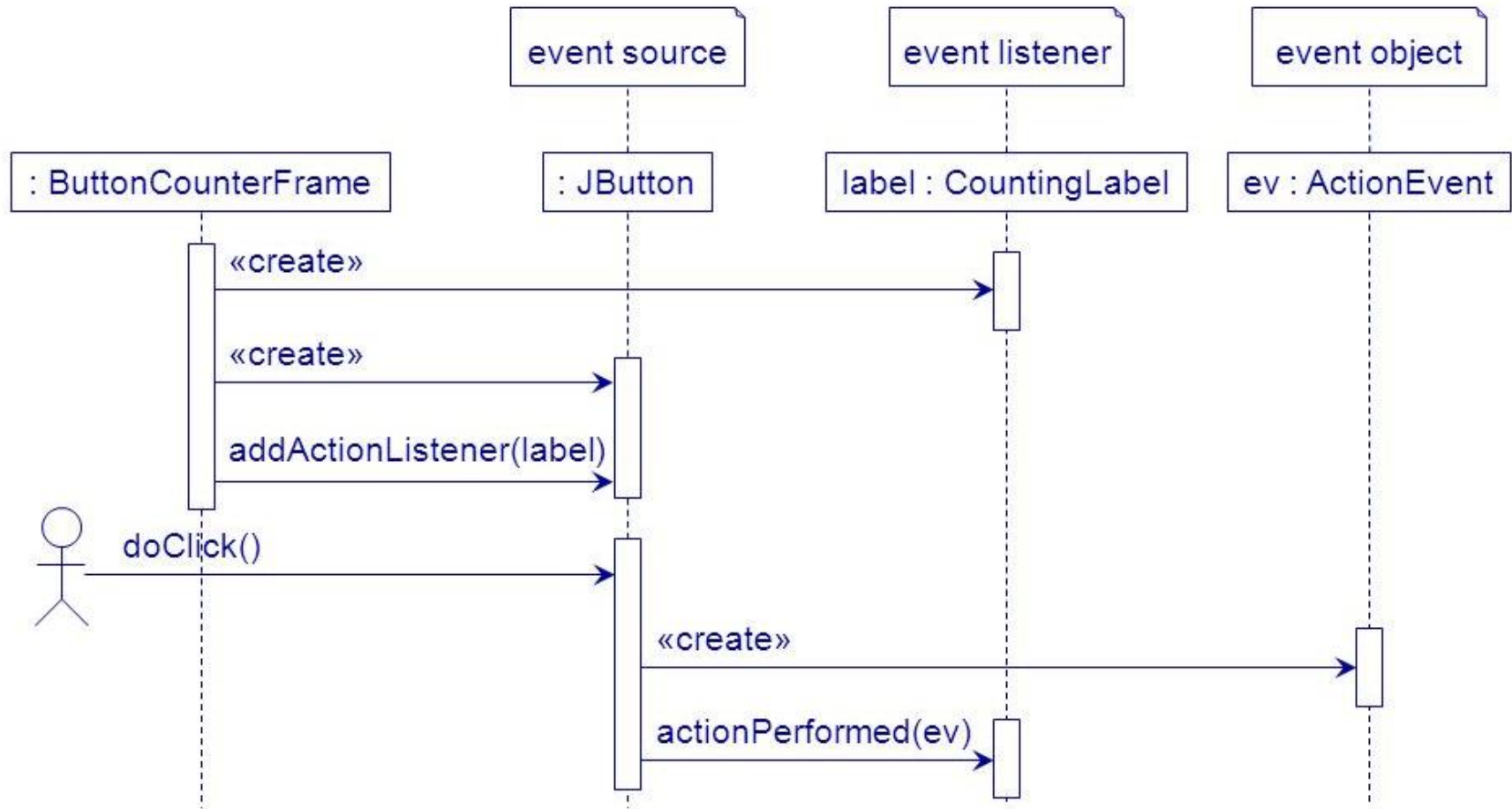


button – izvor događaja  
label – rukovalac događajima



# 7.1. Centralizovana kontrola toka

Primjer: (interakcija objekata)

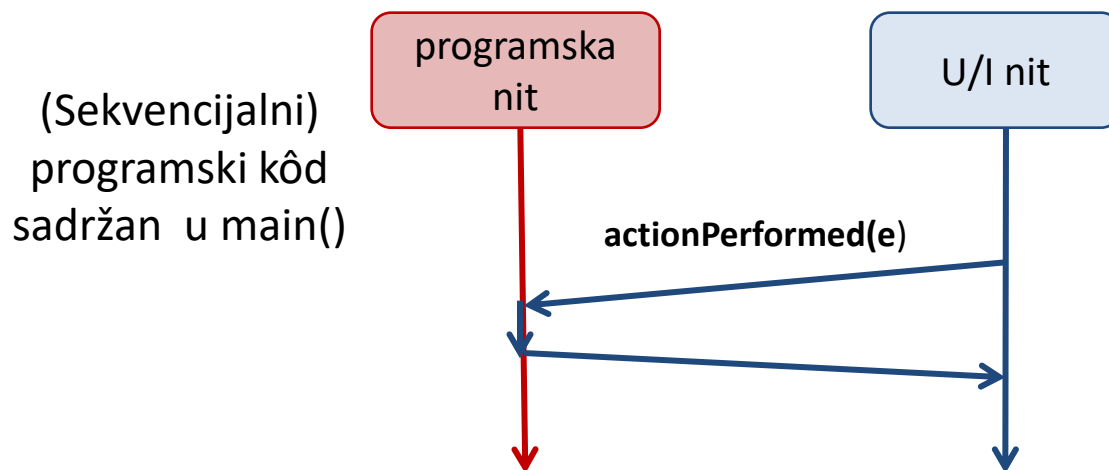


# 7.1. Centralizovana kontrola toka

## Kontrola toka upravljana događajima (*event-driven*)

### Programska nit ↔ U/I nit

Program i korisnički interfejs izvršavaju se u konkurentnim nitima



Sve U/I akcije dešavaju se u U/I niti

Rukovaoci treba da sadrže minimalni kôd – ekran će se zamrznuti ako ima dosta koda

**Vremenski zahtjevan kôd može biti realizovan u trećoj niti (*SwingWorker*) – *doInBackground()***

### Java GUI

Implementacija slijedi **opserver** (*publish/subscribe*) projektni obrazac

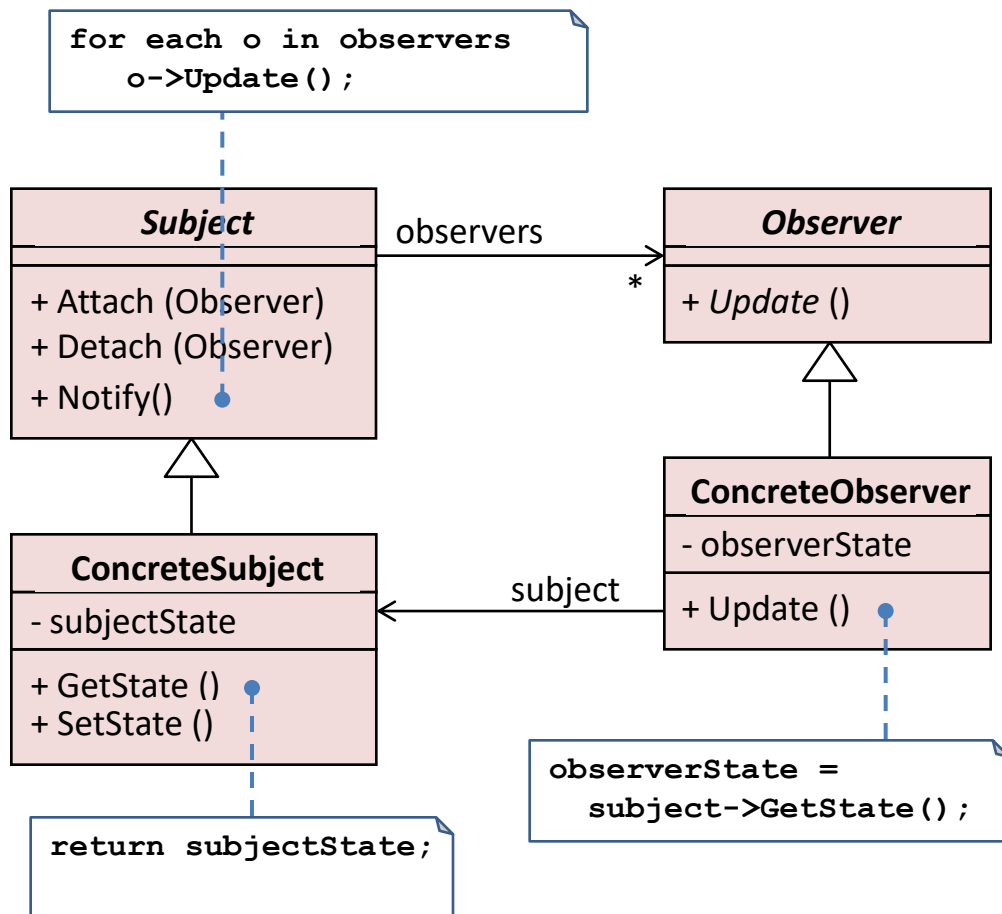
Komponente su *publisher*-i, a rukovaoci su posmatrači (*subscriber*-i) koji su registrovani na komponente od interesa

Kad se desi neki događaj, komponenta obavještava rukovaoce da se desio događaj, a rukovaoci reaguju na sebi svojstven način

# 7.1. Centralizovana kontrola toka

## Observer (Nadzornik, Posmatrač)

- Definiše zavisnosti tipa jedan:više među različitim objektima i obezbeđuje da se promjena stanja u jednom objektu automatski reflektuje u svim zavisnim objektima.



## Subject

- čuva reference prema opserverima
- obezbeđuje interfejs za dodavanje i uklanjanje opservera

## ConcreteSubject

- čuva stanje od interesa za konkretne opservere
- šalje notifikaciju opserverima kad promijeni stanje

## Observer

- definiše interfejs za ažuriranje opservera kad se subject promijeni

## ConcreteObserver

- čuva referencu na ConcreteSubject objekte
- čuva stanje koje treba da ostane konzistentno sa stanjem konkretnog subjekta
- implementira interfejs za ažuriranje objekata koji je definisan u klasi Observer

# 7.1. Centralizovana kontrola toka

## Dobre i loše strane centralizovane kontrola toka

- **Centralizovani dizajn** (jedan upravljački objekat ima kontrolu)
  - **dobre strane:**
    - jednostavne izmjene u kontrolnoj strukturi
    - pogodno za testiranje podsistema
  - **loše strane:**
    - jedan kontrolni objekat je potencijalno “usko grlo”

## Heuristika za centralizovanu/distribuiranu kontrolu

- Kontrola može da se implementira pomoću jednog ili više kontrolnih objekata (kontrolni objekti registruju spoljne događaje, vode računa o privremenim stanjima objekata, adekvatno sekvenciraju operacije, ...)
- Lokalizacija kontrole za jedan slučaj upotrebe u jedan kontrolni objekat kod čini fleksibilnijim, lakšim za pregled i održavanje...
- Ako **dijagram sekvence (ili dijagram komunikacije) izgleda “zvjezdasto”**  
→ **distribuirana kontrola**

## 7.2. Distribuirana kontrola toka

### Distribuirani (decentralizovani) sistemi

- *“Distribuirani softverski sistem predstavlja kolekciju autonomnih softverskih komponenata, koju korisnici vide kao jedinstven sistem” (Tannebaum)*
- *“Distribuirani sistem predstavlja kolekciju softverskih komponenata raspoređenih na različitim hardverskim čvorovima, koje mogu međusobno da komuniciraju i usklađuju svoj rad samo razmjenom poruka” (Coulouris et al.)*
- *“Distribuirani sistem je onaj sistem u kojem otkaz nekog računara, za kojeg i ne znate da postoji, može da uzrokuje neupotrebljivost i vašeg računara” (Lamport)*
- Svi današnji veliki softverski sistemi (*large scale*) su distribuirani.
- Obrada podataka nije ograničena na jedan hardverski čvor, nego je distribuirana na više računara.
- Krajnji korisnici imaju doživljaj jedinstvenog sistema, bez obzira na fizičku distribuciju hardverskih i softverskih komponenata.

# 7.2. Distribuirana kontrola toka

## Osnovne karakteristike distribuiranih sistema

- **Dijeljenje resursa** (hardverskih i softverskih)
- **Otvorenost /Interoperabilnost** (standardni protokoli omogućavaju komunikaciju hardverskih i softverskih komponenata različitih proizvođača, bez obzira na korištene jezike, tehnologije i alate)
- **Konkurentnost** (istovremeno izvršavanje različitih softverskih komponenata)
- **Skalabilnost** (povećavanje propusne moći dodavanjem novih hardverskih i softverskih komponenata)
- **Transparentnost** (korisnici imaju doživljaj jedinstvenog sistema, bez obzira na fizičku distribuciju hardverskih i softverskih komponenata)
- **Složenost** (distribuirani sistemi su mnogo kompleksniji nego monolitni sistemi)
- **Nema “glavne” komponente** (u opštem slučaju nema “nadređene” komponente” u sistemu i kontrola odozgo prema dolje nije moguća)

# 7.2. Distribuirana kontrola toka

## Izazovi u projektovanju distribuiranih sistema

- **Transparentnost** (korisnici imaju doživljaj jedinstvenog sistema, bez obzira na fizičku distribuciju hardverskih i softverskih komponenata)
  - **Gdje je granica transparentnosti** (do kog nivoa korisnik treba da ima osjećaj da pristupa jedinstvenom sistemu)?
  - **Idealno**, korisnici ne bi trebalo da imaju osjećaj da je sistem distribuiran, a servisi bi trebalo da su nezavisni od načine distribucije
  - **Praktično**, to je nemoguće (kontrola nije jedinstvena, postoje mrežna kašnjenja) – zato je bolje da su korisnici toga svjesni i da treba da očekuju nedostatke
  - Da bi se ostvarila transparentnost treba koristiti **apstrakciju resursa i logičko adresiranje** (a ne fizičko) – **za logičko↔fizičko mapiranje zadužen je middleware**

# 7.2. Distribuirana kontrola toka

## Izazovi u projektovanju distribuiranih sistema

### – Skalabilnost

- Sposobnost sistema da pruža servise (odgovarajućeg kvaliteta) bez obzira na:
  - povećavanje broja korisnika,
  - geografsku distribuiranost resursa (npr. nova poslovna jedinica)
- “**Scaling up**” (**vertikalno skaliranje**) – zamjena resursa resursom sa većom propusnom moći
- “**Scaling out**” (**horizontalno skaliranje**) – dodavanje novih resursa
- Horizontalno skaliranje ima bolji odnos uloženo-dobijeno, ali se zahtijeva podrška za konkurentan rad

### – Kvalitet usluge (*QoS – Quality of Service*)

- Sposobnost sistema da servise pruža:
  - pouzdano,
  - sa prihvatljivim vremenom odziva i
  - sa prihvatljivim propusnim opsegom
- Posebno kritično: **vremenski odziv i propusni opseg** (audio/video)



# 7.2. Distribuirana kontrola toka

## Izazovi u projektovanju distribuiranih sistema

### – Sigurnost

- Mnogo više sigurnosnih prijetnji nego u centralizovanim sistemima:
  - **presretanje komunikacije,**
  - **nedostupnost servisa** (*denial of service*) usljed preopterećenja čvora lažnim zahtjevima,
  - **modifikacija podataka ili servisa,**
  - neautorizovani pristup (krivotvorenje kredencijala, krađa identiteta, ...).
- Dijelovi sistema mogu da pripadaju različitim organizacijama, koje mogu da imaju različite sigurnosne politike

### – Upravljanje otkazima (*Failure Management*)

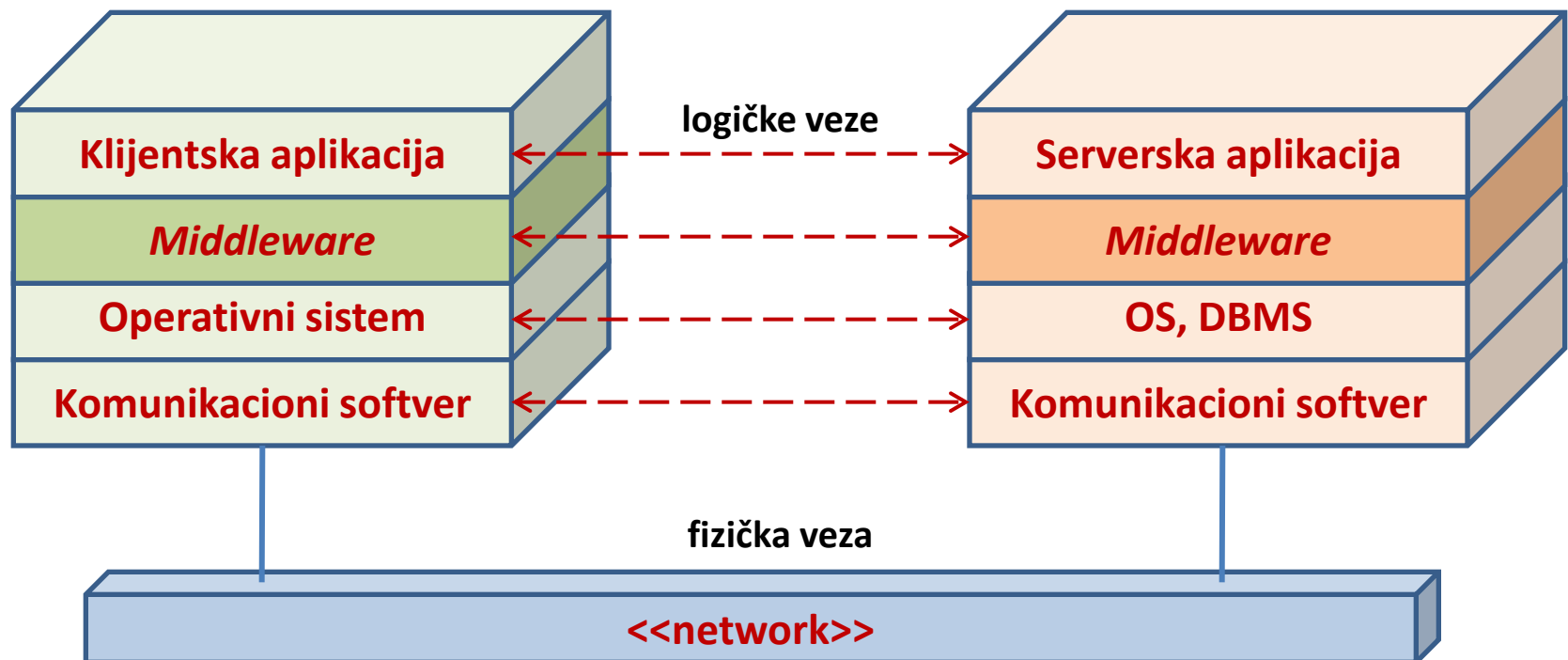
- Otkazi su neizbježni (pogotovo u distribuiranim sistemima), a sistem treba da bude što manje osjetljiv na otkaze:
  - **postojanje mehanizama za otkrivanje komponenata koje su otkazale**
  - **nastavak pružanja maksimalnog broja usluga (koji ne zavise od komponenata koje su otkazale)**
  - **automatski oporavak od otkaza, ako je moguće**

## 7.2. Distribuirana kontrola toka

### Interakcija softverskih komponenata u distribuiranim sistemima

Komunikacija u distribuiranim sistemima: **sinhrona / asinhrona**

Komunikacija se izvodi posredstvom softverskog međusloja – *middleware*



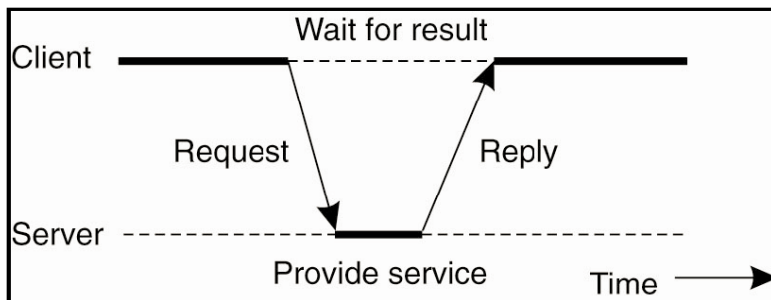
# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

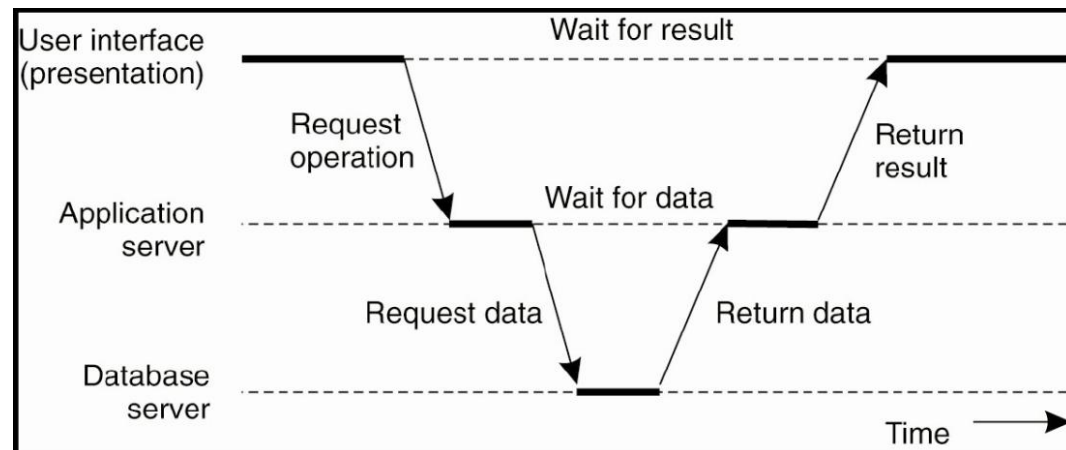
### Sinhrona interakcija

#### – Proceduralna komunikacija

- Jedna komponenta **poziva** (poznati) **servis druge komponente** (na istom ili drugom hardverskom čvoru) i **čeka odgovor** (kao da je riječ o objektima unutar iste softverske komponente)
- Tipična interakcija u  $n$ -slojnim arhitekturama
- Implementacija: **RPC (*Remote Procedure Call*)**



**Proceduralna komunikacija u 2-slojnoj arhitekturi**



**Proceduralna komunikacija u 3-slojnoj arhitekturi**

# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Sinhrona interakcija – RPC (*Remote Procedure Call*)

#### *Request–Response* protokol

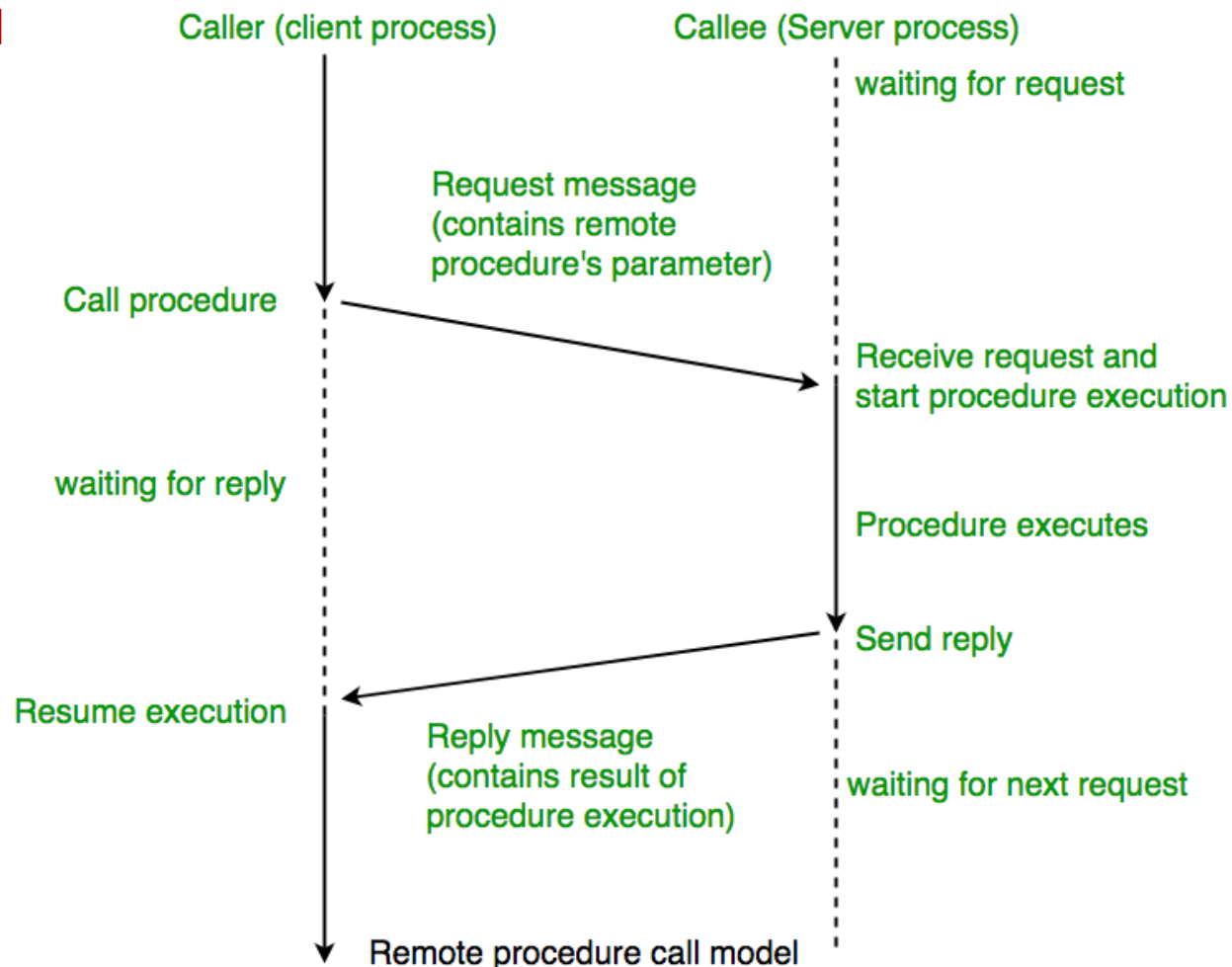
**Pozivalac** (klijentski proces)  
poziva poznati servis (serverski  
proces) na istom ili drugom  
hardverskom čvoru

Nakon što pošalje poziv  
udaljenom servisu, **pozivalac**  
**čeka odgovor** (suspendovano  
izvršavanje klijentskog procesa)

**Serverski proces**, nakon što  
završi izvršavanje, **vraća**  
**rezultat** klijentskom procesu

**Klijentski servis prima rezultat**  
**i nastavlja izvršavanje**

Primjeri: HTTPRequest, JDBC



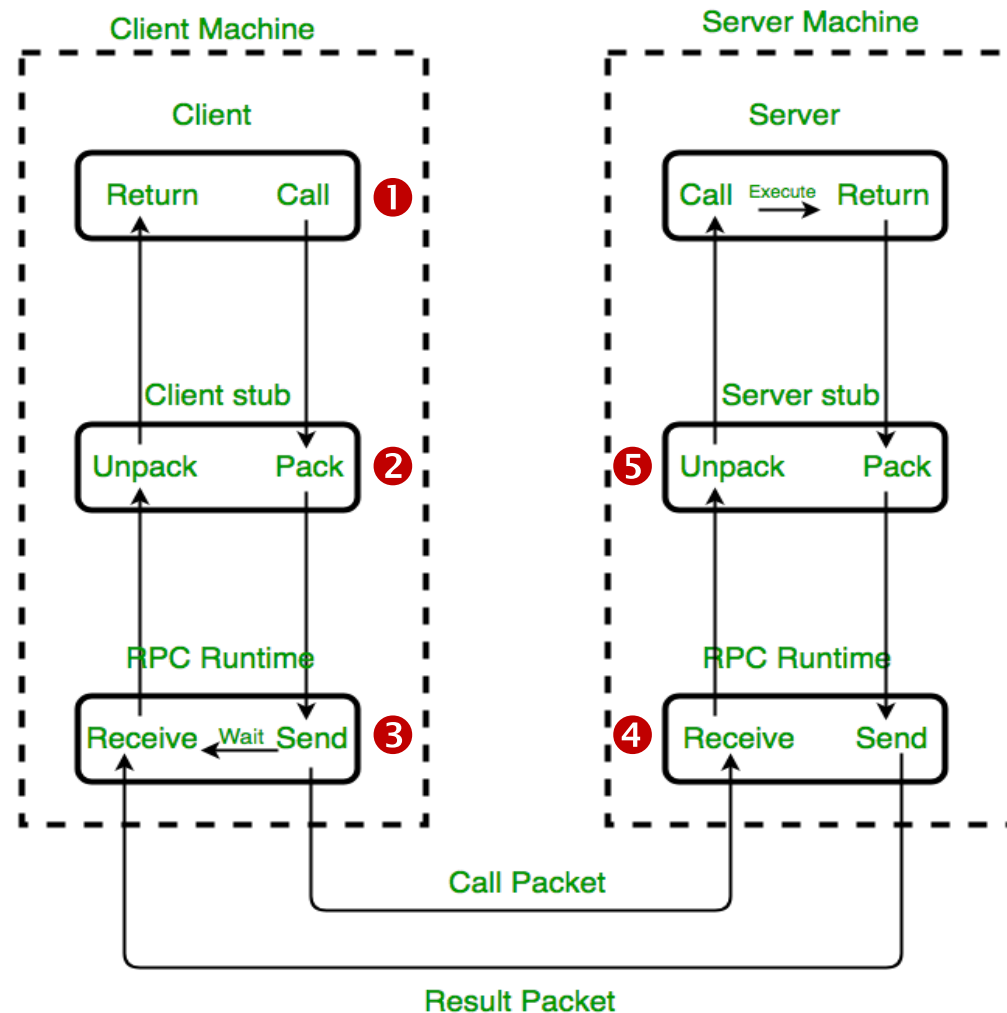
# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Sinhrona interakcija – RPC (*Remote Procedure Call*)

#### Realizacija RPC mehanizma

1. Klijent poziva operaciju klijentskog proksi objekta (*client stub*) – klijent i klijentski proksi nalaze se u istom adresnom prostoru
2. Klijentski proksi pakuje parametre u poruku (*marshalling*) i šalje je transportnom sloju (*middleware*)
3. Transportni sloj prosljeđuje poruku (udaljenom) serveru
4. Na serverskoj strani, transportni sloj prima i prosljeđuje poruku serverskom proksi objektu (*server stub*)
5. Serverski proksi raspakuje poruku (*unmarshalling*) i prosljeđuje parametre serverskom procesu



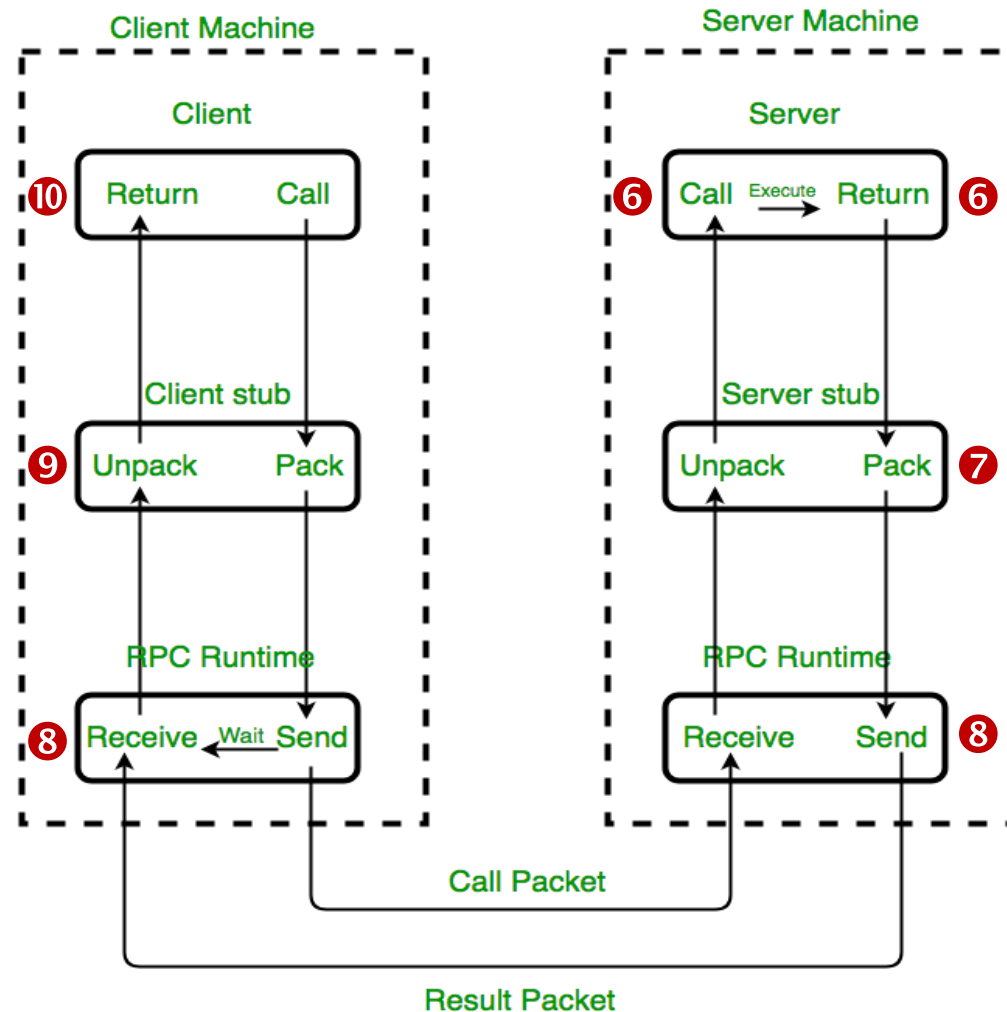
# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Sinhrona interakcija – RPC (*Remote Procedure Call*)

#### Realizacija RPC mehanizma

6. Server izvršava pozvanu proceduru i vraća rezultat serverskom proksiju kao da je riječ o lokalnom pozivu
7. Serverski proksi pakuje rezultat u poruku (*marshalling*) i šalje je transportnom sloju (*middleware*)
8. Transportni sloj šalje poruku klijentskoj mašini, koja prima poruku i prosljeđuje je klijentskom proksiju
9. Klijentski proksi raspakuje poruku, ekstrahuje rezultat i šalje ga klijentu
10. Klijent prima rezultat od klijenskog proksija, kao da je u pitanju bio lokalni poziv



# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Sinhrona interakcija – RPC (*Remote Procedure Call*)

#### RPC middleware

- Kolekcija servisa koji omogućavaju realizaciju *request-response* protokola i implementiraju RPC mehanizam
- Tokom RPC poziva, RPC komponente uspostavljaju komunikaciju preko odgovarajućeg protokola, obezbjeđuju odgovarajuće vezivanje (*binding*), prenose podatke i rješavaju komunikacione greške.

#### Proksi objekti

- Proksi objekti obezbjeđuju transparentnost u klijentskom aplikativnom kodu
- **Klijentski proksi** predstavlja interfejs između lokalnog klijenta i RPC sistema
- **Serverski proksi** predstavlja interfejs između servisa i RPC sistema

#### Vezivanje (*binding*) klijentskog i serverskog koda

- Kako klijent zna koga zove i gdje se servis nalazi?

# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Sinhrona interakcija – Java RMI

#### Java RMI (*Remote Method Invocation*)

- RMI je objektno-orijentisana verzija RPC mehanizma
- osnovne komponente:
  - **serverska strana:**
    - interfejs udaljenog objekta
    - udaljeni objekat – instanca klase koja implementira interfejs
    - serverska aplikacija koja instancira i objavljuje udaljeni objekat
    - registar imena
  - **klijentska strana**



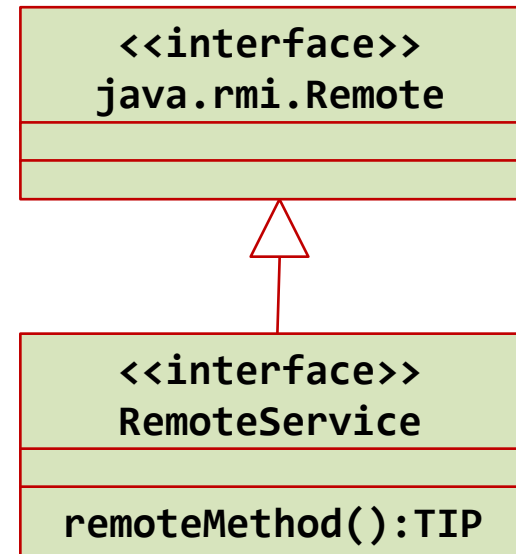
# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Java RMI – Interfejs udaljenog objekta

- proširuje `java.rmi.Remote` interfejs
- deklarirše udaljene metode
- svaka udaljena metoda mora da deklarirše dizanje izuzetka tipa `RemoteException`

```
package example.rmi;  
import java.rmi.Remote;  
import java.rmi.RemoteException;  
public interface RemoteService extends Remote  
{  
    TIP remoteMethod() throws RemoteException;  
}
```



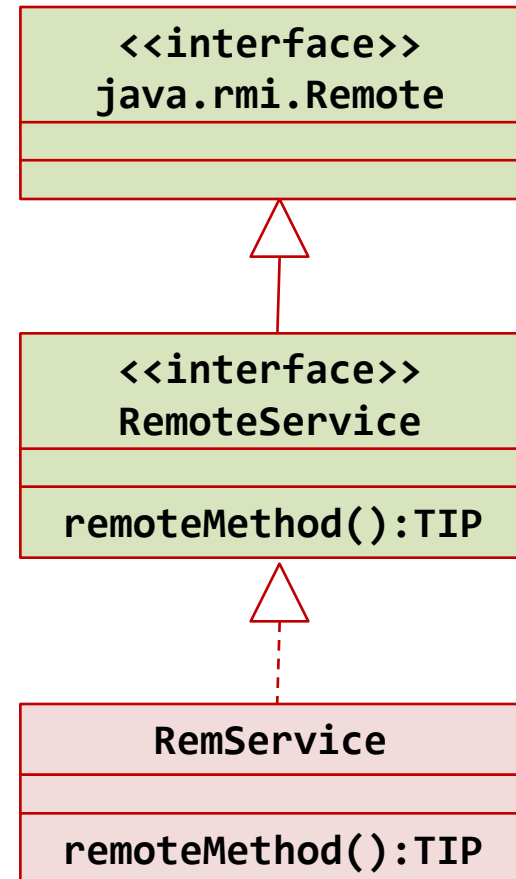
# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Java RMI – Udaljeni objekat (*Remote object*)

- udaljeni servis koji implementira udaljeni interfejs
- implementira udaljene metode
- može biti implementiran u zasebnoj klasi ili u serverskoj klasi

```
package example.rmi;  
public class RemService implements RemoteService  
{  
    public RemService() {}  
    public TIP remoteMethod()  
    { // return rezultat; }  
}
```



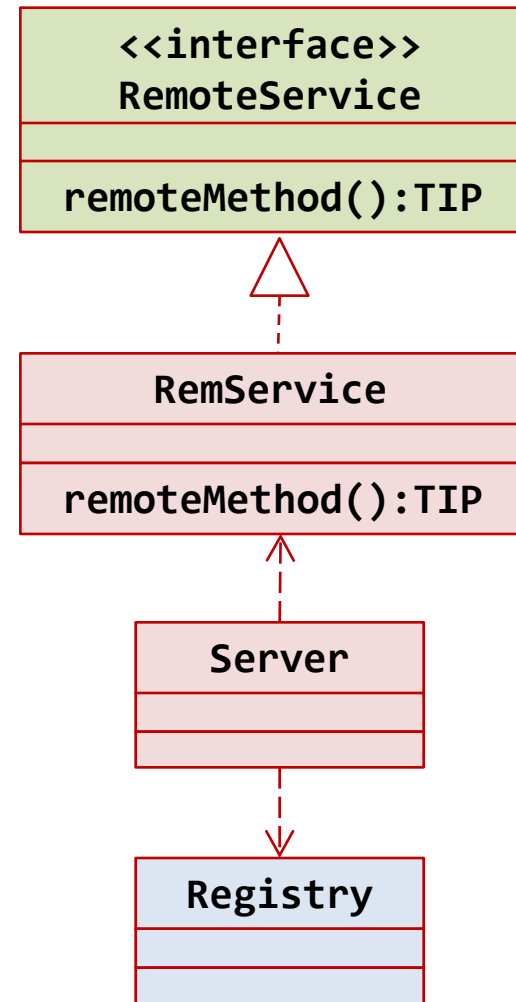
# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Java RMI – Serverska aplikacija

- u main metodi instancira jedan udaljeni objekat, eksportuje ga i u registru udaljenih objekata veže za ime u Java RMI registru
- udaljeni objekat nakon eksportovanja očekuje pozive na odgovarajućem TCP portu
- eksportovanjem se dobija proksi objekat koji se u registru veže za ime i kojeg kasnije klijent može da preuzme, kako bi pozivao metode udaljenog objekta

```
package example.rmi;  
import java.rmi.registry.Registry;  
import java.rmi.registry.LocateRegistry;  
import java.rmi.RemoteException;  
import java.rmi.server.UnicastRemoteObject;  
public class Server { // }
```



# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Java RMI – Serverska aplikacija

```
public class Server
{
    // ...
    public static void main(String args[])
    {
        try
        {
            // instanciranje i eksportovanje udaljenog objekta
            RemService rs = new RemService();
            RemoteService stub = (RemoteService)
                UnicastRemoteObject.exportObject(rs, 0);

            // objavljivanje serverskog proksija u registru
            Registry registrar = LocateRegistry.getRegistry();
            registrar.bind("RemoteService", stub);
        } catch (Exception e) { // }
    }
}
```

# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Java RMI – Klijentska aplikacija

```
import java.rmi.registry LocateRegistry;
import java.rmi.registry Registry;

public class Klijent
{
    private Klijent() {}

    public static void main(String args[])
    {
        try
        {
            // ...
            Registry registrar = LocateRegistry.getRegistry(host);
            RemoteService stub = (RemoteService) registrar.lookup("RemoteService");
            TIP rezultat = stub.remoteMethod();
            // ...
        }
        catch (Exception e) { // ... }
    }
}
```

**LocateRegistry.getRegistry(host)**

vraća registrar stub za pristup registru na hostu  
ako je host==null, gleda lokalno

Na osnovu registrar stub-a pribavlja se stub za  
udaljeni objekat

# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Asinhrona interakcija – Razmjena poruka (*message-based*)

- Jedna komponenta kreira poruku sa zahtjevom prema drugoj komponenti
- Poruka se prosljeđuje putem middleware odredišnoj komponenti
- Odredišna komponenta raspakuje poruku i izvršava odgovarajući servis i po potrebi šalje povratnu poruku sa rezultatom
- Poruke se primaju i privremeno drže u FIFO baferu, sve dok primalac nije raspoloživ
- Komponente mogu biti implementirane različitim jezicima i tehnologijama
- **Middleware je zadužen za rutiranje poruka i uspješnu komunikaciju** različitih komponenata bez obzira na njihove specifičnosti
- **Middleware obezbjeđuje lokacijsku transparentnost** – komponenta ne mora da zna na kojoj fizičkoj lokaciji se nalazi neka druga komponenta
- **Middleware obezbjeđuje neke zajedničke servise** koje mogu da koriste različite komponente (npr. upravljanje transakcijama, sigurnost ...)
- **Primjeri middleware-a: JMS, CORBA, DCOM, .NET**

# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Asinhrona interakcija – JMS (*Java Message Service*)

- **JMS** – Java API za realizaciju asinhrone komunikacije
- Dva modela asinhrone interakcije: **Point-to-Point, Publisher-Subscriber**

### Komponente JMS sistema

- **JMS Client:** Java program koji
  - šalje poruke (*sender/producer/publisher*) ili
  - prima poruke (*receiver/consumer/subscriber*).
- **JMS Provider/Middleware**
  - JMS Provider obezbjeđuje API za realizaciju asinhrone komunikacije
  - Uobičajeni nazivi: MOM, *Message Broker*, *Messaging Server*, *JMS Server*
  - Neke implementacije imaju dodatni UI interfejs za administriranje MOM sistema
- **JMS Administered Objects:** komponente za
  - uspostavljanje komunikacije Client↔MOM (**ConnectionFactory**)
  - realizaciju skladišta poruka (**Queue / Topic**)
- **JMS Message:** objekat koji sadrži podatke koji se razmjenjuju između klijenata

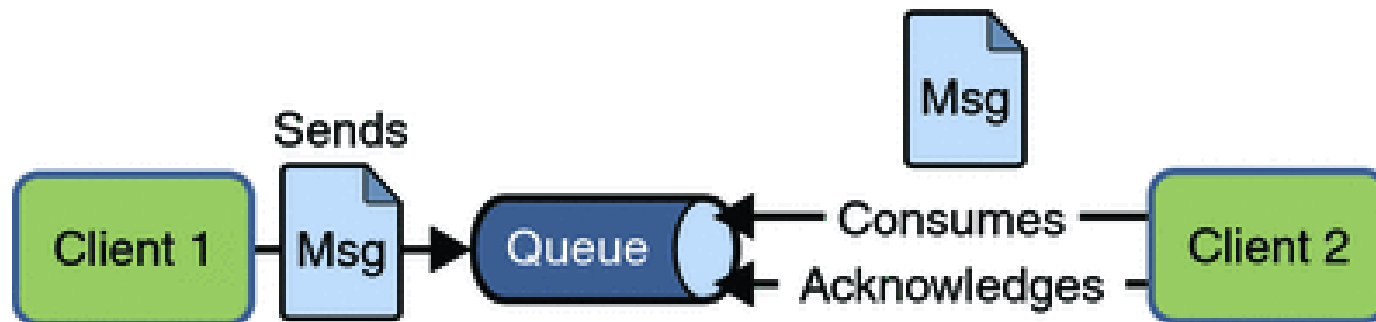
# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Asinhrona interakcija – JMS (*Java Message Service*)

#### Point-to-Point model (P2P)

- Jedna poruka dostavlja se samo jednom primaocu (*unicasting*)
- **Queue** je message-oriented middleware (MOM) zadužen za prijem i čuvanje poruka sve dok primalac nije raspoloživ
- Ne postoji vremenska zavisnost između pošiljaoca i primaoca (poruka će čekati u redu sve dok primalac ne bude raspoloživ)





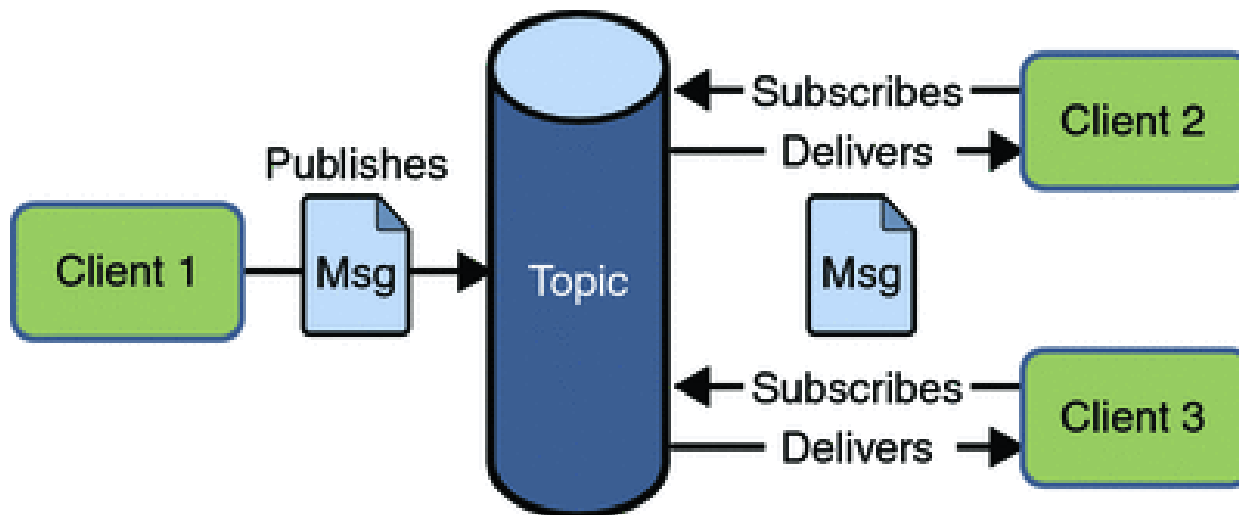
# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Asinhrona interakcija – JMS (*Java Message Service*)

#### Publisher-Subscriber model (Pub-Sub)

- Jedna poruka dostavlja se većem broju primaoca (*broadcasting*)
- **Topic** je MOM middleware zadužen za prijem i distribuciju poruka
- Poruke se distribuiraju primaocima koji su zainteresovani (*subscribers*) za prijem poruke
- Može biti više izvornih objekata (pošiljaoca) koji šalju poruke u MOM
- Postoje vremenske zavisnosti između pošiljaoca i primalaca (MOM ne mora da drži poruke, ako nema prijavljenih primalaca)



# 7.2. Distribuirana kontrola toka

## Modeli interakcije komponenata u distribuiranim sistemima

### Asinhrona interakcija – JMS (Java Message Service)

#### JMS programski model

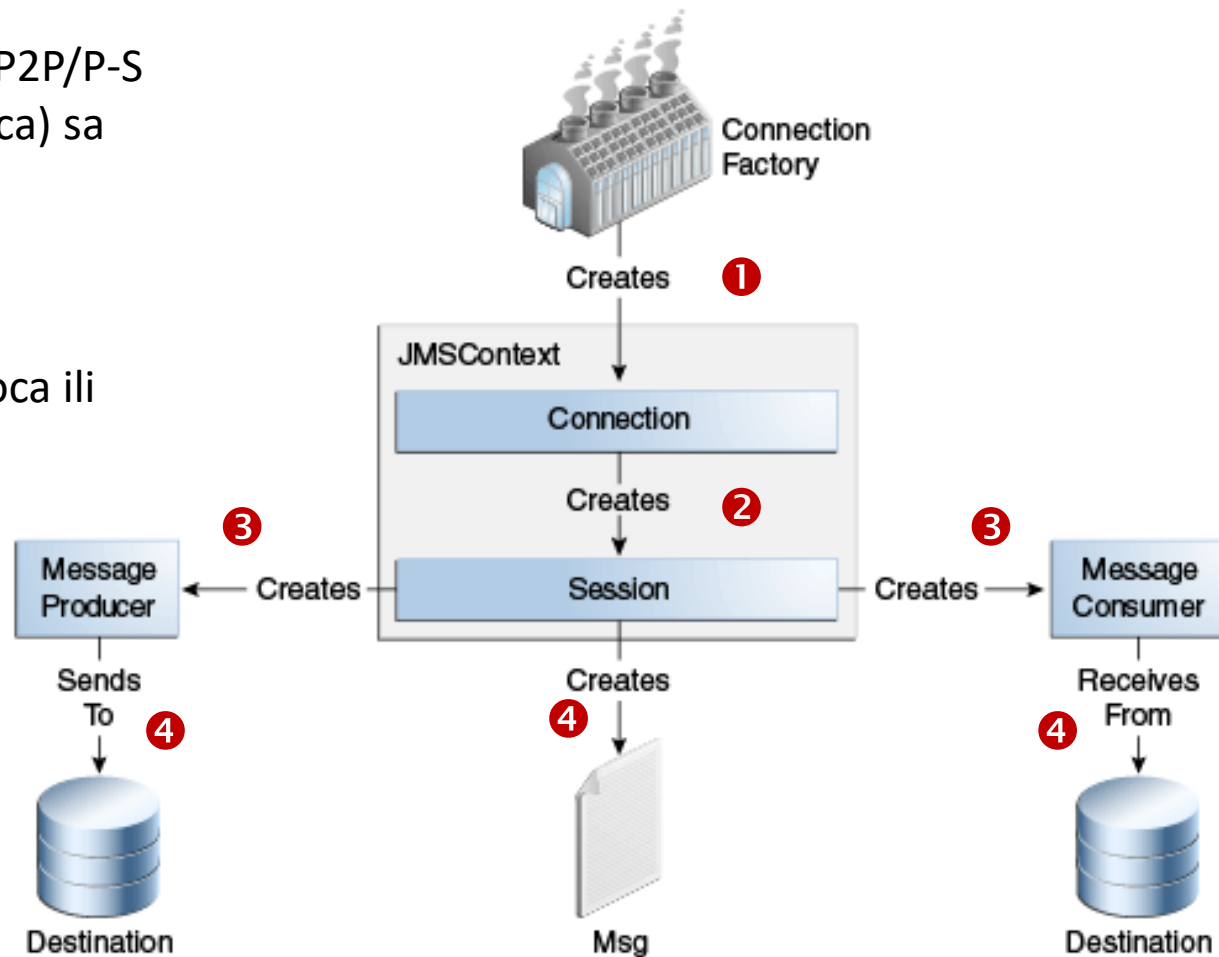
- ❶ ConnectionFactory kreira P2P/P-S konekciju (pošiljaoca/primaoca) sa JMS serverom
- ❷ Kreiranje P2P/P-S sesije (pošiljaoca/primaoca)
- ❸ Instanciranje P2P/P-S pošiljaoca ili primaoca

====

- ❹ Kreiranje poruke i slanje u skladište (pošiljalac)

====

- ❹ Čitanje poruke iz skladišta i slanje primaocu



# 7.2. Distribuirana kontrola toka

## Asinhrona interakcija – JMS (Java Message Service)

### Primjer PTP asinhrone komunikacije - pošiljalac

```
import javax.naming.*;
import javax.jms.*;
public class Sender {
    public static void main(String[] args) {
        try
        {
            //Create and start connection
            InitialContext c=new InitialContext();
            QueueConnectionFactory f=(QueueConnectionFactory)c.lookup("myQCFactory");
            QueueConnection con=f.createQueueConnection();
            con.start();
            // create queue session
            QueueSession ses=con.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
            // get the Queue object
            Queue t=(Queue)c.lookup("myQueue");
            // create QueueSender object
            QueueSender sender=ses.createSender(t);
            // create TextMessage object
            TextMessage msg=ses.createTextMessage();
            // send message
            sender.send(msg);
            // connection close
            con.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

# 7.2. Distribuirana kontrola toka

## Asinhrona interakcija – JMS (Java Message Service)

### Primjer PTP asinhrone komunikacije - **primalac**

```
import javax.naming.*;
import javax.jms.*;

public class Receiver {
    public static void main(String[] args) {
        try
        {
            //Create and start connection
            InitialContext c=new InitialContext();
            QueueConnectionFactory f=(QueueConnectionFactory)c.lookup("myQCFactory");
            QueueConnection con=f.createQueueConnection();
            con.start();
            // create queue session
            QueueSession ses=con.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
            // get the Queue object
            Queue t=(Queue)c.lookup("myQueue");
            // create QueueReceiver
            QueueReceiver r=ses.createReceiver(t);
            // create listener object
            MyListener ls=new MyListener();
            // register the listener with receiver
            r.setMessageListener(ls);
            // ...
            con.close();
        }catch(Exception e){ // ... }
    }
}
```

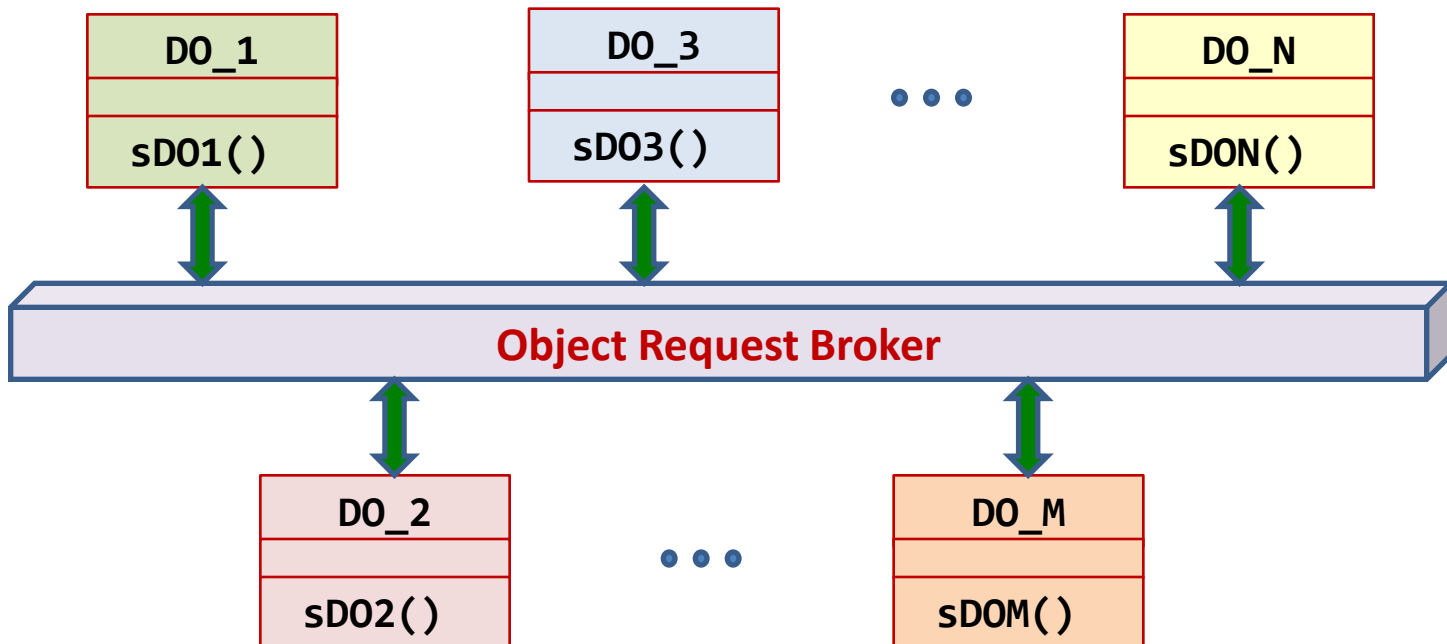
```
import javax.jms.*;

public class MyListener
    implements MessageListener
{
    public void onMessage(Message m)
    {
        try{
            TextMessage mm=(TextMessage)m;
            System.out.println(mm.getText());
        }
        catch(JMSEException e) { // ... }
    }
}
```

## 7.2. Distribuirana kontrola toka

### Arhitektura distribuiranih objekata (*Distribute Objects Architecture*)

- Nema eksplicitne podjele na klijente i servere
- Svaki distribuirani objekat/komponenta pruža servise drugim objektima/komponentama
- Komunikacija između distribuiranih objekata/komponenta odvija se posredstvom middleware-a koji se naziva *Object Request Broker – ORB*



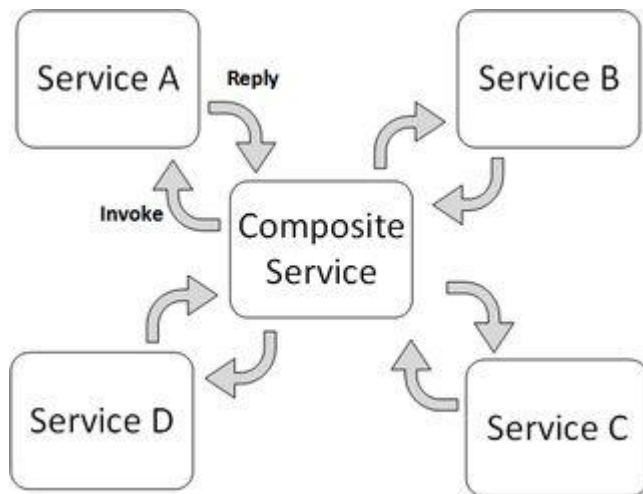
# 7.2. Distribuirana kontrola toka

## Servisno-orijentisana arhitektura (SOA)

- Distribuirana softverska arhitektura koji čine autonomne softverske komponente (servisi) koje mogu biti raspoređene na različitim hardverskim čvorovima
- Funkcionalnost sistema ostvaruje se odgovarajućim kombinovanjem funkcionalnosti pojedinih servisa: **orkestracija** / **koreografija**

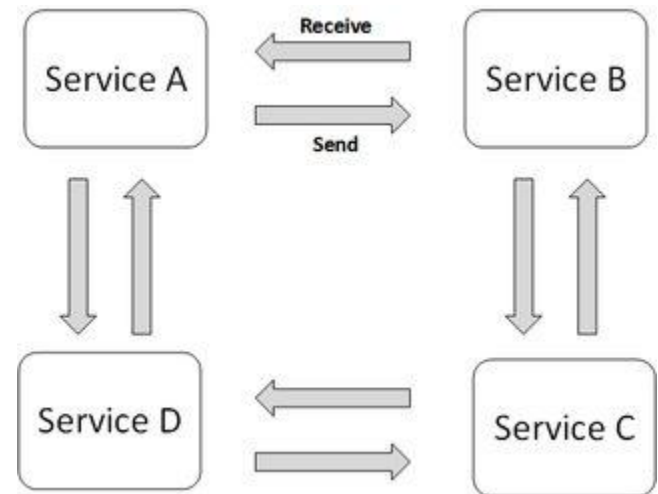
### Orkestracija servisa

- Funkcionalnost se ostvaruje centralizovanim upravljanjem (**orkestratorski servis**)
- **orkestrator** ostvaruje željenu funkcionalnost odgovarajućim kombinovanjem aktivnosti pojedinih servisa



### Koreografija servisa

- Funkcionalnost se ostvaruje decentralizovanim upravljanjem
- Različiti servisi imaju mogućnost da prime poziv, realizuju dio funkcionalnosti i pozovu drugi servis

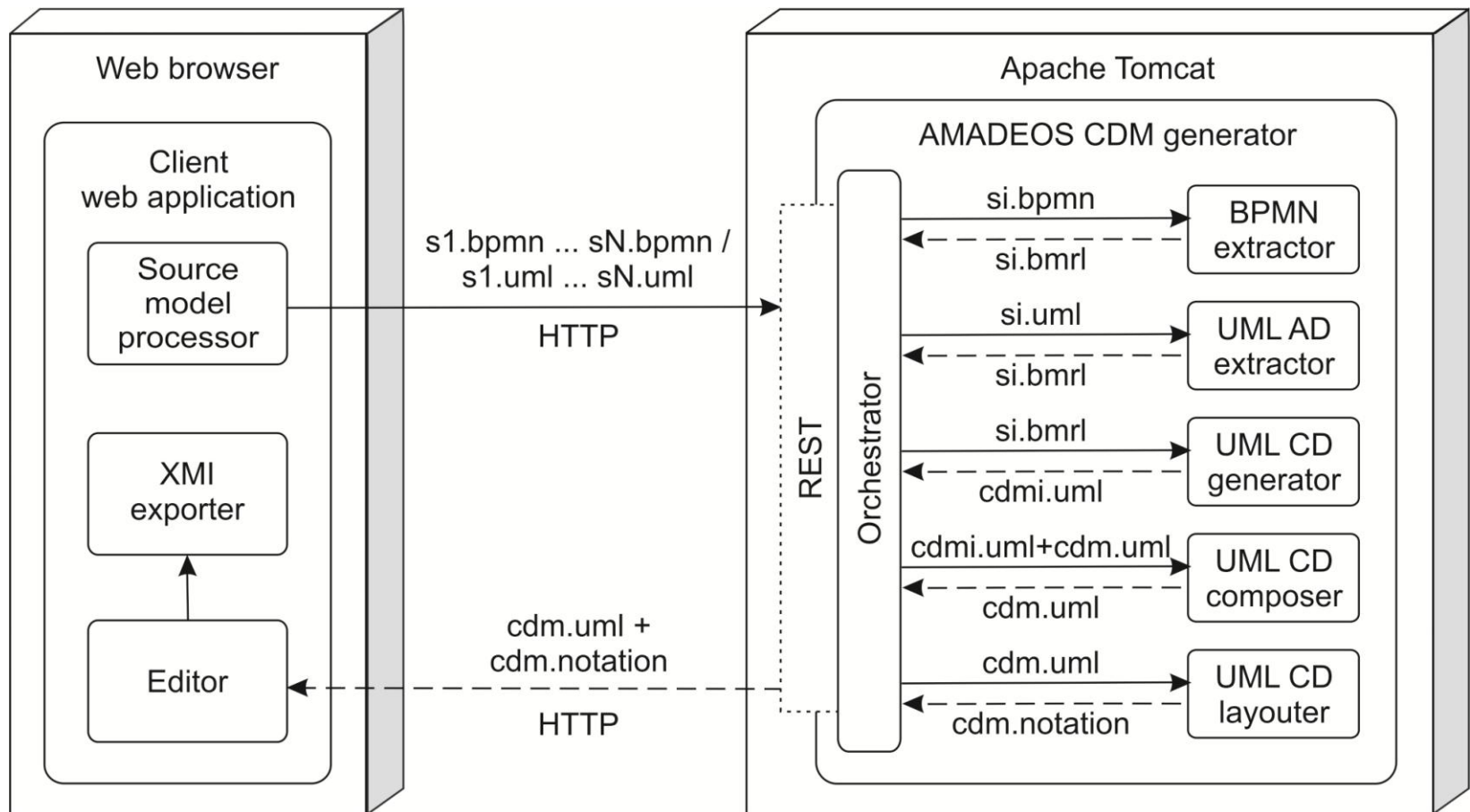


## 7.2. Distribuirana kontrola toka

### Primjer servisno-orijentisane arhitekture – AMADEOS

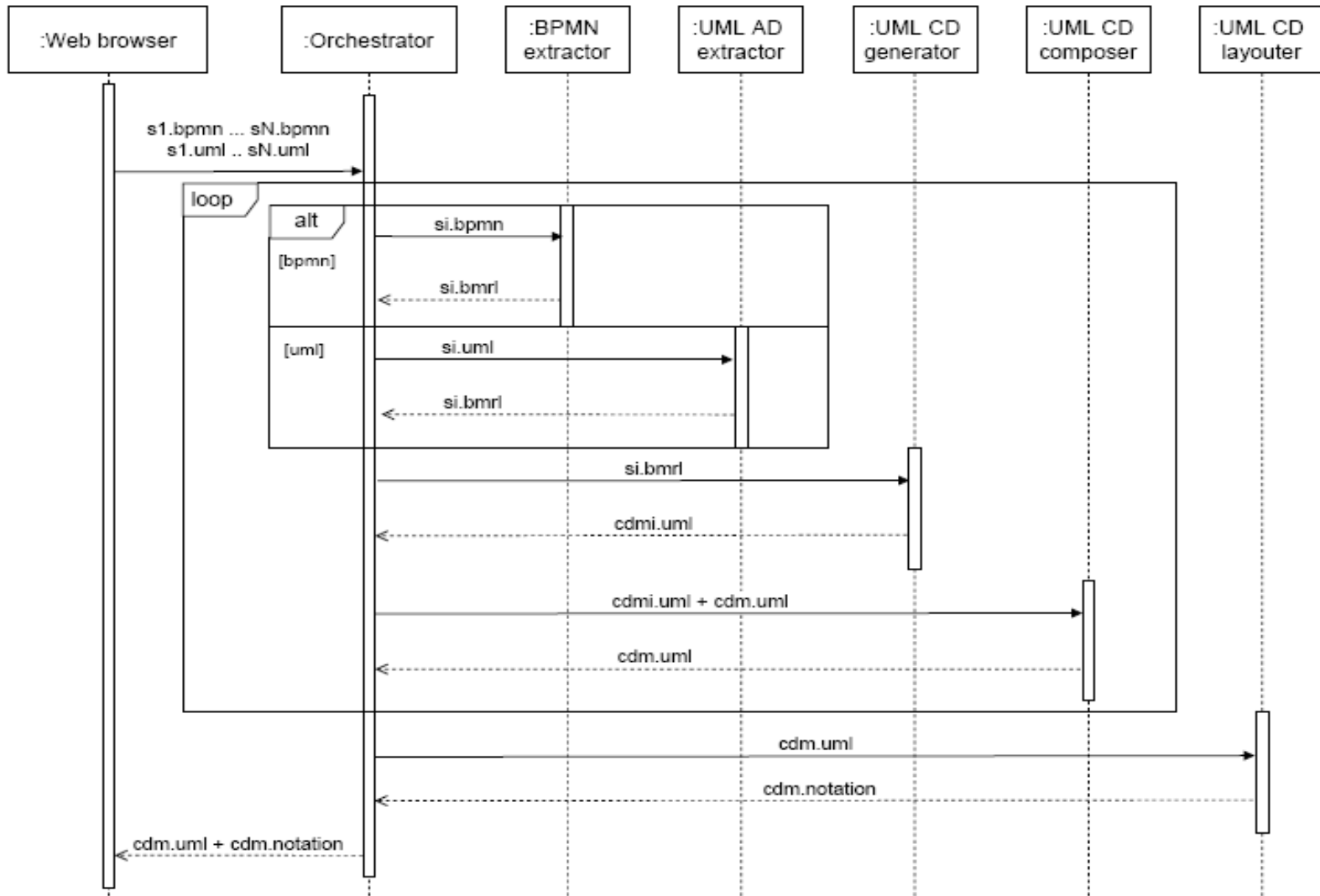
Online sistem za automatsko modelom-vođeno projektovanje baza podataka

<http://m-lab.etf.unibl.org:8080/amadeos>



# 7.2. Distribuirana kontrola toka

## Primjer servisno-orijentisane arhitekture – AMADEOS

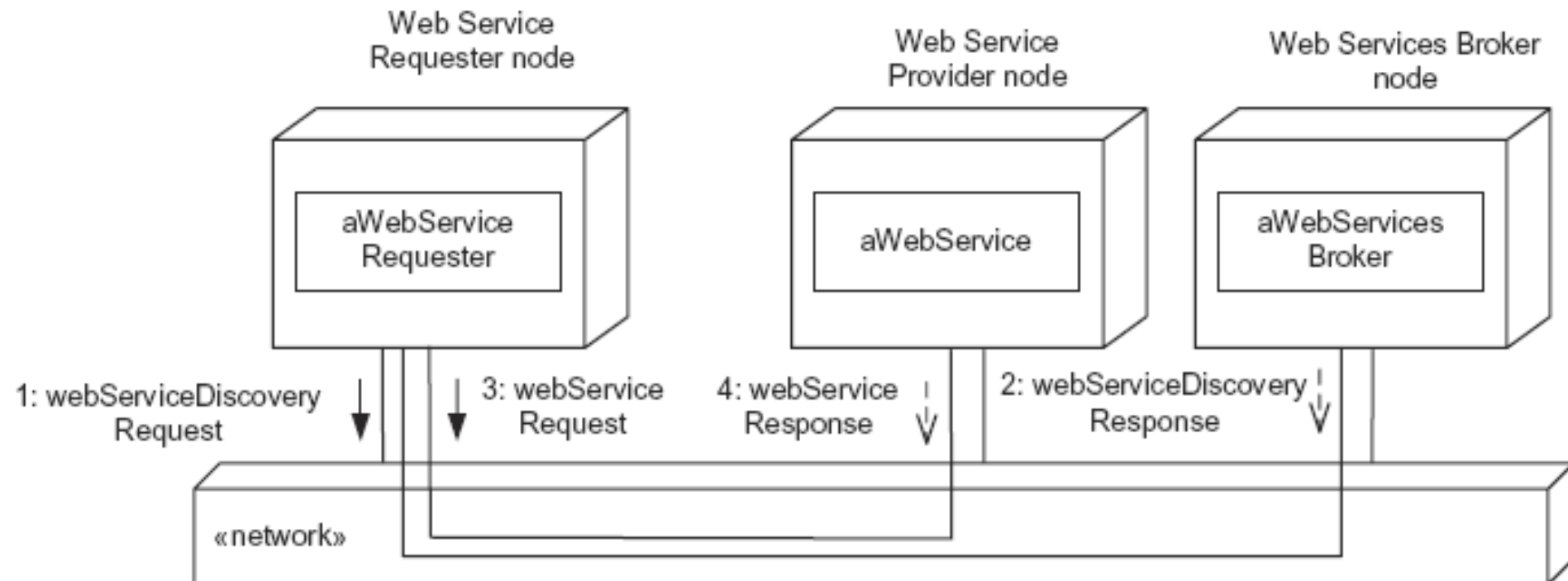




## 7.2. Distribuirana kontrola toka

### Web servisi

- **web servisi koriste web za komunikaciju i omogućavaju realizaciju SOA aplikacija**
- **iz softverske perspektive:** web servisi su API koji omogućavaju komunikaciju između različitih komponenata na standardizovan način (HTTP)
- **iz aplikativne perspektive:** web servisi su funkcionalne cjeline koje obezbjeđuju neku funkcionalnost drugim aplikacijama
- broker je posrednik između servisa (broker održava registar servisa i zna sve o servisima)

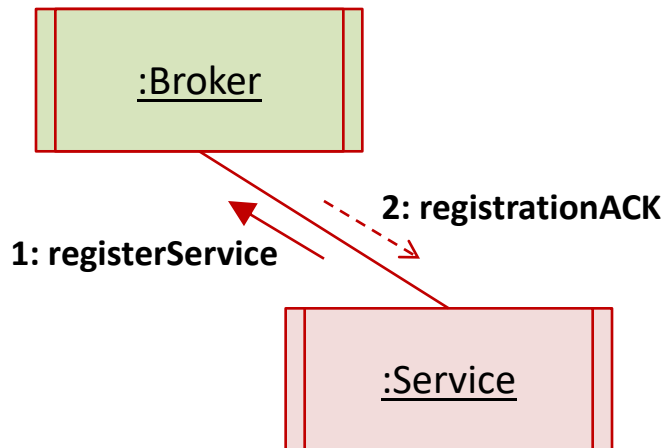


# 7.2. Distribuirana kontrola toka

## Brokerski komunikacioni obrasci

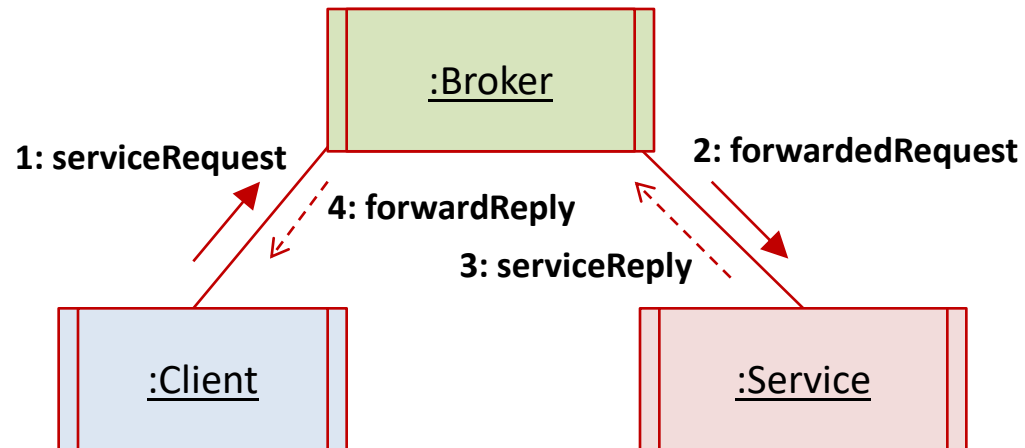
### Registracija servisa kod brokera

- broker je posrednik između servisa
- broker održava registar servisa i zna sve o servisima (gdje se nalaze, kako se pozivaju, ...)
- servisi se registruju kod brokera



### *“Broker forwarding”* obrazac

- klijent upućuje zahtjev brokeru
- broker prosljeđuje zahtjev odgovarajućem servisu
- servis obrađuje zahtjev i vraća rezultat
- broker prosljeđuje rezultat klijentu



# 7.2. Distribuirana kontrola toka

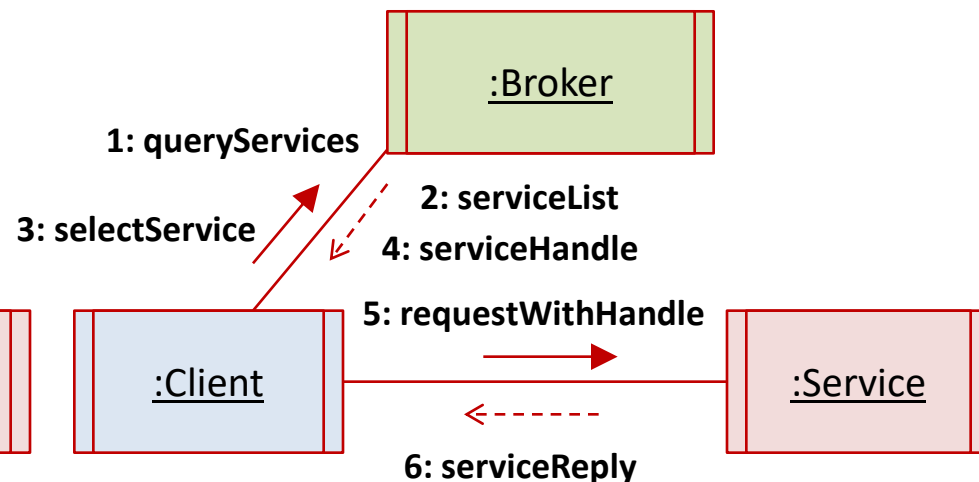
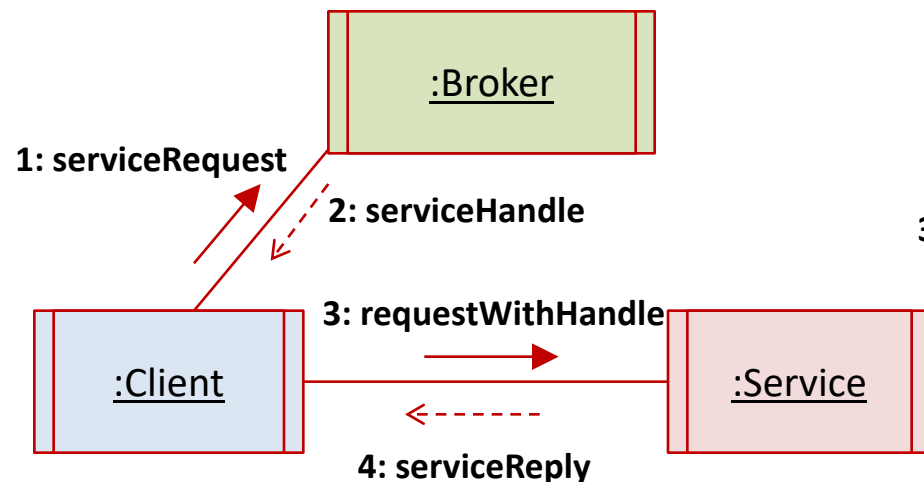
## Brokerski komunikacioni obrasci

### “Broker handle” (white pages)

- klijent upućuje zahtjev brokeru
- broker klijentu vraća *handle*
- klijent na osnovu primljenog *handle*-a poziva odgovarajući servis
- servis obrađuje zahtjev i vraća rezultat

### “Service Discovery” (yellow pages)

- klijent upućuje zahtjev brokeru za listu pogodnih servisa
- broker vraća listu servisa
- klijent traži *handle* za izabrani servis
- broker vraća *handle* za izabrani servis
- klijent na osnovu primljenog *handle*-a poziva odgovarajući servis
- servis obrađuje zahtjev i vraća rezultat



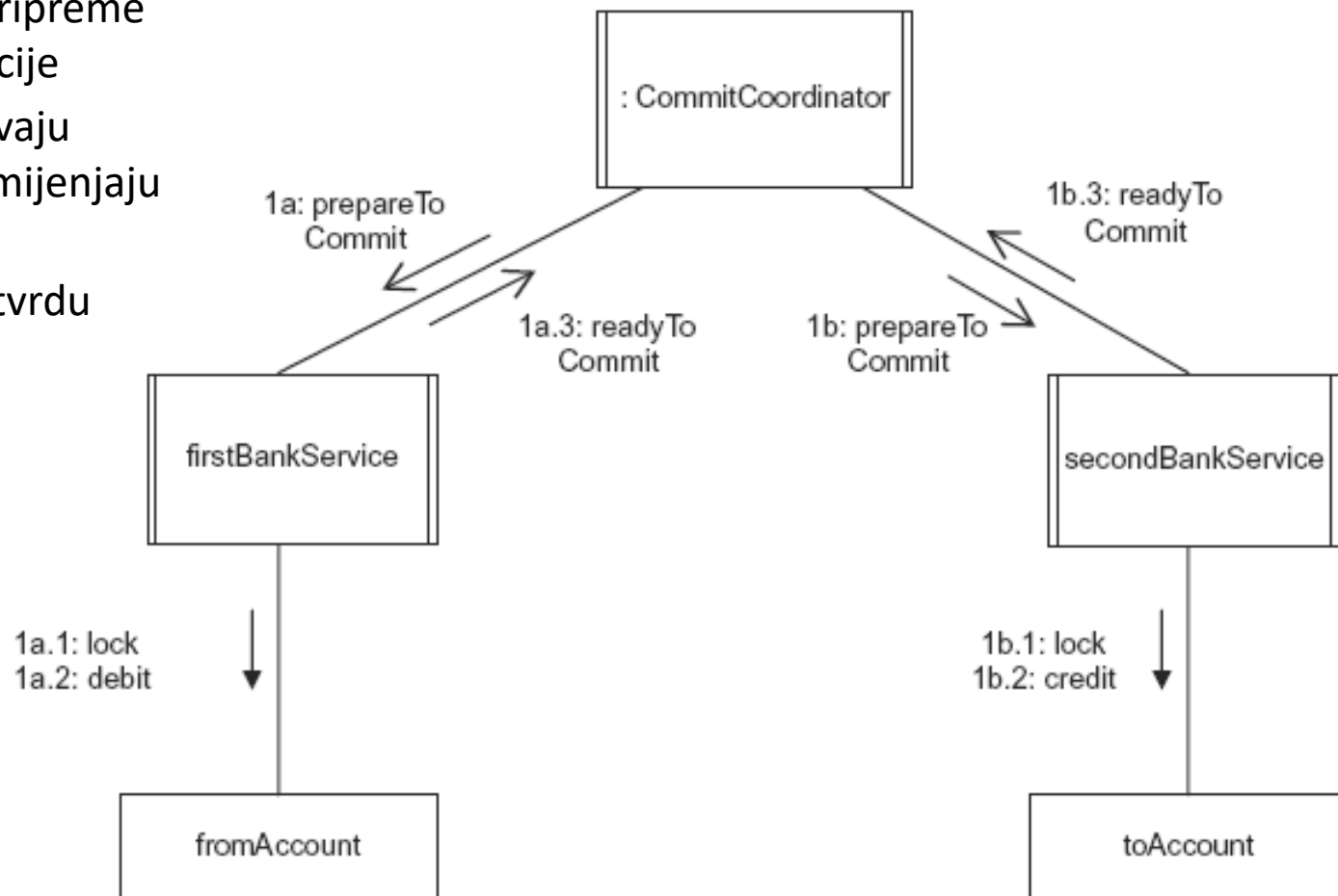
# 7.2. Distribuirana kontrola toka

## Transakcioni protokoli – “Two phase commit”

### 1. faza: priprema

- Koordinator traži od uključenih komponenata da se pripreme za izvršavanje transakcije
- Komponente zaključavaju objekte od interesa i mijenjaju stanje
- Komponente šalju potvrdu koordinatoru

Transakcija je operacija koja mora da se izvrši u cjelosti.



# 7.2. Distribuirana kontrola toka

## Transakcioni protokoli – “Two phase commit”

### 2. faza: potvrda

- Koordinator traži od uključenih komponenata da završe i potvrde transakciju
- Komponente završavaju transakciju i otključavaju objekte od interesa
- Komponente šalju potvrdu koordinatoru da je transakcija završena

Ako transakcija nije u mogućnost da se izvrši u cjelosti, parcijalni efekti moraju biti poništeni (*roll back*)

