

Oblikovanje programske potpore

2012./2013. grupa P01

Objektno usmjerena arhitektura

Prof.dr.sc. Vlado Sruk



Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva
Zavod za elektroniku, mikroel., računalne i inteligentne sustave



Tema

- Raspodijeljeni sustavi
- Arhitektura klijent – poslužitelj
- Ponovna uporaba programskih komponenti
- Posrednička arhitektura
- Uslužno usmjerena arhitektura – SOA
- Arhitektura sustava zasnovanih na komponentama

Literatura

- Timothy C. Lethbridge, Robert Laganière: ***Object-Oriented Software Engineering: Practical Software Development using UML and Java***, McGraw Hill, 2001
 - <http://www.lloseng.com>
- Sommerville, I., ***Software engineering***, 8th ed, Addison Wesley, 2007.

Pripremio i prilagodio: Nikola Bogunović, Vlado Sruk

Ovaj dokument namijenjen je isključivo za osobnu upotrebu studentima Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu.

U pripremi materijala osim literature upotrijebljeni su i drugi izvori, te zahvalujem autorima.



Raspodijeljeni sustavi



- engl. *Distributed Systems*
- Značajke raspodijeljenih sustava:
 - Obradu podataka i izračunavanja obavljaju **odvojeni programi**.
 - Uobičajeno je da su ti odvojeni programi na odvojenom sklopolju (računalima, čvorovima, stanicama).
 - Odvojeni programi **međusobno komuniciraju** računalnom mrežom.
- Primjer raspodijeljenog sustava:
- **Klijent - poslužitelj:**
 - **Poslužitelj (engl. server)**
 - Program koji dostavlja uslugu drugim programima koji su spojeni na njega preko komunikacijskog kanala.
 - **Klijent (engl. client):**
 - Program koji pristupa poslužitelju (ili više njih) tražeći uslugu.
 - Poslužitelju mogu pristupiti mnogi klijenti simultano.

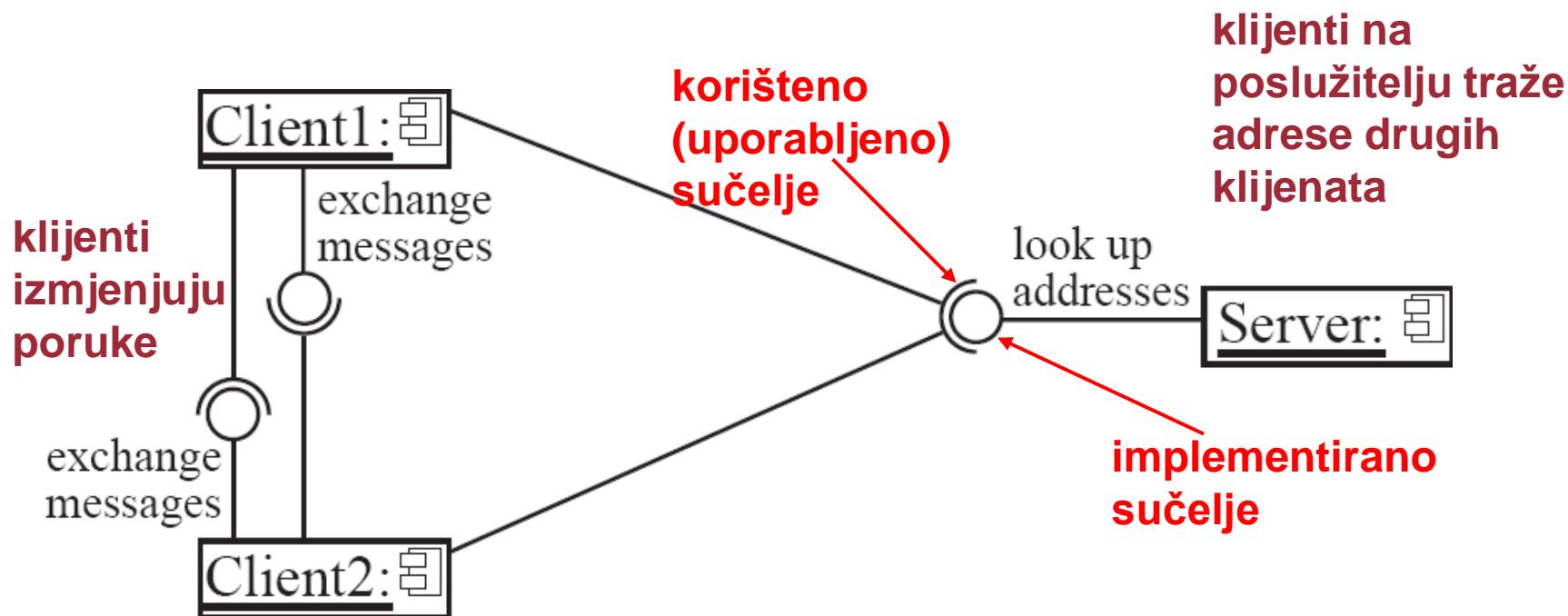
Primjeri:

- Sustavi ravnopravnih sudionika (*engl. Peer-to-Peer*)
 - svaki čvor u sustavi ima jednake mogućnosti i odgovornosti (čvor je istovremeno poslužitelj i klijent).
 - snaga obrade podataka i izračunavanja u P2P sustavu ovisi o pojedinim krajnjim čvorovima, a ne o nekom skupnom radu čvorova.
- Srodne socijalne mreže (*engl. affinity communities*)
 - jedan korisnik se povezuje s drugim korisnikom u cilju razmjene informacija (MP3, video, slike itd.).
- Kolaborativno izračunavanje (*engl. collaborative computing*)
 - neiskorišteni resursi (CPU vrijeme, prostor na disku) mnogih računala u mreži kombiniraju se u izvođenju zajedničkog zadatka (GRID computing, SETI@home, ...).
- Slanje poruka u stvarnom vremenu (*engl. Instant Messaging*)
 - izmjena tekstovnih poruka između korisnika u stvarnom vremenu.
- Upravljanje automobilom

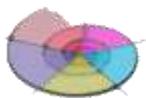
- Kako se alociraju i pokreću funkcije poslužitelja?
- Kako se definiraju i šalju parametri između klijenta i poslužitelja?
- Kako se rukuje neuspjesima (pogreškama) u komunikaciji?
- Kako se postavlja i rukuje sa sigurnošću?
- Kako klijent pronađe poslužitelja?
- Koje strukture podataka koristiti i kako rukovati s njima?
- Koja su ograničenja u paralelnom radu dijelova raspodijeljenog sustava?
- Kako se uopće skupina komponenata usuglašava oko zajedničkih pitanja?

Primjer:

- Primjer raspodijeljenog sustava u koji omogućava izravnu komunikaciju klijentima.



ARHITEKTURA KLIJENT – POSLUŽITELJ

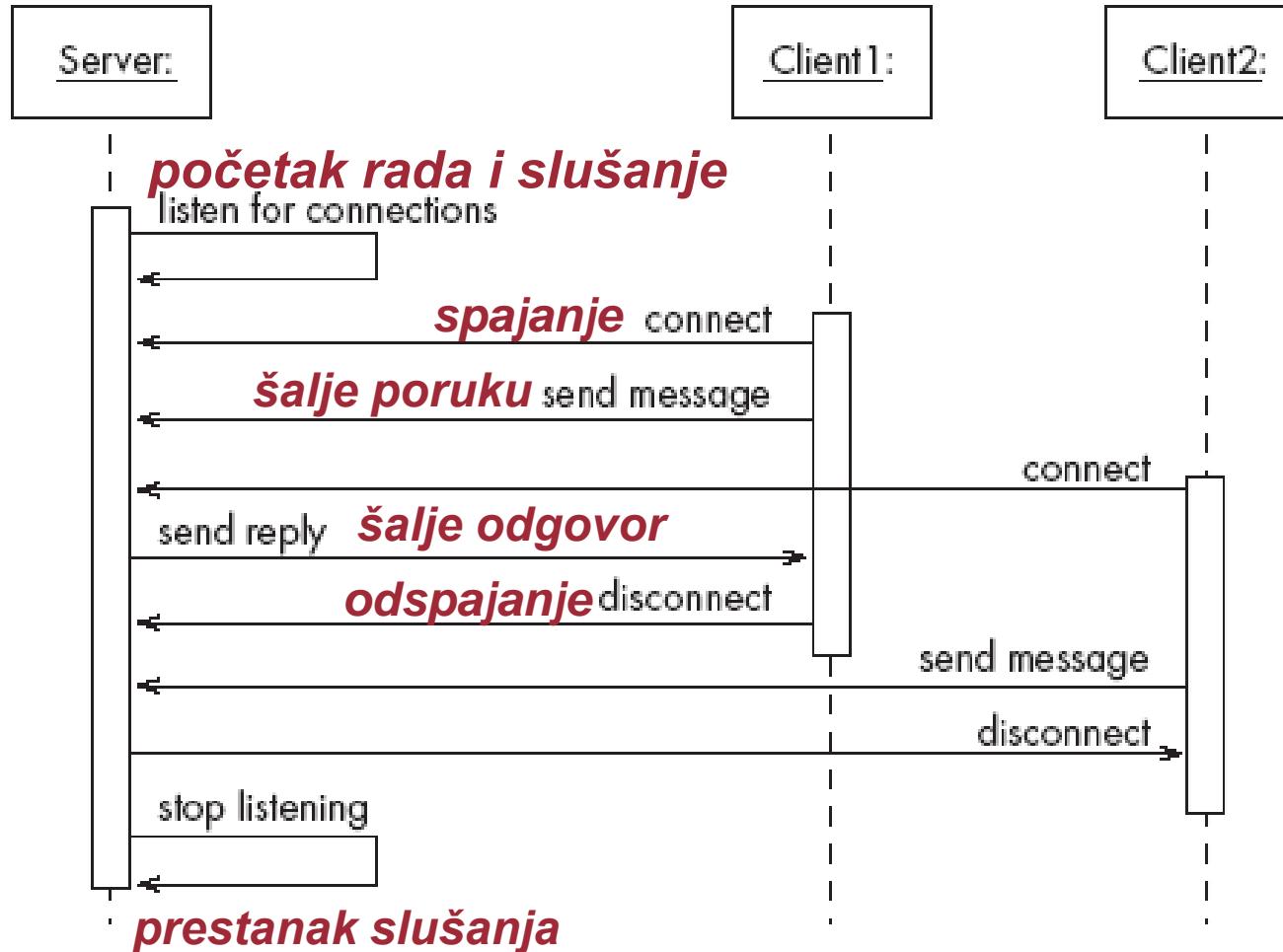


Arhitektura klijent – poslužitelj



- Sekvenca aktivnosti:
 1. Poslužitelj započinje s radom.
 2. Poslužitelj čeka na dolazak klijentskog zahtjeva (poslužitelj sluša).
 3. Klijenti započinju s radom i obavljaju razne operacije,
 - neke operacije traže zahtjeve i odgovore s poslužitelja.
 4. Kada klijent pokuša spajanje na poslužitelja, poslužitelj mu to omogući (ako želi).
 5. Poslužitelj čeka na poruke koje dolaze od spojenih klijenata.
 6. Kada pristigne poruka nekog klijenta poslužitelj poduzima akcije kao odziv na tu poruku.
 7. Klijenti i poslužitelj nastavljaju s navedenim aktivnostima sve do odspajanja ili prestanka rada.

Poslužitelj u komunikaciji s dva klijenta





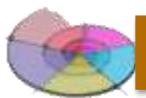
Alternative klijent – poslužitelj arhitekture

- Implementacija jednog programa na jednom računalu koji obavlja sve.
- Računala nisu spojena u mrežu već svako računalo obavlja svoj posao odvojeno.
- Ostvariti neki drugi mehanizam (osim klijent-poslužitelj) kako bi računala u mreži razmjenjivala informacije.
 - npr. jedan program upisuje u bazu podataka a drugi čita.

Prednosti klijent – poslužitelj arhitekture

- Posao se može ***raspodijeliti*** na više računala (strojeva).
- Klijenti ***udaljeno*** pristupaju funkcionalnostima poslužitelja.
- Klijent i poslužitelj mogu se ***oblikovati odvojeno***.
- Oba entiteta mogu biti ***jednostavnija***.
- Svi podaci mogu se držati ***na jednom mjestu*** (na poslužitelju).
- Obrnuto, podaci se mogu ***distribuirati*** na više udaljenih klijenata i poslužitelja.
- Poslužitelju može ***istodobno*** pristupiti više klijenata.
- Klijenti mogu ući u ***natjecanje*** za uslugu poslužitelja (a i obrnuto).

- Posebice značajni obzirom na primjenu i održavanje:
- Sigurnost
 - sigurnost je veliki problem bez savršenog rješenja. Potrebno uporabiti enkripciju, zaštitne zidove (*engl. firewalls*) i sl.
- Potreba za adaptivnim održavanjem
 - budući da se programska potpora za klijente i poslužitelja oblikuje odvojeno potrebno je osigurati da sva programska potpora bude
 - kompatibilna prema unatrag (*engl. backwards*) i
 - prema unaprijed (*engl. forwards*),
 - te kompatibilna s drugim verzijama klijenata i poslužitelja.



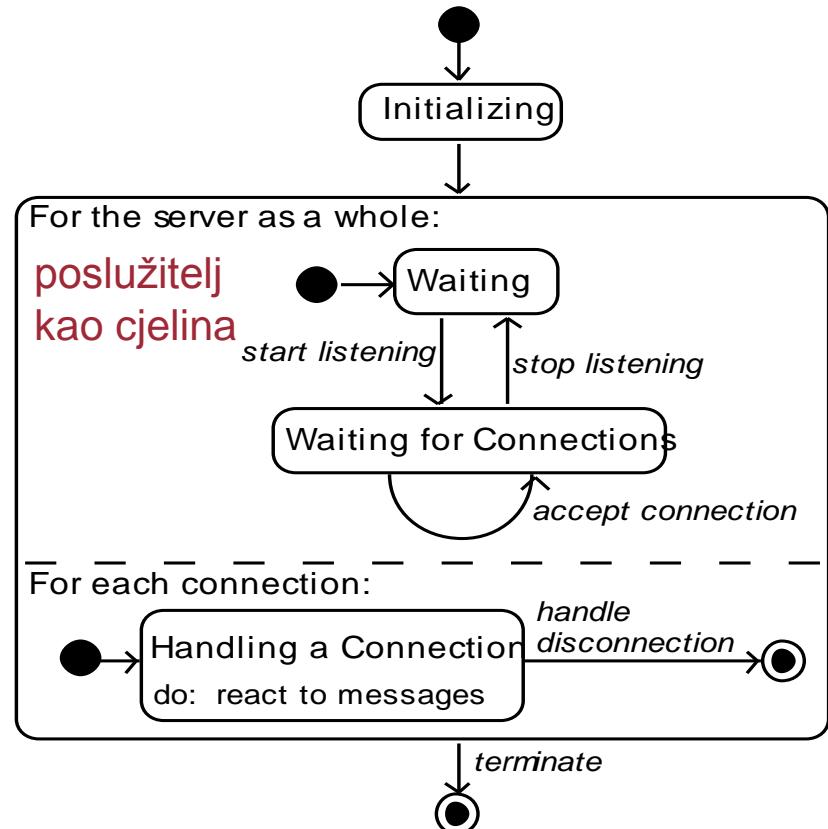
Primjeri klijent – poslužitelj arhitek.



- The World Wide Web
 - E-mail
 - Network File System -NFS
 - Transaction Processing System
 - Remote Display System
 - Communication System
 - Database System
 -
-
- Poslužitelji:
 - Application Servers
 - Audio/Video Servers
 - Chat Servers
 - Fax Servers
 - FTP Servers
 - IRC Servers
 - Mail Servers
 - News Servers
 - Proxy Servers
 - Web Servers
 - Z39.50 Servers

Funkcionalnosti poslužitelja

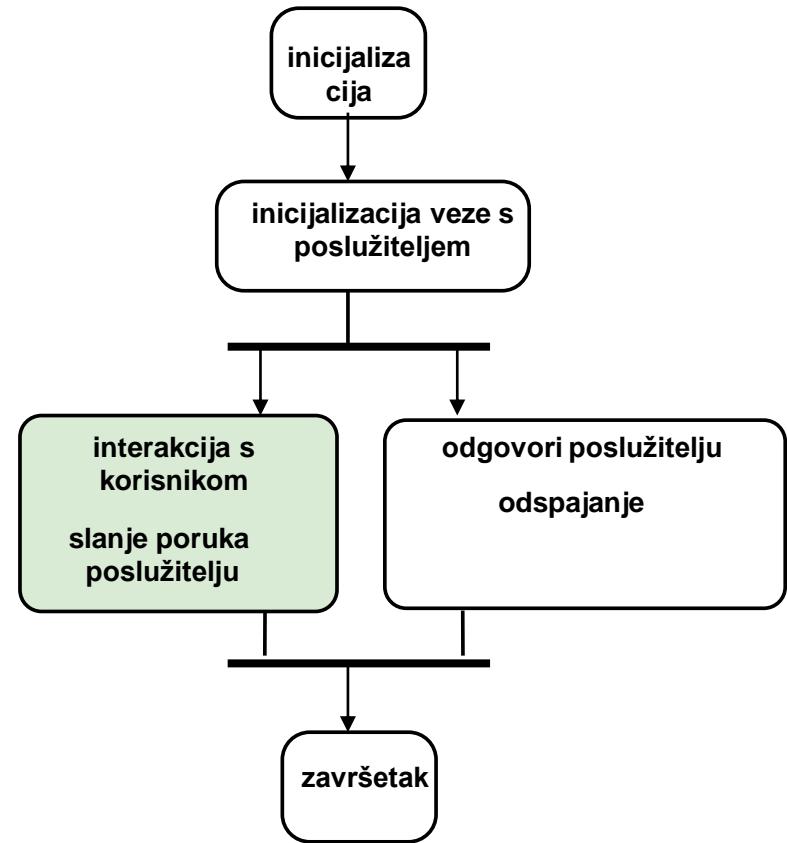
- Inicijalizacija poslužitelja.
- Započinje slušati klijentska spajanja.
- Rukuje slijedećim tipovima događaja koje potiču klijenti:
 1. Prihvata spajanje
 2. Odgovara na poruke
 3. Rukuje odspajanjem klijenta
- Može prestati slušati.
- Mora čisto završiti rad!!



za svako spajanje rukuje spajanjem, reagira na poruku

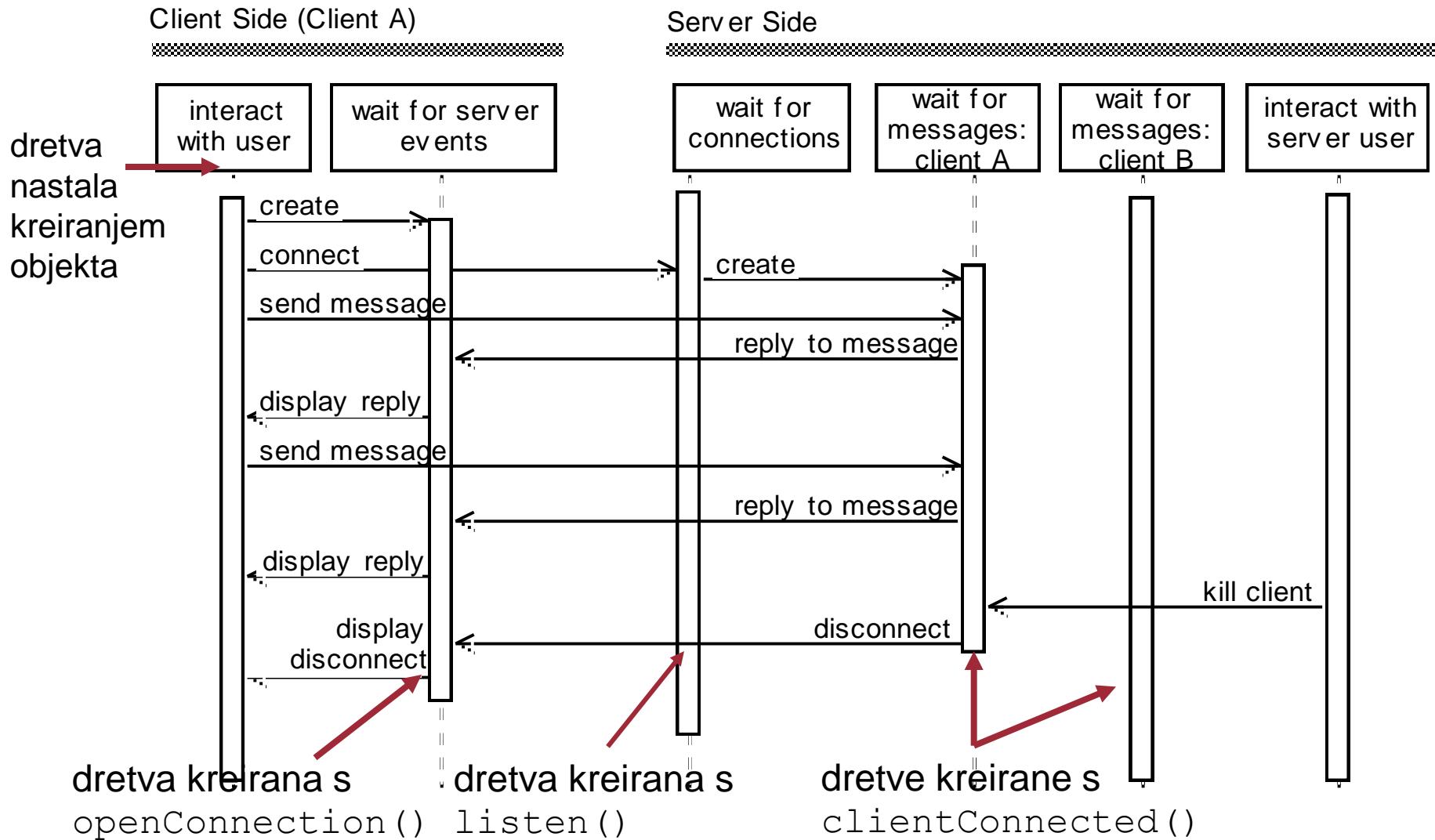
Funkcionalnosti klijenta

- Inicijalizira klijenta.
- Inicijalizira spajanje na poslužitelja.
- Šalje poruke.
- Rukuje slijedećim tipovima događaja koje potiče poslužitelj:
 - odgovara na poruke.
 - rukuje odspajanjem od poslužitelja.
- Mora čisto završiti rad.



Sekvencijski dijagram

- Paralelne dretve/niti izvođenja u klijent - poslužitelj sustavu





Komunikacijski protokoli



- Poruke koje klijenti šalju poslužitelju formiraju jedan jezik.
 - poslužitelj mora biti programiran da razumije taj jezik
- Poruke koje poslužitelj šalje klijentima također formiraju jedan jezik.
 - klijenti moraju biti programirani da razumiju taj jezik
- Kada klijent i poslužitelj komuniciraju oni razmjenjuju poruke uporabom ta dva jezika.
- Ta ***dva jezika i pravila konverzacije*** čine zajedničkim imenom ***protokol***.



Primjer: Internet protokoli



- Internet Protocol (IP)
 - određuje put poruka od jednog računala do drugog.
 - dugačke poruke se cijepaju u manje dijelove.
- Transmission Control Protocol (TCP)
 - nalazi se između primjenskog programa ("aplikacije") i IP protokola.
 - razbija veliku poruku na manje dijelove i šalje dijelove uporabom IP protokola.
 - osigurava da je poruka uspješno primljena (pojedini IP paketi ispravno sastavljeni). TCP je pouzdan protokol.
- Svako računalo (čvor) u mreži ima jedinstvenu IP adresu i ime (*engl. host name*).
 - nekoliko poslužitelja mogu raditi na jednom čvornom računalu u mreži. .
 - svaki poslužitelj na računalu je identificiran preko jedinstvenog broja **ulaznog porta** (*engl. port number*) u rasponu 0 to 65535.
 - kako bi započeo komunikaciju s poslužiteljem klijent mora znati **host name** i **port number**
 - brojevi ulaznih portova 0 – 1023 su rezervirani (npr. port 80 za Web poslužitelj).

Smijete li vi upotrijebiti taj port za nešto drugo?

Oblikovanje klijent-poslužitelj arhitekture

- Preporuke:
 - Oblikuj temeljne poslove poslužitelja i klijenta.
 - Odredi kako će se posao raspodijeliti
 - tanki nasuprot debelog klijenta.
 - Oblikuj detalje skupa poruka koje se razmjenjuju
 - komunikacijski protokol.
 - Oblikuj mehanizme:
 - Inicijalizacije
 - Rukovanja spajanjima.
 - Slanja i primanja poruka.
 - Završetka rada.
- U oblikovanju koristi princip “Uporabi postojeće gotove komponente” (princip br. 6)

RADNI OKVIRI



Koncept ponovne uporabe



- Oblikovanje temeljem iskustva drugih
- Inženjeri koji oblikuju programsku potporu trebaju izbjegavati razvoj programske potpore koja je već bila razvijena.
 - problem?
 - utjecaj kvalitete?
- Tipovi ponovne uporabe:
 - ponovna uporaba znanja
 - *engl. Reuse of expertise*
 - ponovna uporaba standardnog oblikovanja i algoritama
 - *engl. Reuse of standard designs and algorithms*
 - ponovna uporaba knjižnice razreda ili procedura
 - *engl. Reuse of libraries of classes or procedures*
 - ponovna uporaba značajnih naredbi jezika i OS-a
 - *engl. Reuse of powerful commands built into languages and operating systems*
 - ponovna uporaba okvira
 - *engl. Reuse of frameworks*
 - ponovna uporaba cijelih aplikacija
 - *engl. Reuse of complete applications*

Radni okviri

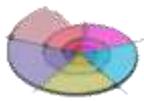
- engl. *Frameworks*
- Podsustavi prilagođeni ponovnoj uporabi
- Radni okvir je učestalo korišten dio programske potpore koji implementira generičko (opće ili zajedničko) rješenje generičkog problema.
 - osigurava opća (zajednička) sredstva koja se mogu uporabiti u različitim primjenским programima.
- Princip:
 - aplikacije/primjeni programi namijenjene obavljanju različitih poslova slične namjene slično su oblikovane.
- Okviri su u osnovi nepotpuni
 - ne postoje razredi ili metode koje koriste (engl. *slot*)
 - nekih funkcionalnosti nije obvezna
 - Prema potrebi mogu se dodati tijekom razvoja (engl. *hook*)
 - pružaju određene usluge
 - Application Program Interface (API)



Objektno usmjereni radni okviri



- engl. *Object Oriented Frameworks*
- U objektno usmjerenoj paradigmi radni **okvir se sastoji iz knjižnice razreda.**
- Primjensko sučelje definirano je kao skup svih javnih (engl. *public*) metoda tih razreda
 - engl. *Application programming interface – API*
 - neki od tih razreda su apstraktni.
 - zahtijevaju implementaciju



Radni okviri i linije proizvoda



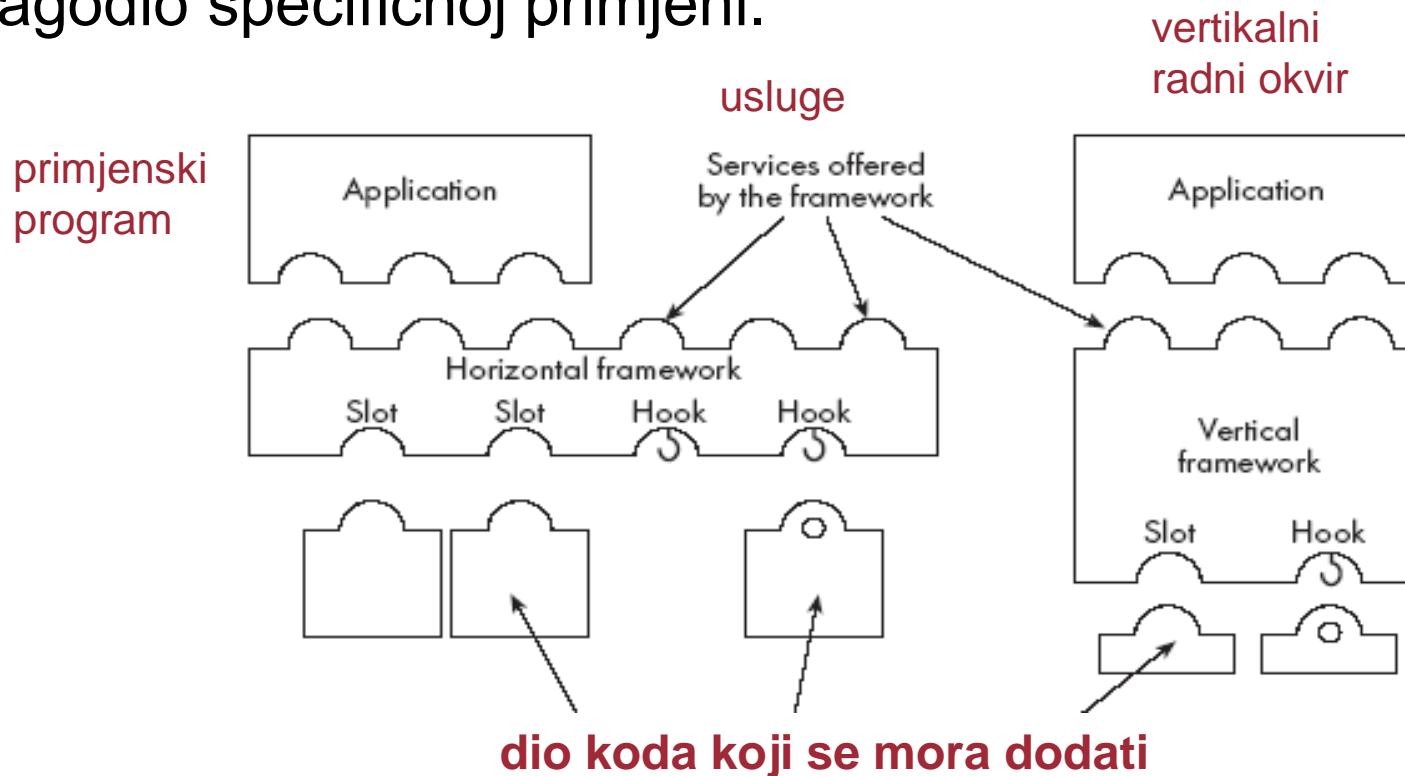
- engl. *software product lines* -SPL
- “*a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way*”, www.sei.cmu.edu/productlines
- Linije proizvoda (ili porodica proizvoda) je skup svih produkata izrađenih na zajedničkoj osnovnoj tehnologiji.
- Različiti produkti u liniji proizvoda imaju različite značajke kako bi zadovoljili različite segmente tržišta.
 - npr. “demo”, “pro”, “lite”, “enterprise” i sl. verzije.
 - lokalizirane verzije su također linije proizvoda.
- Programska tehnologija zajednička svim proizvodima u liniji proizvoda uključena je u radni okvir (engl. framework).
- Svaki proizvod izrađen je temeljem radnog okvira u kojem su popunjena odgovarajuća prazna mjesta.

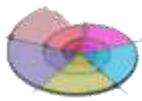
Koji su poticaji i ograničenja?



Arhitekture radnih okvira

- **Horizontalni radni okvir** osigurava opća i zajednička sredstva koja mogu koristiti mnogi primjenski programi (aplikacije).
- **Vertikalni radni okvir** (primjenski) je mnogo cjelovitiji ali još uvijek traži popunu nekih nedefiniranih mesta kako bi se prilagodio specifičnoj primjeni.





Primjeri radnih okvira



- Radni okvir za obradu plaća.
- Radni okvir za podršku tipa čestih putnika (engl. *frequent flyer*) u avioprijevozu.
- Radni okvir za rukovanje čestim kupcima.
- Radni okvir za upis i registraciju predmeta na fakultetu.
- Radni okvir za komercijalno web sjedište (e-dućan).
- Radni okvir za mrežnu uslugu XYZ .

OBJEKTNI RADNI OKVIR KLIJENT- POSLUŽITELJ

- engl. *Object Client-Server Framework* - OCSF
- Metoda oblikovanja arhitekture klijent-poslužitelj temeljena na ponovnoj i višestrukoj uporabi komponenata (engl. reuse).
- Pravila uporabe:
 - ne mijenjati apstraktne razrede u OCSF;
 - kreirati podrazrede;
 - konkretizirati metode u podrazredima;
 - ponovo definirati (engl. override) neke metode u podrazredima;
 - napisati kod koji kreira instance i inicira akcije.
- **Izvorni kod OCSF u Javi nalazi se na web stranicama knjige (potrebno proučiti):**
- T.C.Lethbridge, R.Laganiere: Object-Oriented Software Engineering, 2nd ed., McGraw-Hill, 2005.



Java programski jezik



■ Uporaba `java.net` paketa

- implementacija TCP/IP veze

■ Za uspostavu veze nužno da poslužitelj bude spreman osluškivanje na definiranom portu:

```
ServerSocket serverSocket = new  
    ServerSocket(port);  
Socket clientSocket = serverSocket.accept();
```

■ Ostvarenje veze klijenta s poslužiteljem :

```
Socket clientSocket= new Socket(host, port);
```

■ Razmjena informacija

- uporaba paketa `java.io` (`InputStream`, `OutputStream`)

```
output = new OutputStream(clientSocket.getOutputStream());  
Input = new InputStream(clientSocket.getInputStream());
```

Razmjena poruka

■ Izravno

```
output.write(msg);  
msg = input.read();
```

■ Formatirani podatci DataInputStream / DataOutputStream

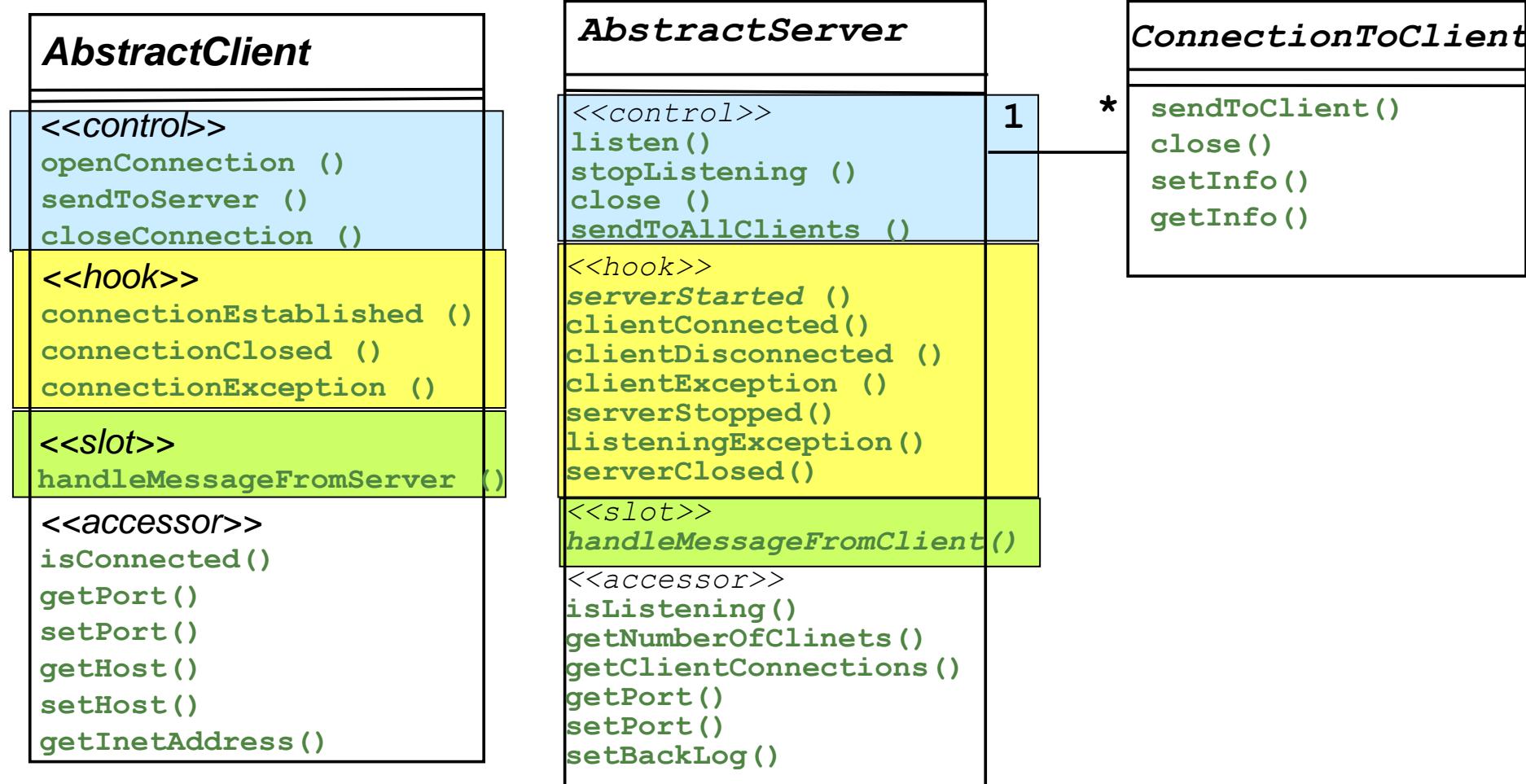
```
output.writeDouble(msg);  
msg = input.readDouble();
```

■ Objekti ObjectInputStream / ObjectOutputStream

```
output.writeObject(msg);  
msg = input.readObject();
```



Object Client-Server Framework (OCSF)





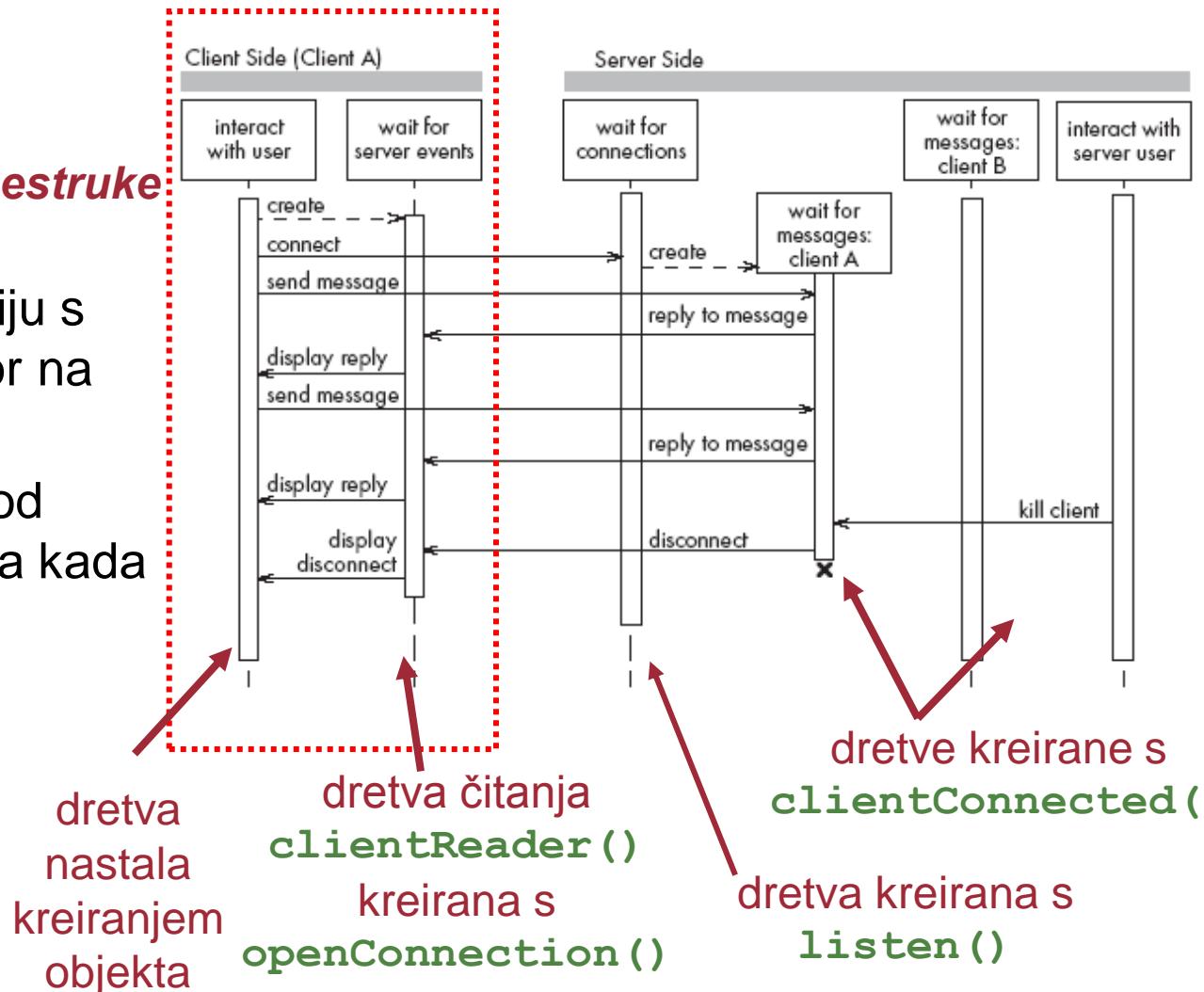
Uporaba OCSF

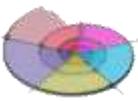


- Programski inženjeri u uporabi OCSF ***nikada ne modificiraju tri navedena razreda.***
- Umjesto modifikacije izvornih razreda treba:
 - kreirati podrazrede apstraktnih razreda u radnom okviru i implementirati metode (učiniti ih konkretnima).
 - zvati javne metode koje uključuje radni okvir. Te metode su usluge koje pruža radni okvir.
 - redefinirati neke metode posebno označene u kategorije **<<hook>>** i **<<slot>>** koje su eksplicitno namijenjene da budu redefinirane.
- Zašto?
 - implementirane (konkretne) metode su rigorozno provjerene (nadamo se) i nisu predviđene za redefiniranje.

Klijent

- engl. *Client Side*
- paralelne aktivnosti na klijentskoj strani – implementirane kao **višestruke niti izvođenja (dretve)**.
 - čekanje na interakciju s korisnikom i odgovor na interakciju.
 - čekanje na poruku od poslužitelja i reakcija kada poruka stigne.





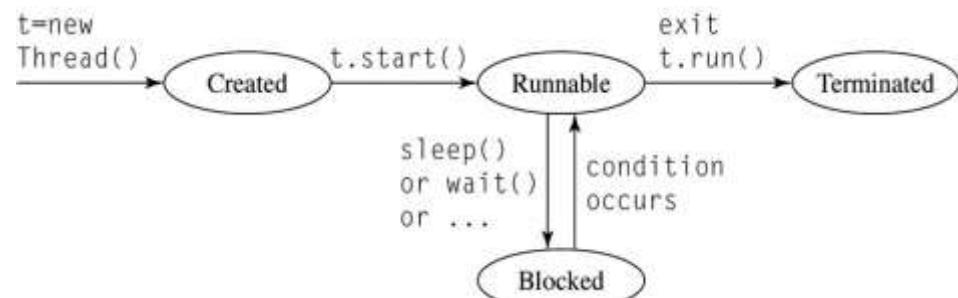
OO Klijent



- Sadrži jedan apstraktan razred: **AbstractClient**
- Iz razreda **AbstractClient** *moraju* se izvesti podrazredi.
 - Svaki podrazred mora osigurati implementaciju operacije:
handleMessageFromServer ()
navedenu u skupu <<slot>> oznake. Metoda je zadužena za odgovarajuću akciju po primitku poruke od poslužitelja.
 - **abstractClient** implementira rad s poslužiteljem. Za čitanje poruka postoji posebna nit izvođenja (dretva) svoje instance (objekta) - vidi sekvensijski dijagram.
 - Dretva započinje izvođenje nakon što upravljačka metoda **openConnection ()** pozove **start ()** dretve naziva **clientReader**
 - ona pokreće **run ()** metodu (glavni program za dretvu).
 - **run** metoda sadrži petlju koja se izvodi tijekom životnog ciklusa dretve (prima poruke i zove metodu za rukovanje s porukama).

Dretve u Javi

- Svaka dretva (engl. *thread*) mora imati **run** metodu.
 - započinje pozivom metode **start**
`t.start()`
 - završava završetkom **run** metode





Dretva ConnectionToClient



```
public class ReadThread implements Runnable
{
    ...
    final public void run()
    {
        object msg;
        clientSocket= new Socket(host, port);
        output = new ObjectOutputStream(clientSocket.getOutputStream());
        input = new ObjectInputStream(clientSocket.getInputStream());

        while(!readyToStop)
        {
            msg = input.readObject();
            handleMessageFromServer(msg);
        }
    }
}
```

Klijentska strana - apstraktan razred



AbstractClient

```
<<control>>
openConnection ()
sendToServer ()
closeConnection ()

<<hook>>
connectionEstablished ()
connectionClosed ()
connectionException ()

<<slot>>
handleMessageFromServer ()

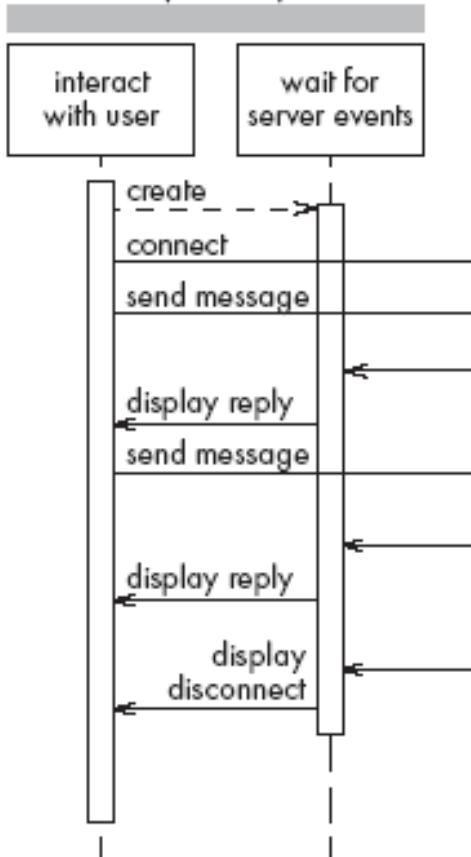
<<accessor>>
isConnected()
getPort()
setPort()
getHost()
setHost()
getInetAddress()
```

- mora imati podrazred koji:
- implementira `handleMessageFromServer`
- obavlja akcije na poruke pristigle od poslužitelja
- implementira kod za pokretanje dretvi
 - u Javi `Runnable` interface
- izvodi `run` metodu za vrijeme života dretve



Sekvencijski dijagram klijenta

Client Side (Client A)



Main()

// korisničko sučelje – GUI, konzola

Ui = new ClientConsole()

ClientConsole()

c = new ChatClient()

Chatclient()

this.openConnection()

Openconnection() //otvori komunikacijski kanal

Run()

Loop:

Msg = input.ReadObject()
handleMessageFromServer(msg)

Handlemessagefromserver()
ui.display(msg)

Ui.Accept()

Accept()

msg = fromConsole.readLine();
c.handleMessageFromClientUi(msg)

Handlemessagefromclientui()
sendToServer(msg)

Sendtoserver()

output.writeObject(msg)



Uporaba razreda `AbstractClient`



1. Kreiraj podrazred od `AbstractClient`
2. U podrazredu implementiraj `handleMessageFromServer` <<`slot`>> metodu
3. Napiši kod koji:
 - kreira instancu novoga podrazreda (start prve dretve)
 - pozove `openConnection` (start druge dretve)
 - šalje poruku poslužitelju uporabom `sendToServer` metode
4. Implementiraj `connectionEstablished` povratnu metodu
 - npr. izvijesti korisnika
5. Implementiraj `connectionClosed` povratnu metodu
 - npr. izvijesti korisnika o čistom završetku veze
6. Implementiraj `connectionException` povratnu metodu
 - npr. kao odziv na kvar na mreži



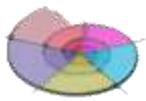
Klijentska strana - openConnection



- Otvaranje veze i komunikacije input - output

```
final public void openConnection () {  
    // ne redefinirati  
    // kreiraj socket i data streams (odlazne i povratne  
    // poruke)  
    clientSocket= new Socket(host, port);  
    output = new  
        ObjectOutputStream(clientSocket.getOutputStream());  
    input = new  
        ObjectInputStream(clientSocket.getInputStream());  
    // kreiraj dretvu čitanja  
    clientReader = new Thread(this);  
    // makni zabranu  
    readyToStop = false;  
    // startaj dretvu koja aktivira run metodu  
    clientReader.start(); }
```

Gdje je definiran start?

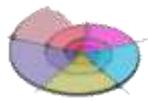


Klijentska strana – run



```
public class ReadThread implements Runnable
{
    ...
final public void run() {
    object msg;                      // varijabla za poruku
    try {
        while(!readyToStop) {
            // dohvati poruke s poslužitelja i poziv metode za obradu poruke
            // dretva čeka poruku na sljedećoj naredbi
            try{ msg = input.readObject();
                // i zove obavezno implementiranu metodu
                // handleMessageFromServer za obradu poruke
                if (!readyToStop) {
                    handleMessageFromServer(msg);
                }
            } catch (Exception exception) { ... }
        }
    }
}
```

U ovom skraćenom prikazu nije prikazana obrada iznimke u **catch** bloku **try-catch** za manipulaciju iznimkama: **catch (Exception ex) { }**



Slanje poruka poslužitelju



- Nije funkcija **clientReader** dretve

```
public void sendToServer(Object msg)
{
    output.writeObject(msg);
    // writeObject je definiran u razredu
    // ObjectOutputStream
    // output preko clientSocket zna kome šalje
}
final public void closeConnection() {
    // zaustavljanje dretve
    readyToStop = true;

    try
    {
        // closeAll() je implementirana u radnom okviru
        closeAll();
    }
    finally
    {
        // poziv metode
        connectionClosed();
    }
}

private void closeAll() {
    try
    {
        //zatvori socket
        if (clientSocket != null)
            clientSocket.close();
        // zatvori output stream
        if (output != null)
            output.close();
        // zatvori input stream
        if (input != null)
            input.close();
    }
    finally
    {
        output = null;
        input = null;
        clientSocket = null;
    }
}
```

Privatni dijelovi razreda `AbstractClient`

- Osobe zadužene za *oblikovanje programske potpore* ne moraju znati te detalje, ali može pomoći u razumijevanju.
- Varijable instanci:
 - `clientSocket` (tipa `Socket`)
 - drži sve informacije o vezi s poslužiteljem.
 - dva niza tipa `ObjectOutputStream` i `ObjectInputStream`
 - koriste za slanje i prijam objekata uporabom varijable `clientSocket`.
 - `clientReader` tipa `Thread`
 - izvodi se `run` metodom objekta razreda `AbstractClient`. Dretva započinje kada `openConnection` pozove `start` koja pozove `run`. Petlja unutar `run` čeka na poruku koja dolazi od poslužitelja. Kada je poruka primljena, `run` zove metodu `handleMessageFromServer`.
 - varijabla `boolean readyToStop`
 - za signalizaciju zaustavljanja dretve čitanja poruka servera.
 - dvije varijable koje čuvaju `host` i `port` adresu poslužitelja.



Klijentska strana - AbstractClient



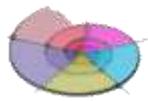
Varijable instance:

```
// kanal za komunikaciju operacijskog  
// sustava  
private Socket clientSocket;  
// nizovi za manipulaciju izlaza - ulaza  
private ObjectOutputStream output;  
private ObjectInputStream input;  
// dretva čitanja  
private Thread clientReader;  
// indikacija kada dretva može završiti  
private boolean readyToStop;  
// ime i broj porta poslužitelja  
private String host;  
private int port;
```



Javno sučelje

- Konstruktor **AbstractClient** pri stvaranju objekta inicijalizira **host** i **port** varijable na koje će se klijent spojiti.
- Upravljačke (<<control>>) metode (dekl. **final**, ne redefinirati):
 - **openConnection** (spaja se na poslužitelj, koristi **host** i **port** varijable koje postavlja konstruktor ili može koristiti metode **setHost**, **setPort**, uspješna veza starta dretvu)
 - **sendToServer** (šalje poruku koja može biti bilo koji objekt)
 - **closeConnection** (zaustavlja rad dretve u petlji i završava)
- Pristupne (<<accessor>>) metode daju info ili mijenjaju vrijednost
 - **isConnected** (ispituje da li je klijent spojen)
 - **getHost** (ispituje koji **host** je spojen)
 - **setHost** (omogućuje promjenu **hosta** dok je klijent odspojen)
 - **getPort** (ispituje na koji **port** je klijent spojen)
 - **setPort** (omogućuje promjenu **porta** dok je klijent odspojen).
 - **getInetAddress** (dobavlja neke detaljnije informacije o vezi)



Klijentska strana - AbstractClient



Konstruktor za **AbstractClient**:

```
public AbstractClient(String host, int port)
{
    // inicijalizacija varijabli
    this.host this.host= host;
    this.port this.port= port;
}
```



Uporaba razreda AbstractClient



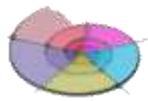
```
public class MyClient extends AbstractClient {  
    . . . // varijable instanci  
    // konstruktor instance  
    public MyClient(String host, int port) {  
        super(host, port); // konstruktor superrazreda  
        openConnection();}  
    .  
    . // metode instance  
    .  
    // glavna konkretizirana metoda  
    public void handleMessageFromServer(Object msg) {  
        .  
        // obrada poruke  
        . . .  
    } }
```



Povratne metode u AbstractClient



- engl. *Callback*
- Metode koje se **mogu** redefinirati
 - označene su kao skupina «hook».
 - ne zovu se izravno
 - npr. u C++ zovu se preko pokazivača
 - najčešće se povratne metode zovu kao odziv na neki asinkroni događaj
 - npr. sva moderna grafička korisnička sučelja zasnovana su na povratnim metodama
 - ako podrazred ima namjeru poduzeti neke akcije kao odziv na događaj
 - **connectionEstablished** (poziva se po uspostavi veze s poslužiteljem)
 - **connectionClosed** (poziva se nakon završetka veze)
 - **connectionException** (poziva se kada nešto pođe krivo, npr. poslužitelj prekida vezu).
- Metoda koja se **mora** implementirati (najvažniji dio koda):
handleMessageFromServer
 - definira se u podrazredima i pozove kada je primljena poruka od poslužitelja.



Uporaba razreda AbstractClient



Povratne metode :

```
// Aktivira se kad je uspostavljena veza  
// "Default" implementacija ne čini ništa  
// Može se ponovo redefinirati ako potrebno  
protected void connectionEstablished() {}  
  
// slično za završetak veze  
protected void connectionClosed() {}  
  
// kao i za obradu iznimke  
protected void connectionException(Exception exception) {}
```



Poslužiteljska strana



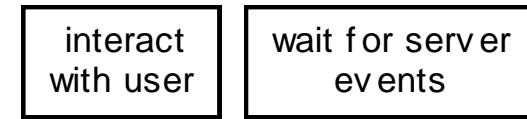
- Poslužiteljska strana sadrži dva razreda
- Potrebne su dvije niti izvođenja (dretve):
 - jedna koja sluša novo spajanje klijenta,
 - druga(e) koja rukuje vezama s klijentima
 - jedna ili više njih
 - sekvensijski dijagram
- Implementacija:
 - instanca razreda **AbstractServer**
 - sluša novo spajanje klijenta.
 - jedna ili više instanci razreda **ConnectionToClient**
 - rukuje vezana s klijentima.



Paralelne dretve klijent - poslužitelj



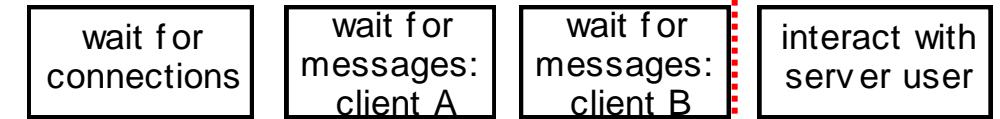
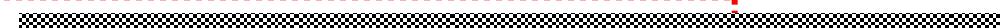
Client Side (Client A)



create
connect
send message

display reply
send message
display reply
display disconnect

Server Side



create
reply to message
reply to message

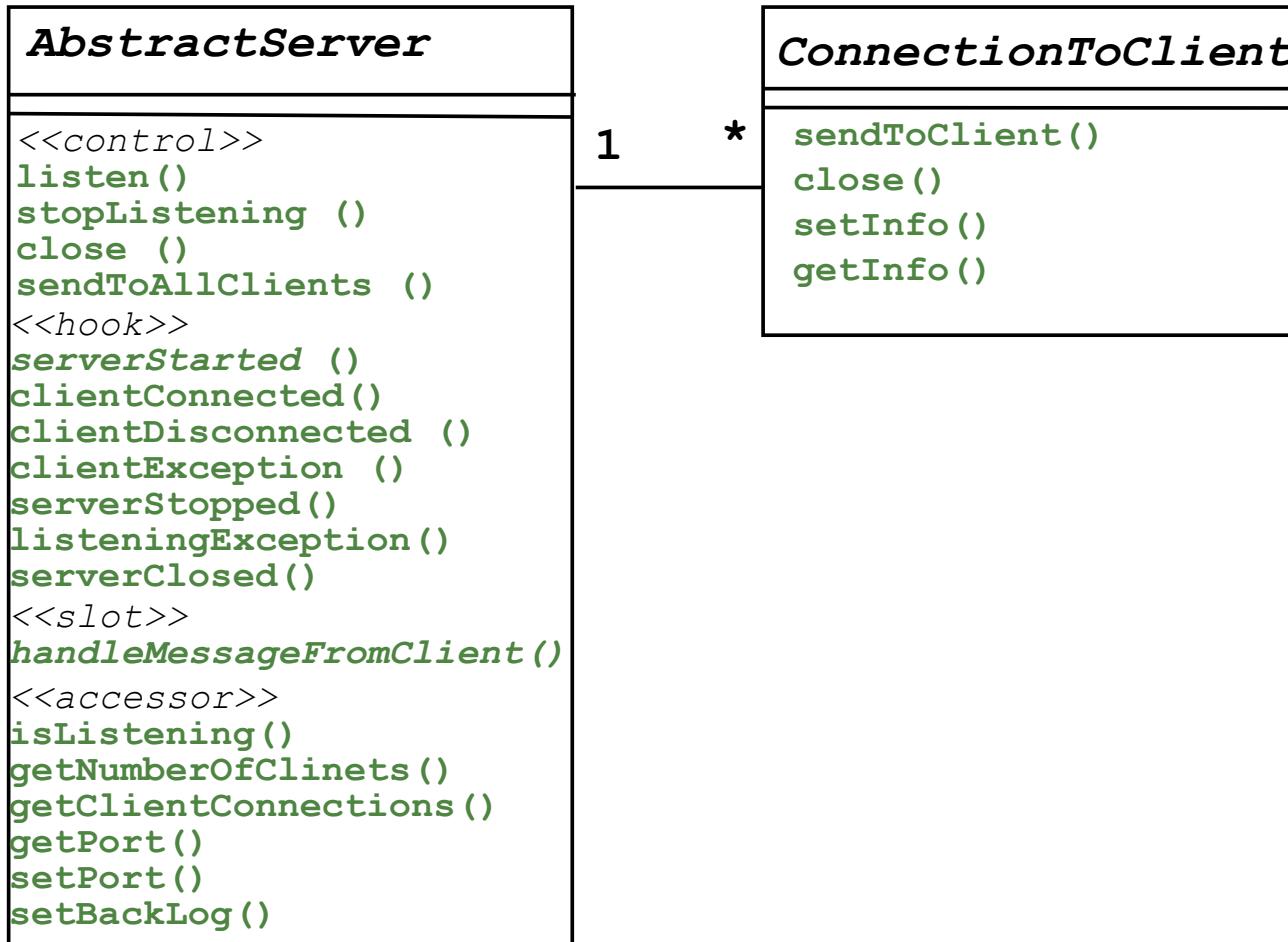
disconnect

kill client

dretva kreirana s
listen()

Dretve kreirane kada i
ConnectionToClient()

Poslužiteljska strana





Javno sučelje AbstractServer



- Konstruktor **AbstractServer**:
 - stvara objekt s brojem porta na kojem poslužitelj sluša. Kasnije se može promijeniti sa **setPort**.
- Upravljačke (<<control>>) metode:
 - listen (kreira varijablu **serverSocket** tipa **Socket** koja će slušati na portu specificiranom konstruktorom, inicira dretvu, a **run** metoda čeka na spajanje klijenta) – vidi raniji sekvenički dijagram.
 - **stopListening** (signalizira **run** metodi da prestane slušati. Spojeni klijenti i dalje komuniciraju).
 - **close** (kao **stopListening** ali odspaja sve klijente)
 - **sendToAllClients** (šalje poruku svim spojenim klijentima)!!
- Pristupne (<<accessor>>) metode:
 - **isListening** (vraća da li poslužitelj sluša)
 - **getClientConnections** (može se iskoristiti za neki rad sa svim klijentima)
 - **getPort** (vraća na kojem portu poslužitelj sluša)
 - **setPort** (koristi se za sljedeći **listen()**, nakon zaustavljanja)
 - **setBacklog** (postavlja veličinu repa čekanja)



Povratne metode u AbstractServer



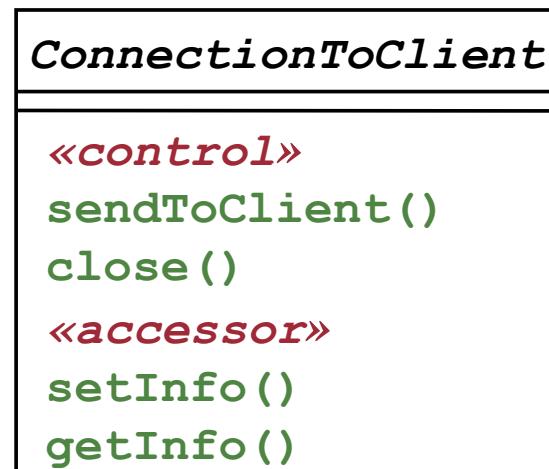
- Metode koje **mogu** biti redefinirane
 - ukoliko su konkretni podrazredi zainteresirani za poseban odziv na događaj).
 - **serverStarted** (poziva se kada poslužitelj započinje prihvati spajanja).
 - **clientConnected** (poziva se kada se novi klijent spoji, sadrži instancu razreda **ConnectionToClient** kao argument) – vidi OCSF.
 - **clientDisconnected** (poziva se kada poslužitelj odspoji klijenta, argument je instanca razreda **ConnectionToClient**).
 - **clientException** (poziva se kada se klijent sam odspoji ili kada nastupi kvar u mreži)
 - **serverStopped** (poziva se kada poslužitelj prestane prihvati povezivanje s klijentima a kao rezultat **stopListening**).
 - **listeningException** (poziva se kada poslužitelj prestane slušati zbog nekog kvara).
 - **serverClosed** (poziva se nakon završetka rada poslužitelja).
- Metoda koja se **mora** implementirati (najvažniji dio koda):
 - **handleMessageFromClient** (argumenti su primljena poruka te instanca razreda **ConnectionToClient**).



Javno sučelje ConnectionToClient



- Tko ga stvara?
- Početna aktivacija metode `listen()` u `AbstractServer` pokreće dretvu koja u svojoj `run()` metodi preko metode `accept()` "socketa" poslužitelja stvara objekt razreda `ConnectionToClient` (s pripadnom dretvom)
 - po jedan za svako spajanje klijenta





Javno sučelje ConnectionToClient



- Za svakog klijenta postoji instanca razreda **ConnectionToClient** za vrijeme dok je klijent spojen na poslužitelja.
- **connectionToClient** je konkretan razred.
 - korisnici ne moraju kreirati podrazrede.
- Upravljačke (<<**controller**>>) metode:
 - **sendToClient** (središnja metoda, koristi se za komunikaciju s klijentom).
 - **close** (odspaja klijenta)
- Pristupne (<<**accessor**>>) metode:
 - **getInetAddress** (dobavlja Internet adresu klijenta)
 - **setInfo** (omogućuje spremanje proizvoljnih informacija o klijentu. Npr. posebne privilegije)
 - **getInfo** (omogućuje čitanje informacija o klijentu).

Uporaba AbstractServer i ConnectionToClient

1. Kreiraj podrazred od **AbstractServer** razreda.
2. U podrazredu implementiraj **handleMessageFromClient**.
3. Napiši kod koji:
 - kreira instancu podrazreda od **AbstractServer**
 - poziva **listen()** metodu
 - odgovara na "call back" **clientConnected()** slanjem poruke uporabom metoda u ovom objektu:
 - **getClientConnections()** ili **sendToAllClients()** (nisu callback metode)
 - odnosno metodom **sendToClient()** u objektu razreda **ConnectionToClient**
4. Po potrebi implementiraj jednu ili više drugih povratnih metoda kao odzive na zanimljive događaje.



Sinkronizacija rada s više klijenata



- Mnoge metode na poslužiteljskoj strani su sinkronizirane. Budući da postoji više **ConnectionToClient** dretvi, koje mogu istovremenu mijenjati podatke na poslužitelju, sinkronizacija garantira da se kritične operacije odvijaju jedna po jedna, te se tako čuva integritet podataka.
- Kolekcija objekata **ConnectionToClient** koje održava **AbstractServer** smješteni su u poseban Java razred **ThreadGroup**.
 - taj razred će automatski maknuti instancu kada njena dretva završi (terminira).
- Poslužitelj mora povremeno (npr. svakih 500ms) provjeriti da li je pozvana metoda **stopListening**. Ako nije, nastavlja s radom.



Implementacija AbstractServer



Varijable instanci

```
// pristupni socket poslužitelja  
private ServerSocket serverSocket = null;  
// dretva slušanja na spajanje klijenta  
private Thread connectionListener = null;  
// broj porta  
private int port;  
// povremeni prekid za provjeru na "stop slušanja"  
private int timeout = 500;  
// duljina repa čekanja klijenata na spajanje  
private int backlog = 10;  
// grupa dretvi pridružena grupi spojenih klijenata  
private ThreadGroup clientThreadGroup;  
// indikacija da li je čitanje spremno za stop  
private boolean readyToStop = true;
```

Implementacija AbstractServer

Konstruktor:

```
public AbstractServer(int port) {  
    // koji port  
    this.port = port;  
    // struktura za klijente  
    this.ClientThreadGroup =  
        new ClientThreadGroup("ConnectionToClient Threads"); }
```

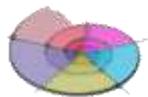
Metode instance:

```
final public void listen() {  
    // definiranje konekcije i start dretve slušanja  
    serverSocket = new ServerSocket(getPort(),  
        backlog);  
    connectionListener = new Thread(this);  
    connectionListener.start(); }  
    // ovaj start pokreće run metodu dretve - vidi nastavak
```

Dijelovi implementacije

Run metoda:

```
final public void run() {  
    // poslužitelj započinje s radom  
    readyToStop readyToStop= false;  
    // poziv callback metode za neku opciju akciju  
    serverStarted();  
    // čekaj na spajanje klijenta  
    while(!readyToStop)  
        socket clientSocket = server.Socket(accept);  
        // kad je klijent spojen, kreiraj novu instancu  
        // ConnectionToClient koja se izvodi kao dretva  
        // vidi konstruktor objekta ConnectionToClient  
        new ConnectionToClient(this.clientThreadGroup,  
            clientSocket, this); }
```

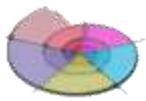


Implementacija ConnectionToClient



Varijable instanci:

```
// klijent mora znati za poslužitelja -  
// asocijacija  
private AbstractServer sever;  
// komunikacijski kanal klijenta  
private Socket clientSocket;  
// nizovi za čitanje i pisanje  
private ObjectInputStream input;  
private ObjectOutputStream output;  
// indikacija da li je dretva spremna na  
// zaustavljanje  
private boolean readyToStop;
```

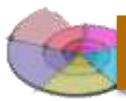


Implementacija ConnectionToClient



Konstruktor objekta ConnectionToClient:

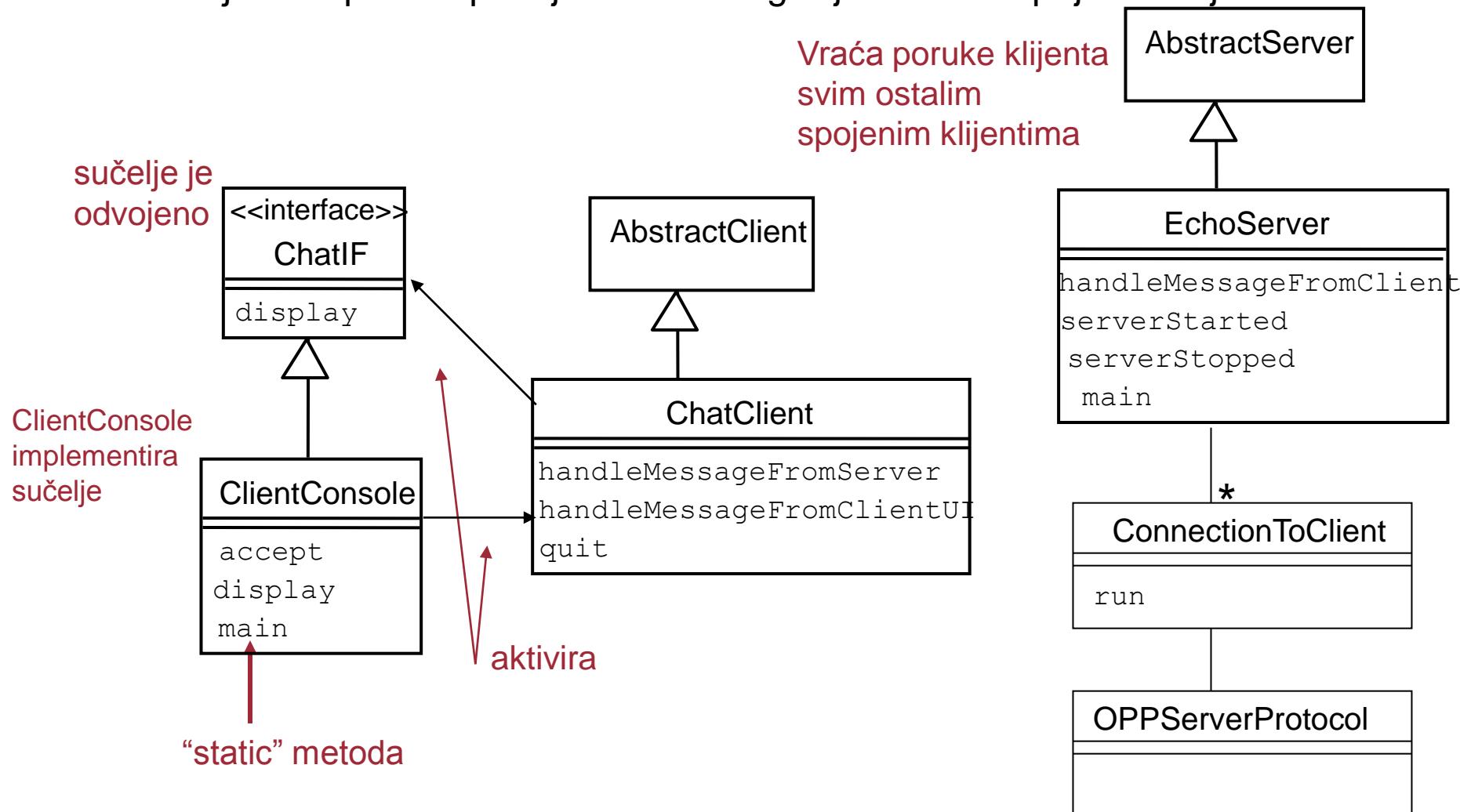
```
protected ConnectionToClient(ThreadGroup group,  
socket clientSocket, AbstractServer server) {  
// inicijalizacija varijabliConnectionToClient (biti će pozvan  
kada se neki klijent spoji (varijabla clientSocket dobiva podatke o klijentu).  
  
this.clientsocket this.clientSocket= clientSocket;  
this.server this.server = server;  
// inicijalizacija nizova  
input input= new  
ObjectInputStream(clientSocket.getInputStream());  
output output= new  
ObjectOutputStream(client.Socket.getOutputStream());  
// start dretve i run metode, čekanje na podatke  
// vidi nastavak  
readyToStop readyToStop= false;  
start(); } //aktivира run() metodu -> slijedeća slika
```

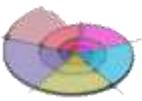


Primjer OCSF-a : Jednostavan “Chat”



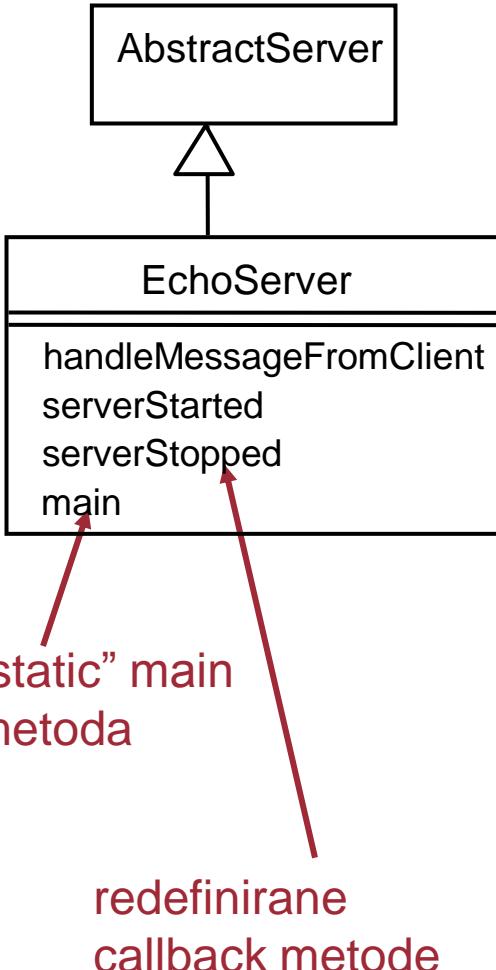
- Jednostavan klijent-poslužitelj sustava razmjene poruka u stvarnom vremenu.
- Poslužitelj vraća poruku primljenu od nekog klijenta svim spojenim klijentima.

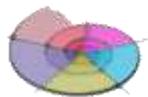




“Chat” poslužitelj

- **echoServer** je podrazred od **AbstractServer** razreda
 - **main** metoda kreira novu instancu i pokreće ju.
 - Instanca razreda **EchoServer** sluša spajanje klijenata (pozivom **listen** metode) i rukuje s vezama sve dok se poslužitelj ne zaustavi.
 - **EchoServer** nema korisničkog sučelja.
 - tri povratne metode samo ispisuju poruku korisniku.
 - **handleMessageFromClient**,
 - **serverStarted**
 - **serverStopped**
 - metoda **handleMessageFromClient** poziva **sendToAllClients**
 - time se proslijeđuju poruke svim klijentima.
 - od **ConnectionToClient** metode postoji instance za spojene klijente (ali se ne koriste njihove metode **sendToClient()**, već metoda **sendToAllClients ()**)





Dio implementacije EchoServer



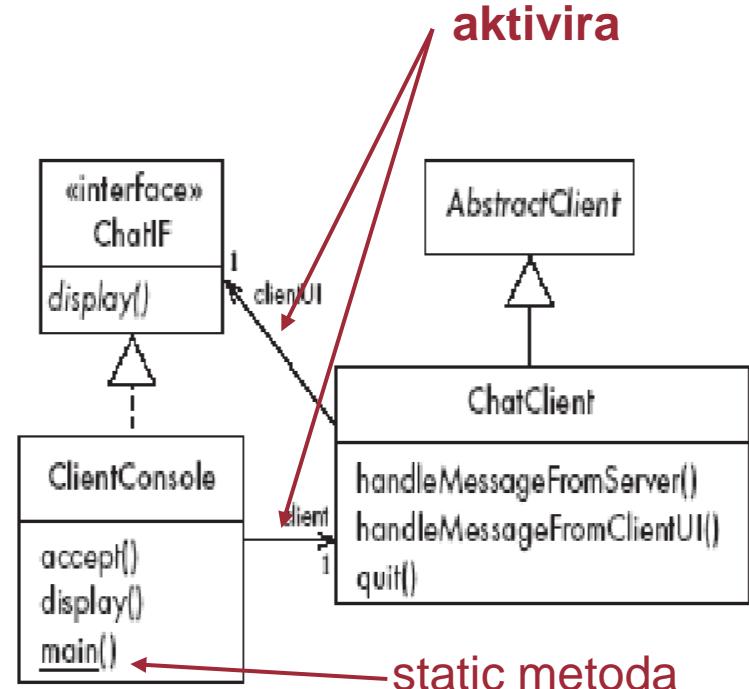
```
Redefinirana metoda handleMessageFromClient: // početna metoda u poslužitelju
public static void main(String[]
args)
{
    port = DEF_PORT;
    // kreiranje objekta
    poslužitelja sv
    EchoServer sv = new
    EchoServer(port);
    // započinjanje slušanja
    sv.listen();
} // kraj main metode
} // kraj razreda EchoServer

// stvaranje podrazreda
public class EchoServer extends
AbstractServer {
final public static int DEF_PORT = 5555;
// konstruktor objekta s definiranim
portom slušanja
// na koji se spajaju klijenti
public EchoServer(int port) {
super(port); }
// metoda za rukovanje porukom mora se
redefinirati

public void handleMessageFromClient (Object msg, ConnectionToClient client)
{
    System.out.println(
        "Message received: "
        + msg + " from " + client);
    this.sendToAllClients(msg);
}
```

“Chat” klijent

- `chatClient` je podrazred od `AbstractClient`.
- `chatClient` redefinira metodu `handleMessageFromServer` koja samo inicira prikaz poruke korisniku.
- Druge dvije metode u `ChatClient` pozivane su od korisničkog sučelja.
- `chatIF` je sučelje odvojeno od funkcionskog dijela, tj. od `ChatClient`.
- Implementaciju sučelja (samo jedne metode `display`) obavlja razred `ClientConsole`.



`ClientConsole`
implementira sučelje
`ChatIF`

“Chat” klijent

- Klijentski program započinje s radom main metodom u **ClientConsole**. Ona kreira instance od dva razreda (**ChatClient** i **ClientConsole**) koje se izvode u dvije odvojene dretve :
 - **chatClient**
 - To je podrazred od **AbstractClient**
 - Redefinira se **handleMessageFromServer**
 - jer samo zove **display** metodu korisničkog sučelja.
 - **clientConsole** (nije podrazred od **AbstractClient**)
 - Korisničko sučelje kao razred je odvojeno je od funkcionalnog dijela klijenta. **ClientConsole** implementira ovo sučelje, tj. implementira **display** metodu koja prikazuje na konzoli.
 - Prihvata ulaz korisničkih podataka pozivom **accept** metode koja šalje sve ulazne podatke objektu razreda **ChatClient** pozovom metode **handleMessageFromClientUI**
 - Metoda **handleMessageFromClientUI** zove metodu **sendToServer** (javna metoda u **AbstractClient**).



Dio implementacije ChatClient



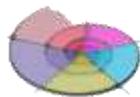
```
chatIF clientUI; // clientUI je varijabla tipa sučelja ChatIF.
```

Slanje poruke poslužitelju:

```
public void handleMessageFromClientUI(String message) {  
    try {  
        sendToServer(message); // prijenos do poslužitelja  
    }  
    catch (IOException e) // rukovanje iznimkom  
    {  
        clientUI.display (  
            "Could not send message. " +  
            "Terminating client.");  
        quit(); } }
```

Primanje poruke od poslužitelja:

```
Public void handleMessageFromServer(Object msg)  
{  
    clientUI.display(msg.toString());  
}
```



```
public class ClientConsole implements ChatIF {  
    // odredi port, varijabla razreda  
    final public static int DEFAULT_PORT = 5555;  
    // varijabla instance, asocijacija s ChatClient  
    chatClient client;  
    // konstruktor implementira objekt razreda ChatClient  
  
    public ClientConsole(String host, int port) {  
        client= new ChatClient(host, port, this); }  
    // metoda za prihvati poruke od korisnika  
    public void accept() { BufferedReader fromConsole =  
        new BufferedReader(new InutStreamReader(System.in));  
        String message;  
        while (true) {  
            message = fromConsole.readLine();  
            client.handleMessageFromClientUI(message); }  
    }
```



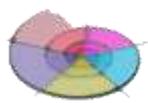
Dio implementacije razreda ClientConsole (2)



```
// metoda display() sučelja je implementirana ovdje
public void display(String message) {
    System.out.println("> " + message); }

// početna metoda razreda za stvaranje ovoga objekta
// klijentska strana započinje rad pozivom ove metode

public static void main(String[] args) {
    host = "local host";
    clientConsole chat=
        new ClientConsole(host, DEFAULT_PORT);
    // prihvati poruke od korisnika
    chat.accept();
}
} // kraj razreda ClientConsole
```



“Chat” klijent – sučelje ChatIF



```
// redefinira se u razredu koji to implementira  
// prikazuje poruku pozivom metode display
```

```
public interface ChatIF  
{  
    public abstract void display(String message);  
}
```



Osnovni kod “Chat” klijenta



```
public void handleMessageFromClientUI (
    String message)
{
    try
    {
        sendToServer(message);
    }
    catch (IOException e)
    {
        clientUI.display (
            "Could not send message. " +
            "Terminating client.");
        quit();
    }
}
public void handleMessageFromServer(Object msg)
{
    clientUI.display(msg.toString());
}
```

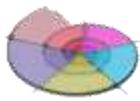
Principi oblik. i raspodijeljena arhitektura

1. Podijeli i vladaj: Podjelom sustava na klijenta i poslužitelja je uspješan način optimalne podjele.
 - klijent i poslužitelj mogu se oblikovati odvojeno.
2. Povećaj koheziju: Poslužitelj osigurava kohezijski spojenu uslugu klijentima.
3. Smanji međuovisnost: Uobičajeno je da postoji samo jedan komunikacijski kanal preko kojega se prenose jednostavne poruke.
4. Povećaj apstrakciju: Odvojene raspodijeljene komponente su dobar način povećanja apstrakcije.
5. Povećaj uporabu postojećeg: Često je moguće pronaći odgovarajući radni okvir (*engl. framework*) temeljem kojega se oblikuje raspodijeljeni sustav. Međutim, klijent-poslužitelj arhitektura je često specifična obzirom na primjenu.

Principi oblik. i raspodijeljena arhitektura

7. oblikuj za fleksibilnost: Raspodijeljeni sustavi se često vrlo lako mogu rekonfigurirati dodavanjem novih poslužitelja ili klijenata.
8. oblikuj za prenosivost: Klijenti se mogu oblikovati za nove platforme bez promjene poslužiteljske strane.
9. oblikuj za ispitivanje: Klijenti i poslužitelji mogu se ispitivati neovisno.
10. oblikuj konzervativno: U kod koji rukuje porukama mogu se ugraditi vrlo stroge provjere (npr. rukovanje iznimkama).

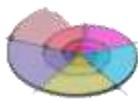
POSREDNIČKA ARHITEKTURA



Posrednička arhitektura



- Predstavlja proširenje raspodijeljene arhitekture klijent – poslužitelj.
- Proširenje se ogleda kroz:
 - slojevitu organizaciju klijenata i poslužitelja
 - Više razina (naslaga, slojeva) - engl. *tier*
 - uvođenje međusloja
 - Posrednici - engl. *middleware*
 - uvođenje zastupnika
 - engl. *broker*



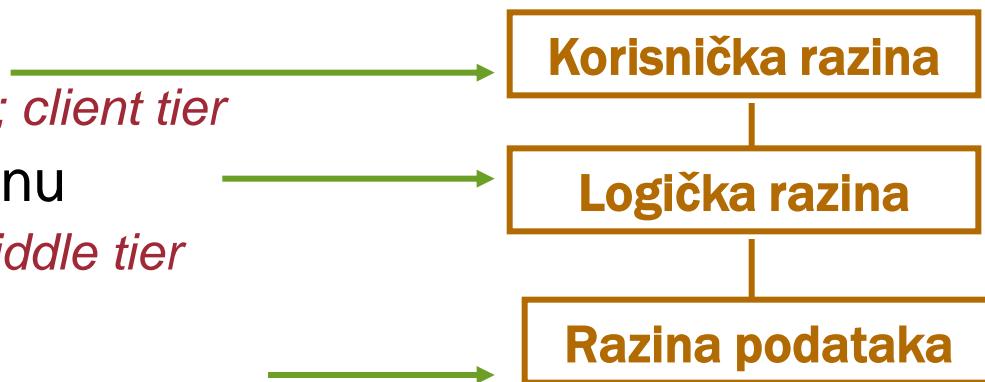
Općenita višerazinska arhitektura



- *engl. n-tier architecture*
- **Uvođenje više razina**
 - klijenti i poslužitelji organiziraju se u **razine** (slojeve, naslage).
- Svaka razina pruža uslugu razini iznad.
- Svaka razina oslanja se na razinu ispod.
- Razina zatvara (skriva, enkapsulira) skup usluga i implementacijske detalje niže razina o kojoj ovisi.
- Često niža razina predstavlja “virtualni stroj” za razinu iznad.
- Višerazinska arhitektura **nije** u posebnom smislu **slojevita** (*engl. layered*).
 - **slojevita arhitektura** odnosi se na strukturu modula, dakle statički pogled,
 - **višerazinska arhitektura** (*engl. tiered*) odnosi na organizaciju u izvođenju (*engl. run-time*), dakle dinamički pogled.

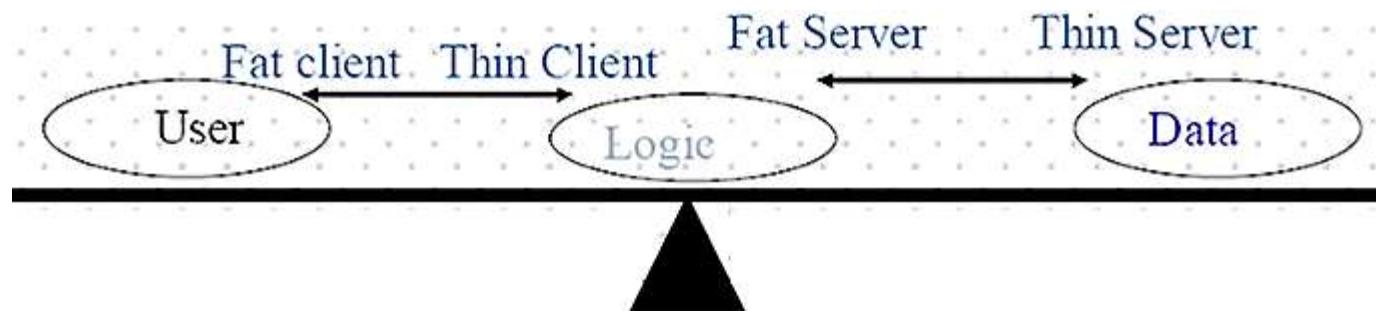
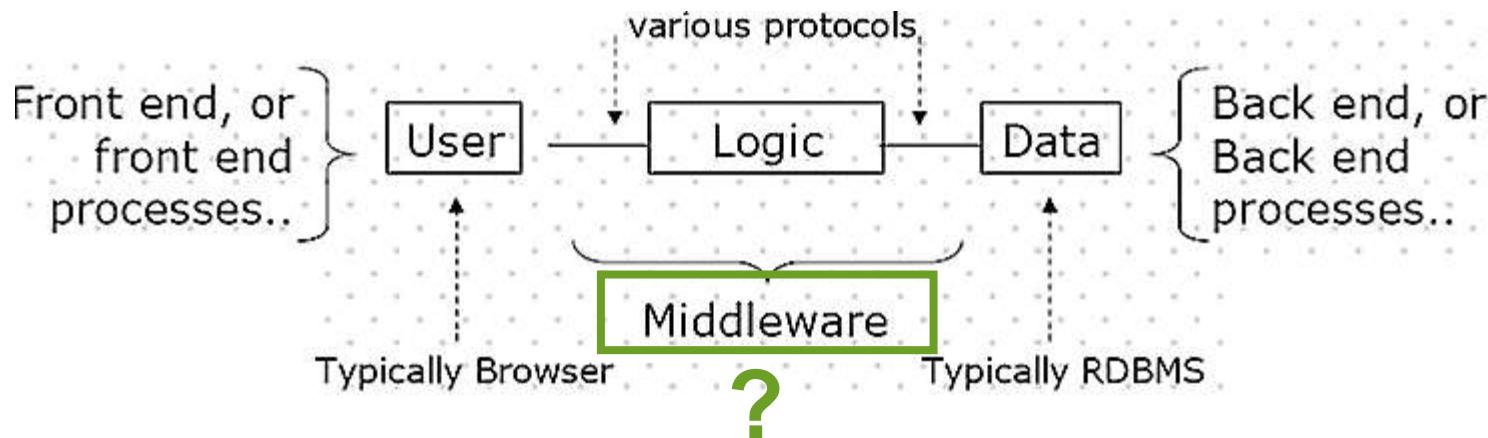
Trorazinska klijent-poslužitelj arhitektura

- engl. *Three-tier (layer) architecture*
- Predstavlja vrstu arhitekture klijent-poslužitelj
- Znatno bolje promovira skalabilnost i mogućnost jednostavnije modifikacije.
- Nastoji otkloniti neke nedostatke klasične klijent-poslužitelj arhitekture, a posebice nastoji povećati performanse, raspoloživost i sigurnost.
- Općenito trorazinska arhitektura sadrži
 - **korisničku** razinu
 - engl. *user; presentation; client tier*
 - **logičku** ili poslovnu razinu
 - engl. *business; logic; middle tier*
 - **podatkovnu** razinu
 - engl. *data tier*



Primjer

- Postoje mnoge varijacije trorazinske arhitekture, ovisno koliko se funkcionalnosti pridodaje svakoj razini.





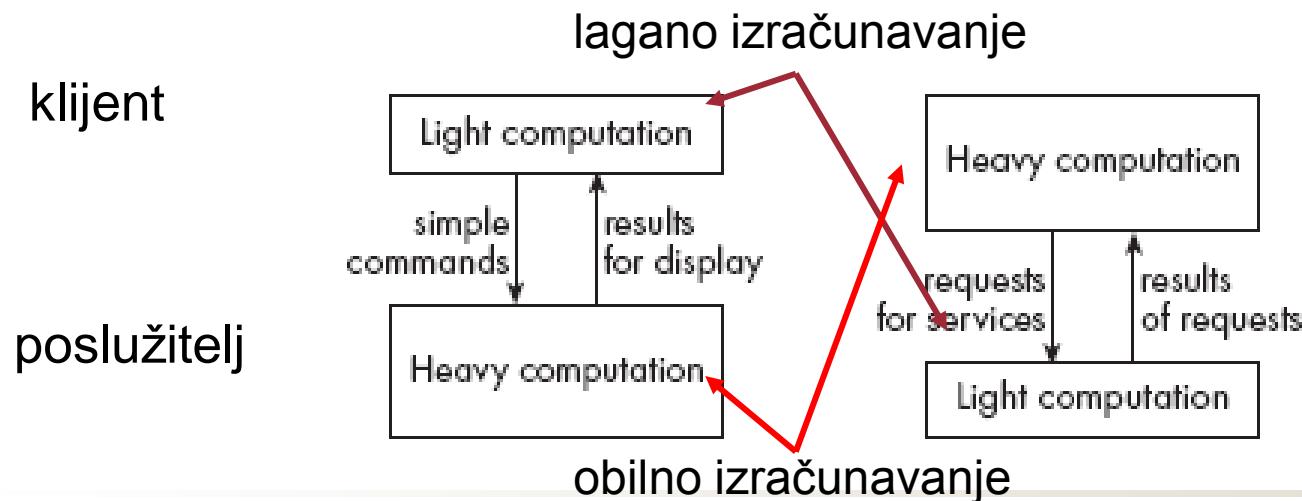
Tanki i debeli klijent

Sustav tankog klijenta (engl. *Thin-client*)

- klijent je oblikovan da bude što je moguće manji i jednostavniji.
- većina posla obavlja se na poslužiteljskoj strani.
- oblikovnu strukturu klijenta i izvršni kod jednostavno se preuzima preko računalne mreže.

Sustav debelog klijenta (engl. *Fat-client*)

- što je moguće više posla delegira se klijentima.
- poslužitelj na taj način može rukovati s više klijenata.





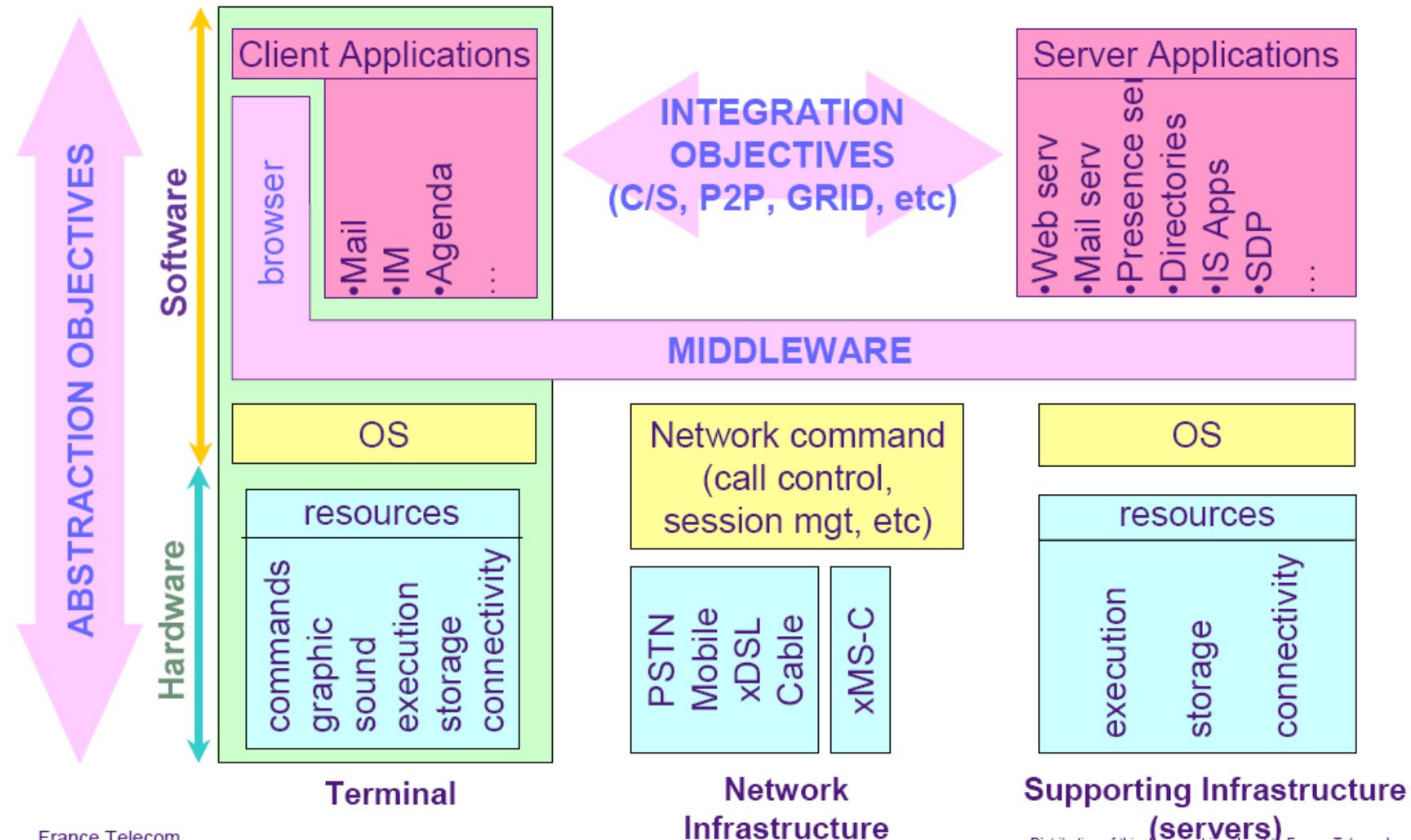
Posrednička arhitektura



- *engl. Middleware*
- Uvođenje posredničke razine
- Nalazi se između klijenta i poslužitelja.
- Programska podrška koja omogućava uzajamno djelovanje aplikacija bez potrebe poznavanja i potrebe za kodiranjem operacija potrebnih za implementaciju.
 - skriva osobama koji oblikuju raspodijeljeni sustav detalje operacijskog sustava i druge specifičnosti implementacije.
 - posrednička razina preuzima detalje komunikacijske mreže.
 - omogućuje osobama koje oblikuju raspodijeljeni sustav da se usredotoče na primjenski (aplikacijski dio).
- Olakšava oblikovanje i razvoj raspodijeljenih sustava



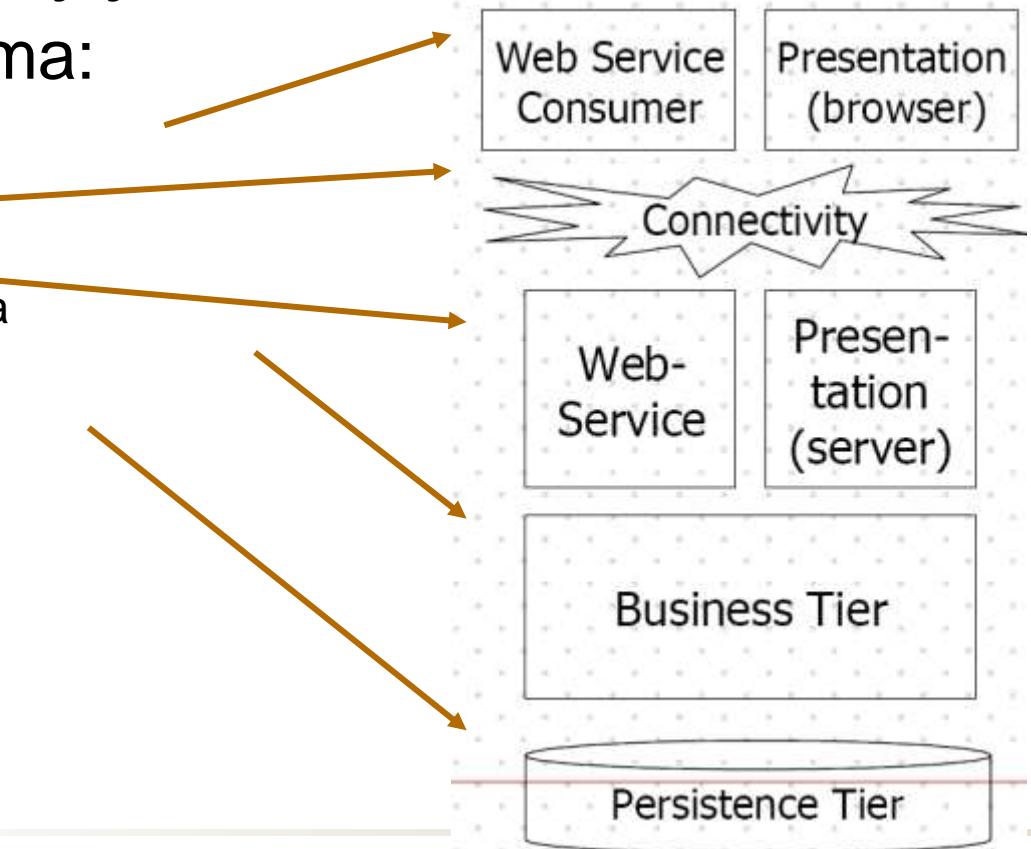
Primjer posredničke razine





Arhitektura s više razina

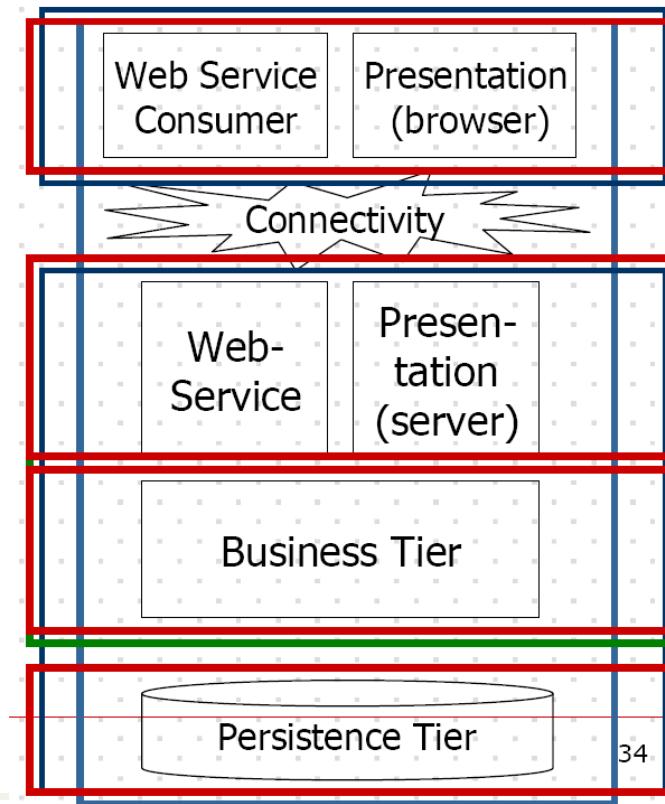
- *engl. N-tier*
- Funkcionalne odgovornosti (najviša razina apstrakcije funkcionalnih zahtjeva u objektno usmjerenoj paradigmi) se raščlanjuju i pridjeljuju razinama.
- Primjer web programa:
 - prezentacijska razina
 - povezivanje
 - web usluge
 - logička (poslovna) razina
 - podatkovna razina (trajno čuvanje)





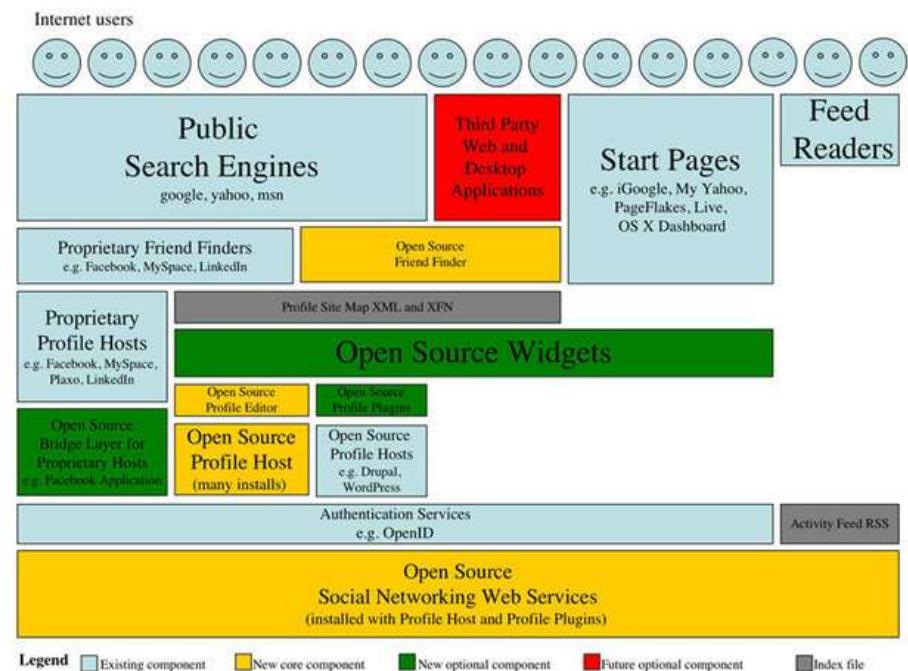
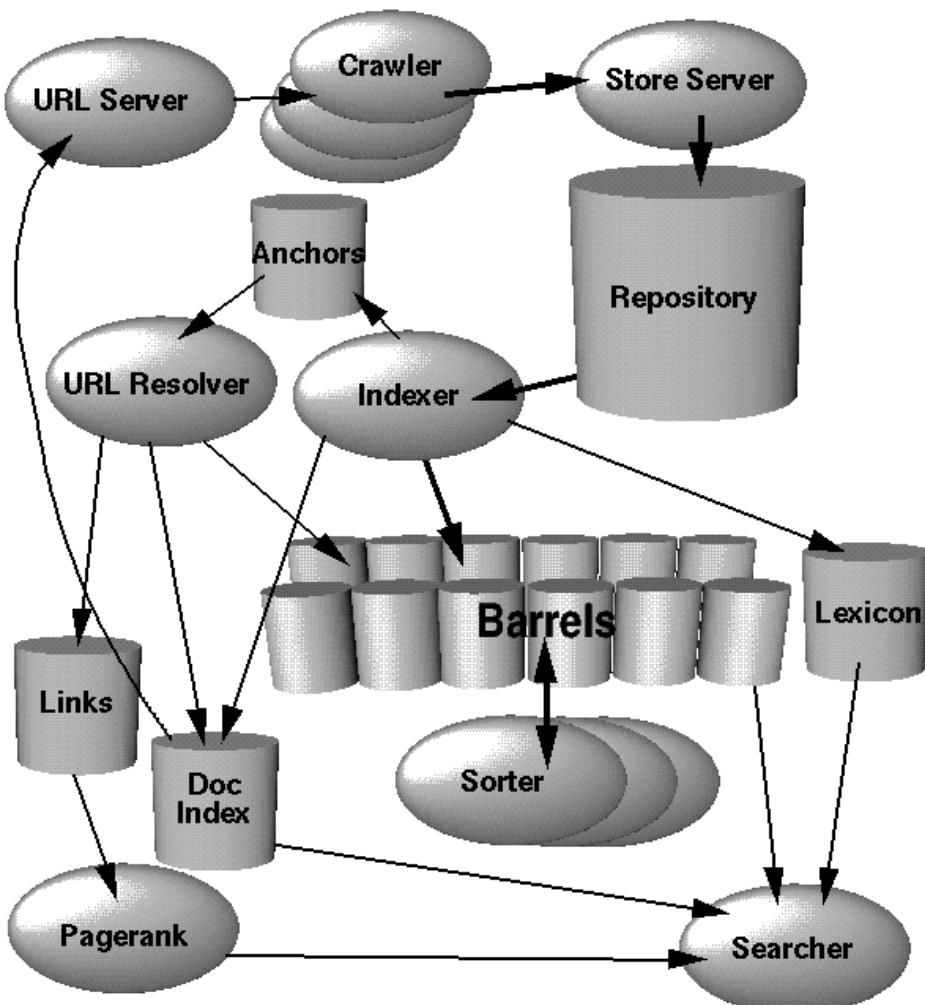
Arhitektura s više razina

- Potrebno je razlikovati alociranje odgovornosti od skupa komponenata u izvođenju.
- Preslikavanje odgovornosti na komponente u izvođenju može se izvesti na više načina.



34

Primjer:



Izvor: Jeff Reifman: Open source social networking architecture

Izvor: Sergey Brin and Lawrence Page: The Anatomy of a Large-Scale Hypertextual Web Search Engine

■ Prednosti:

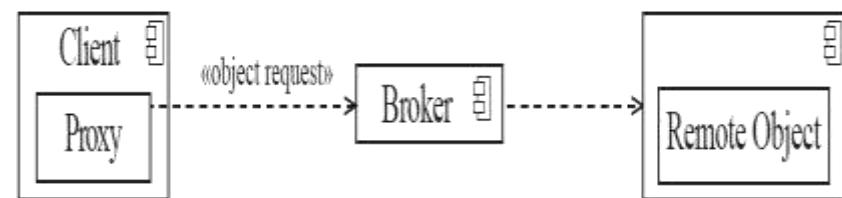
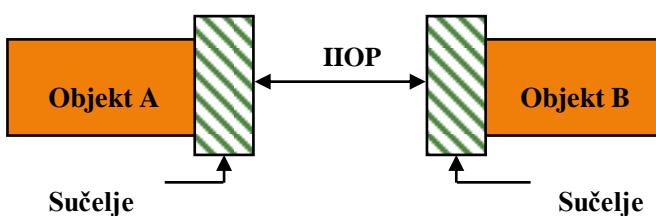
- oblikovanje temeljem više razine apstrakcije.
- podupire povećanje i poboljšanje sustava
 - promjena na jednoj razini utječe samo još na razinu ispod i iznad
- podupire ponovno korištenje, prenosivost i sl.

■ Nedostaci:

- teško je odrediti optimalno preslikavanje odgovornosti na razine.
- ponekad se izračunavanje i funkcionalnosti sustava ne mogu razbiti na razine.
- ako se želi poboljšati performanse mora se preskakati ili "tunelirati" kroz razinu.

Arhitektura zastupnika

- Posrednik, Zastupnik engl. *Broker*
- U objektno usmjerenom stilu arhitekture uvodi se specifična posrednička razina (engl. *specific middleware*) koja omogućuje interoperabilnost u *heterogenim* sustavima (različiti operacijski sustavi, različiti programski jezici) u *raspodijeljenom* okruženju.
- To je infrastruktura za **raspodijeljene objekte**. Time se transparentno alociraju pojedini aspekti programske potpore na pojedine čvorove u mreži.



- Common Object Request Broker Architecture - CORBA je jedna vrlo popularna i standardna implementacijska posrednička razina za oblikovanje raspodijeljenih objektno usmjerениh sustava.
 - **CORBA** standard razvija **OMG (Object Management Group)**
 - <http://www.omg.org/spec/index.htm>



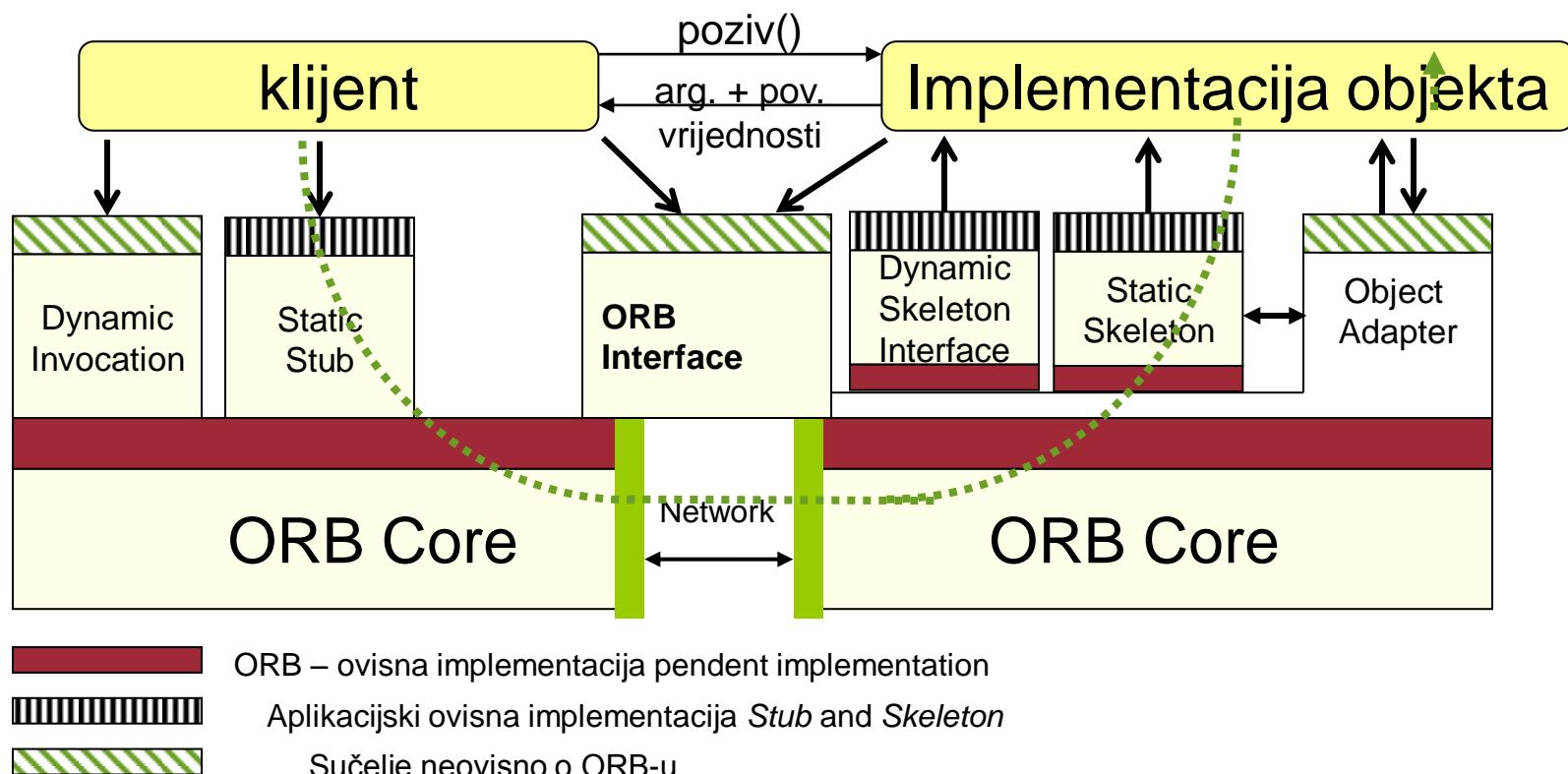
CORBA klijent-poslužitelj



- Poslužitelji posjeduju sučelje koje je neovisno o bilo kojem programskom jeziku. Dostupno je preko jezika **IDL** koji se prevodi u standardne programske jezike (Java, C++, ...).
- Klijenti i poslužitelji mogu se programirati različitim programskim jezicima (ali svaki takav programski jezik mora imati kopče za CORBA-u).
 - **jezična transparentnost** na razini izvornog koda (ne na binarnoj/izvršnoj razini).
- **Posrednik ORB** (Object Request Broker), u obliku programske knjižnice na strani klijenta i poslužitelja prenosi prevedeni zahtjev klijenta do poslužiteljskog objekta.
- Klijenti i poslužitelji ne brinu se o lokaciji jednog ili drugog.
 - **lokacijska transparentnost** - Svi objekti su dostupni preko standardnog sučelja i "object reference" ma gdje bili.
- Klijenti i poslužitelji mogu biti implementirani na različitim sklopovskim platformama i operacijskim sustavima i još uvijek razgovarati međusobno standardnim protokolom.
- CORBA nema raspodijeljeno upravljanje memorijom (*engl. garbage collection*).

Posrednička arhitektura: CORBA model

- ORB – Object Request Broker, je središnja povezujuća posrednička arhitektura za transparentno komuniciranje između objekata.
 - oslanja se i instalira izravno na operacijski sustav i mrežne servise.
 - najčešće je to skup knjižnica.



Object Management Architecture (OMA)

CORBAServices

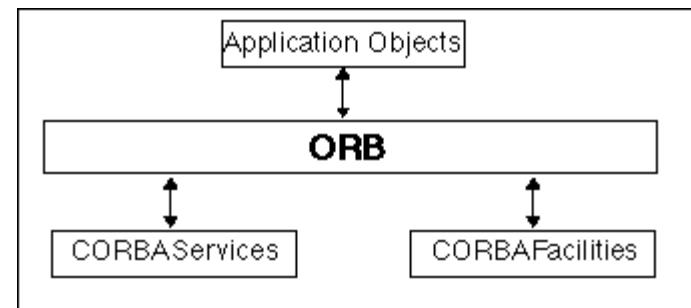
- rukovanje asinkronim događajima, transakcije, paralelnost, imenovanje, odnosi, ...

Application Objects

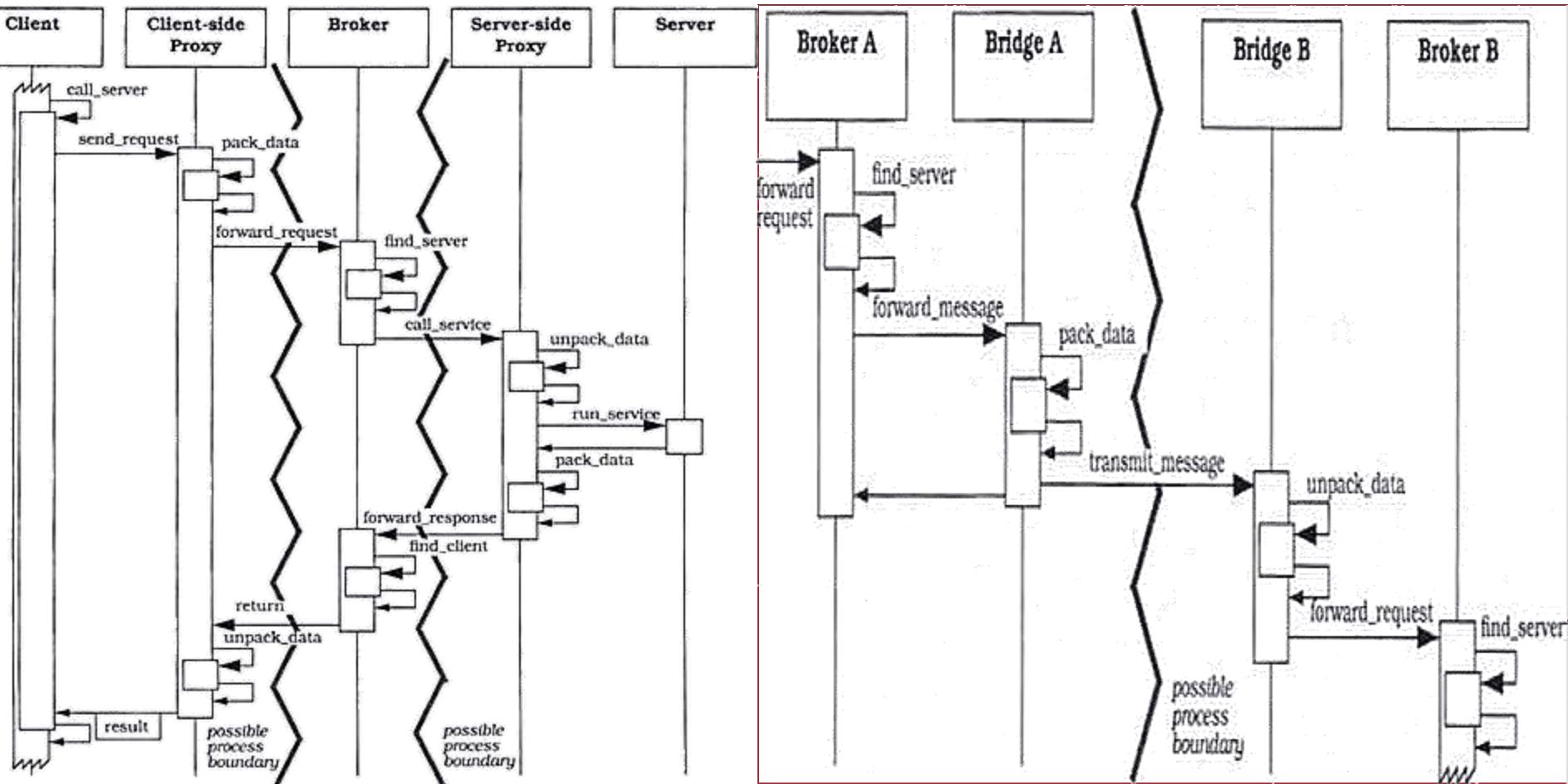
- usluge aplikaciji ili skupu aplikacija

CORBAFacilities

- korisničko sučelje, upravljanje informacijama, sustavom, ...



Primjeri



Prednosti i nedostaci

■ Prednosti

- transparentnost lokacija;
- izmjenjivost i proširivost komponenti;
- prenosivost;
- interoperabilnost različitih sustava Brokera;
- ponovna uporaba.

■ Nedostaci

- smanjena efikasnost
- veća osjetljivost na pogreške
 - engl. *fault-tolerance*

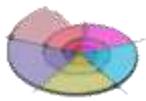
USLUŽNO USMJERENA ARHITEKTURA



Uslužno usmjerena arhitektura

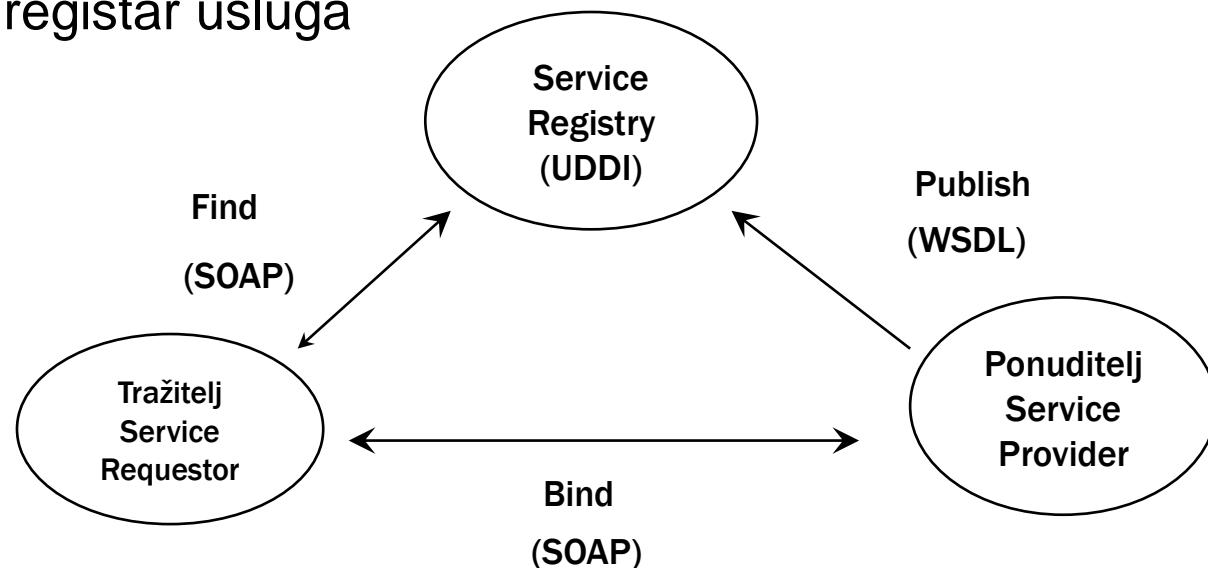


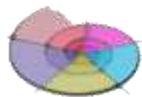
- engl. *Service oriented architecture (SOA), Service oriented architectural pattern (SOAP)*
- **Uslužno usmjerena arhitektura** organizira primjenski program (cjelovitu aplikaciju) kao kolekciju usluga koje međusobno komuniciraju uporabom dobro definiranih **sučelja**.
- Npr. Internet okruženje – Web usluge (engl. *Web services*)
 - web usluga = aplikacija dostupna putem Interneta, u svrhu izgradnje složenog sustava omogućava integraciju s ostalim Web uslugama
 - Dobro definirana funkcija
 - Samodostatna
 - Ne ovisi o drugim uslugama (okolina ili stanje)
 - komunikacija – standardnim protokolima razmjene podataka npr. XML



Model web uslužne arhitekture

- Oglašavanje, otkrivanje selekcija i uporaba web usluge
 - tehnologije zasnovane na XML-u
- Simple Object Access Protocol
 - razmjena poruka Web usluga
- Web Service Definition Language
 - definira web uslugu
- Universal Description, Discovery, and Integration
 - API za registar usluga

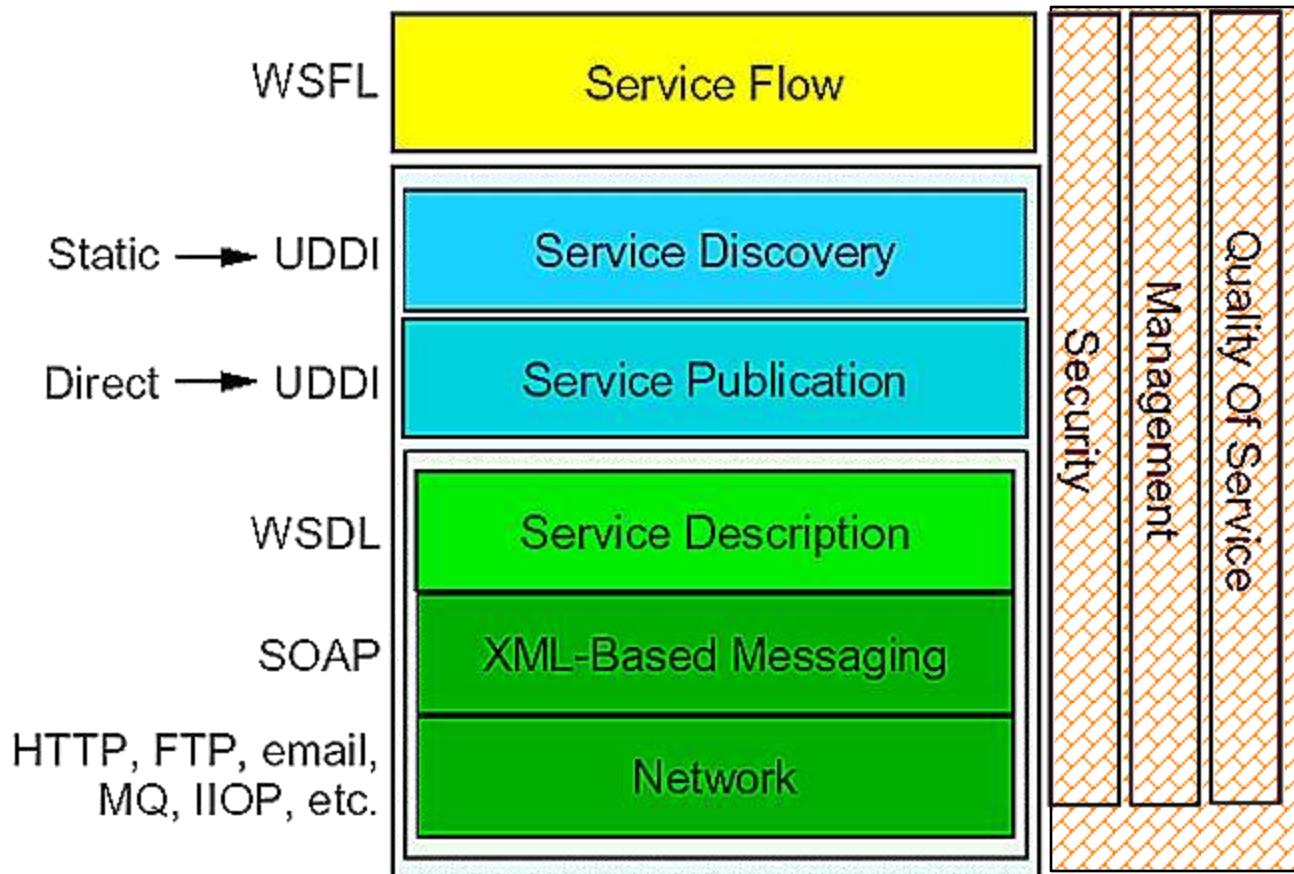




Web usluga



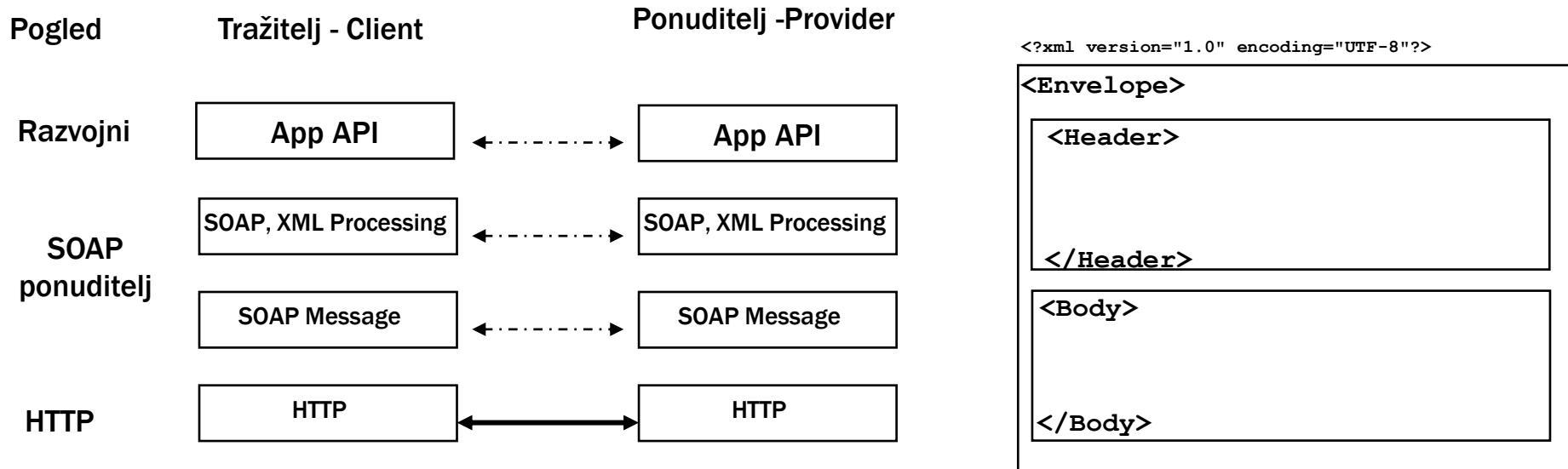
■ Struktura Web usluge



SOAP poruka

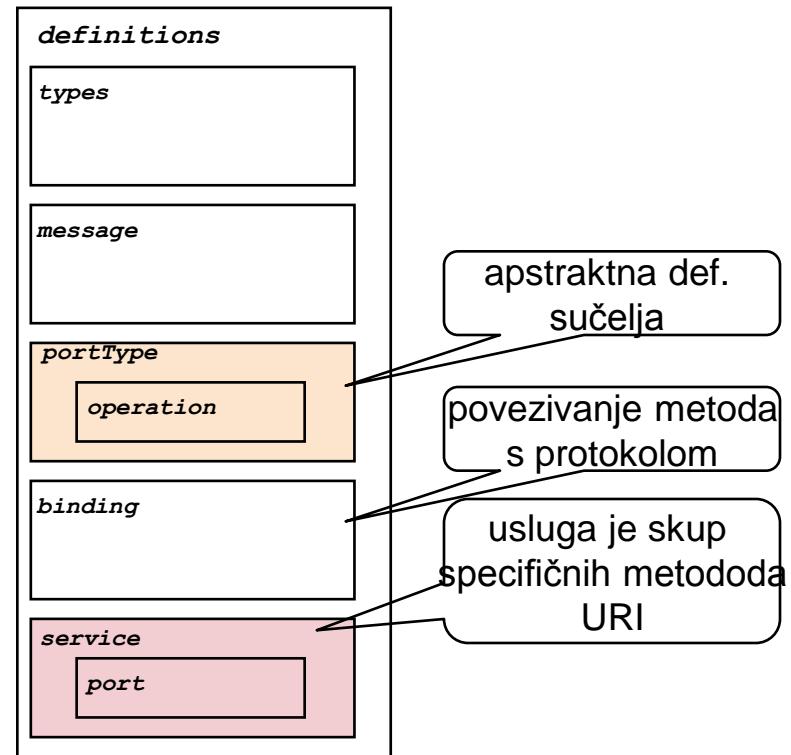
■ SOAP je komunikacijski protokol

- neovisan o platformi
- zasnovan na XML-u
- namijenjen komunikaciji aplikacija
- zamjenjuje ranije razvijene protokole u Internet okruženju

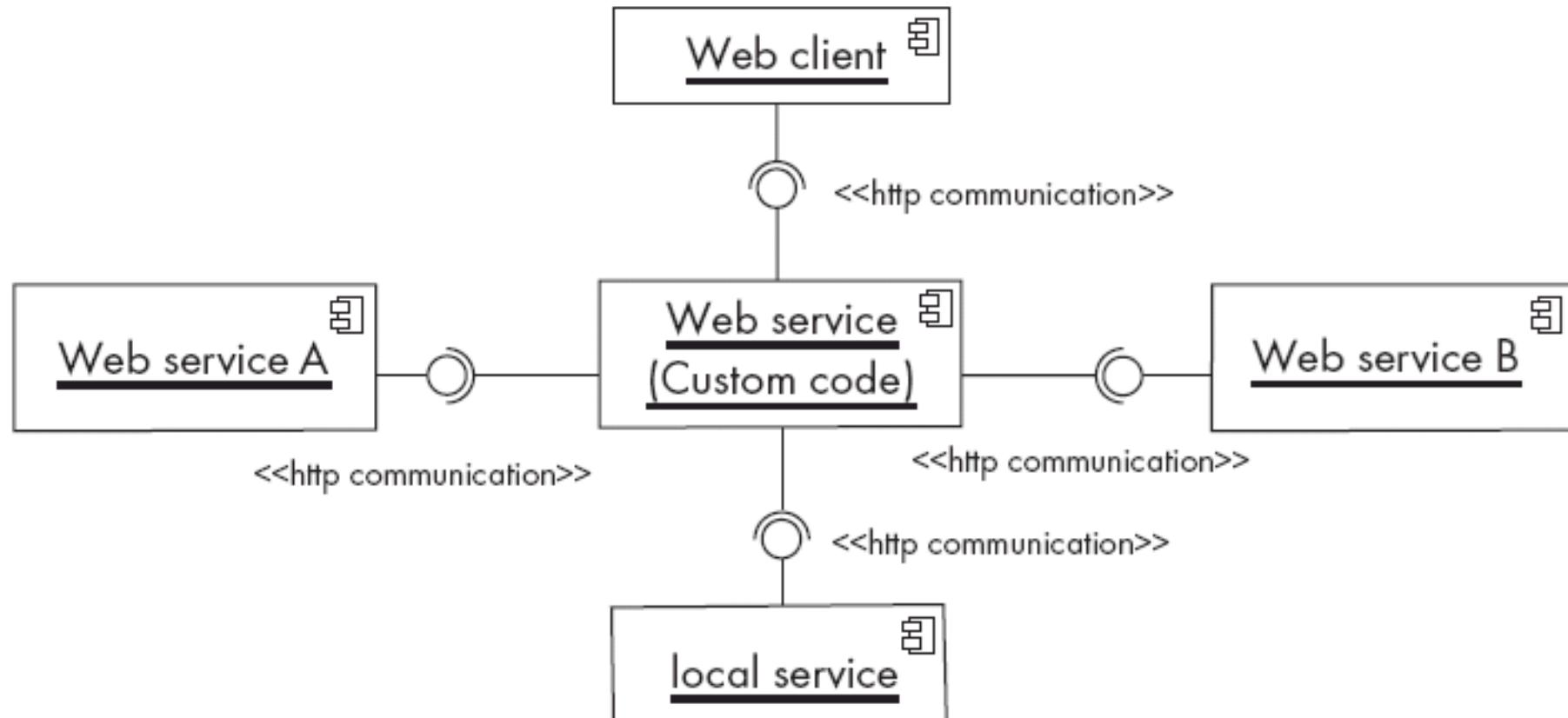


WSDL

- XML dokument prema WSDL specifikaciji
- Opisuje
 - što radi servis
 - Operacije (metode) koje može pružiti
 - način pristupa
 - Format podataka i opis protokola
 - smještaj/lokaciju usluge
 - URL adresa
- Alati za automatsko generiranje koda iz WSDL-a



■ UML dijagram komponenata





Utjecaj Web usluga



- Pojednostavljaju integraciju
 - unutar i između organizacija
- Faze uvođenja Web usluga i uslužno usmjerene arhitekture
 1. eksperimentiranje
 - Upoznavanje, vanjske web usluge, razmjena podataka, školovanje
 2. prilagodba postojećih sustava
 3. uklanjanje međuvisnosti
 4. uspostava interne arhitekture zasnovane na principima uslužno usmjerene arhitekture
 5. uključivanje vanjskih Web usluga
 6. pružanje usluga drugima
- Proces razvoja i standardizacija uklanja nepovjerenje
- Podizanje razine kvalitete
- Iskorištavanje novih mogućnosti

Principi oblikovanja i uslužna arhitektura

1. Podijeli i vladaj: Cijeli primjenski program sastoji se iz neovisno oblikovanih komponenata - usluga.
2. Povećaj koheziju: Web usluge su strukturirane kao slojevi i imaju dobru funkcionalnu koheziju.
3. Smanji međuovisnost; Web zasnovani primjenski programi su slabo vezani i oblikovani su spajanjem raspodijeljenih komponenata.
5. Povećaj ponovnu uporabu: Web usluga je posebice značajno ponovno uporabiva komponenta.
6. Povećaj uporabivost postojećeg: Web zasnovane usluge su oblikovane ponovnom uporabom postojećih web usluga.
8. Planiraj zastaru: Zastarjele usluge mogu se zamijeniti novim implementacijama bez utjecaja na primjenske programe koje ih koriste.

ARHITEKTURA PROGRAMSKE POTPORE ZASNOVANA NA KOMPONENTAMA

- engl. *Component Based Design - CBD*
- Pouke procesa oblikovanja programske potpore s fokusom na princip ponovne uporabe dijelova.

	Objektno usmjeren pristup	Oblikovanje zasnovani na komponentama
Sastavi komponente u jedinstveni produkt	<u>Teško</u> , traži se vještina objektnog programiranja	<i>Može izvesti vješt korisnik</i>
Oblikuj komponente	<u>Teško</u> , traži se vještina objektnog programiranja	<u>Jako teško</u> , mora se voditi računa o mnogo korisnika

Izvor: A. R. Newton, UC Berkeley

- Rekapitulacija principa ponovnog korištenja
- U fokusu arhitekture zasnovane na komponentama je njihovo ponovno i višestruko korištenje. Princip višestrukog korištenja u oblikovanju programske potpore manifestirao se tijekom vremena kroz:
 - Ponovno korištenje konzistencije (programski jezici).
 - Ponovno korištenje fragmenata rješenja (knjižnice).
 - Ponovno korištenje dijelova arhitekture (arhitektturni obrasci – *engl. architectural patterns, design patterns*).
 - Ponovno korištenje arhitekture (radni okviri – *engl. frameworks*).
 - Ponovno korištenje cjelokupne arhitekture sustava.

■ **Programski jezik:**

- uvodi kulturu kako se oblikuje program.
- često uvodi dogmu kako neke stvari treba napraviti.
- programski jezik ne osigurava ispravno oblikovanje ali isključuje mnoge stvari koje unose pogreške.

■ **Knjižnice:**

- prvi programske jezici imali su uključene sve funkcionalnosti.
- C jezik (1978) izvlači specifične rutine u knjižnice (npr. I/O).
- od tada postoji tendencija odvajanja posebnih funkcionalnosti.

■ **Arhitekturni oblikovni obrasci:**

- nastoji se izgraditi katalog najmanjih ponavljajućih dijelova arhitekture (obrascima) u objektno usmjerrenom programiranju.
- obrazac je definiran imenom, opisom problema (s uvjetima) koji rješava, elementima (razredi, odnosu između razreda, odgovornosti, kolaboracije), te navođenjem dobroih strana uporabe obrasca.
- e.Gamma je 1995. identificirao 23 oblikovna obrasca. Danas znatno više.

- ***Radni okviri:***

- skup razreda i interakcija.
- radni okviri traže oblikovanje podrazreda i implementaciju nekih operacija.
- radni okviri najčešće nisu posebno prilagođeni pojedinim primjenama već oslikavaju određene koncepte.

- Cjelokupna arhitektura:

- Uvodi zasebne stilove, npr.:

- cjevovodi i filtri
- događajno usmjerena
- repozitorij podataka
- virtualni strojevi
- ...

■ ***Definicija programske komponente***

- programska komponenta je jedinica kompozicije s ugovorno specificiranim sučeljem i kontekstnom ovisnošću.
- programska komponenta može se nezavisno razmjestiti u sustavu kojega oblikuju drugi dionici.
- programska komponenta je binarna jedinica kompozicije nezavisno proizvedena.
- programska komponenta nastoji se potvrditi na tržištu komponenata.

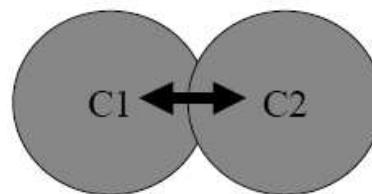
■ ***Razlika između objekta i komponente***

- definicija objekta je tehnička; ne uključuje pojam nezavisnosti.
- kompozicija objekata nije namijenjena širokom krugu korisnika.
- ne postoji niti će postojati tržište objekata.
- objekti ne podržavaju paradigmu priključi-i-radi (engl. *plug-and-play*)

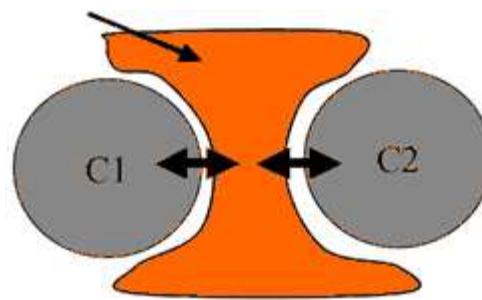
Arhitektura zasnovana na komponentama



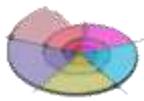
- Temeljni problem u širokoj uporabi komponenata je nepostojanje zajedničke platforme (okruženja) koja objedinjuje komponente.
- Danas komponente surađuju preko definiranog sučelja:



- Potreba za zajedničkom platformom (novim "operacijskim sustavom"):



- U arhitekturi sustava zasnovanog na web uslugama:
 - web = Operacijski sustav



Najvažnije tehnologije



- Najznačajnija uloga u omogućavanju distribuirane komponentne arhitekture:
- Object Management Group – CORBA - Common Object Request Broker Architecture
- Oracle/Sun Microsystems - Java Beans, EJB(Enterprise Java Beans)
- Microsoft – COM - Component Object Model, DCOM - Distributed COM, .NET Remoting

CORBA

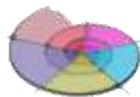
- Dobro: standard (OMG grupa), transparentna komunikacija između objekata koji su oblikovani različitim programskim jezicima i žive na različitim strojevima, u heterogenoj raspodijeljenoj okolini.
- Loše: Definirano na razini izvornog koda, a ne na binarnoj razini.
 - jako usporava rad jer se komunikacija odvija na visokoj razini definiranih protokola.
 - programski jezici moraju imati implementirane kopče za CORBA sučelje.

Java/JavaBeans:

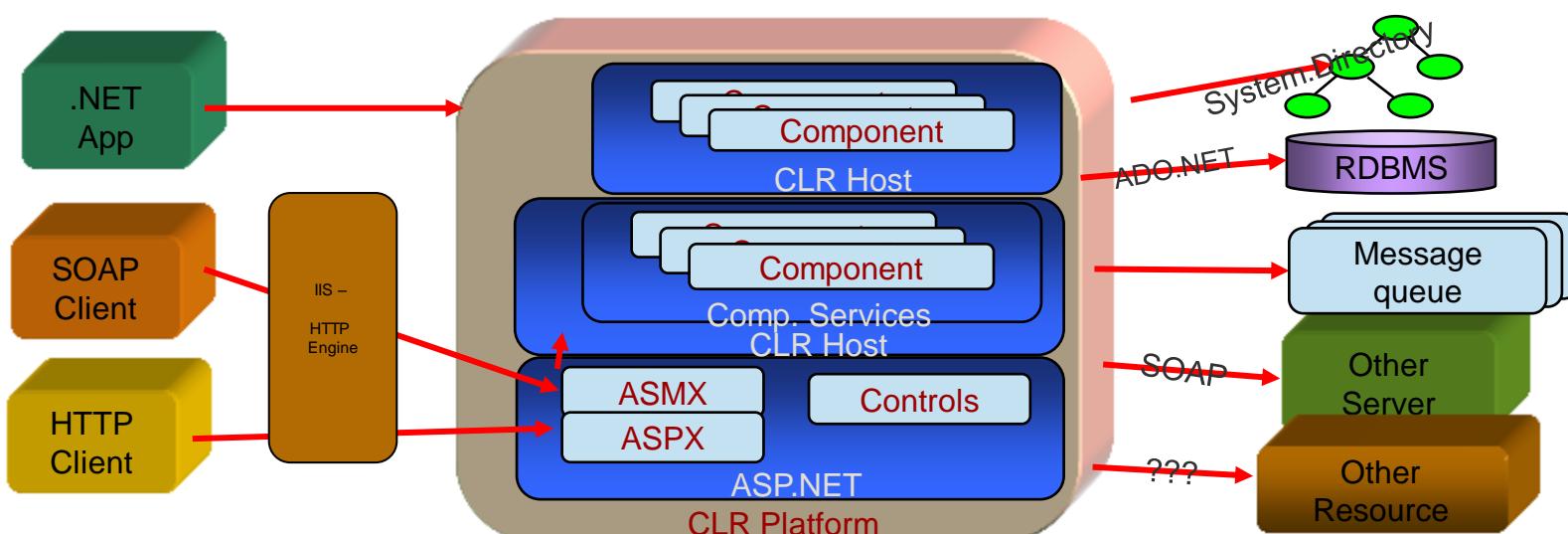
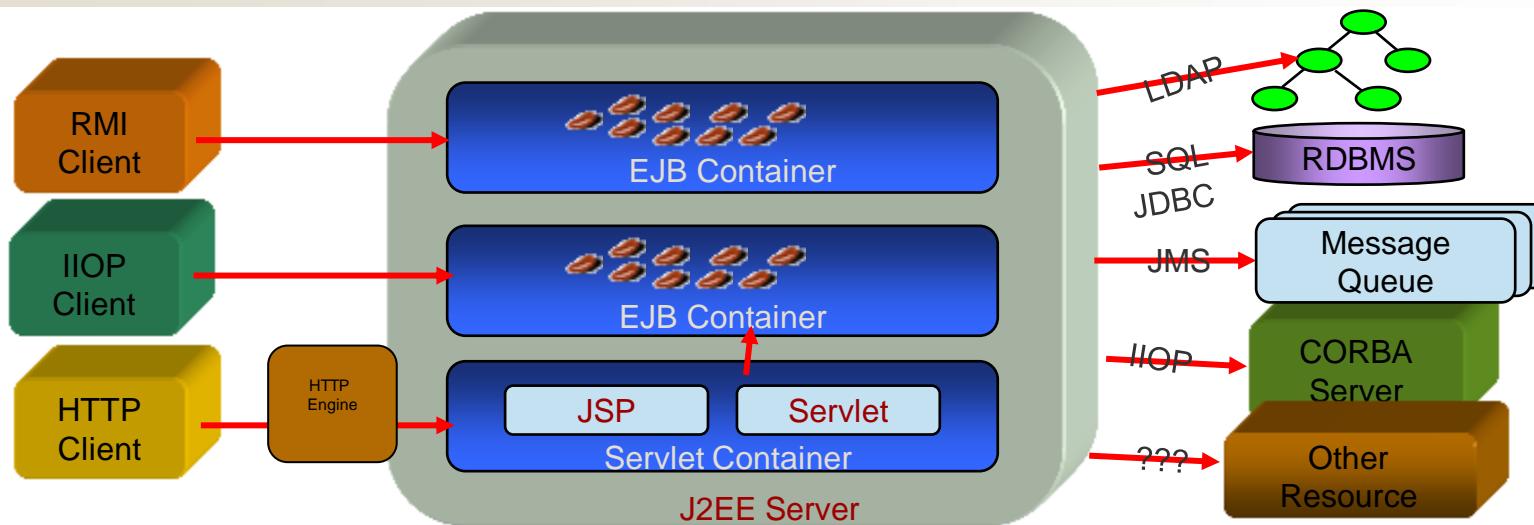
- Dobro: neovisnost o radnoj platformi, kao reakcija na događaj Beans komponenta može komunicirati i spajati se s ostalim Beans komponentama, Beans komponenta se može prilagođavati specijalnoj primjeni, dostupan izvorni kod.
- Loše: prepostavka virtualnog stroja usporava rad, otkriva se unutarnja struktura.

.NET

- Dobro: binarni i mrežni standard za komunikaciju između objekata, primjena u više programskih jezika (C#, VB, Javascript, VisualC++) , moguća je implementacija više globalno poznatih sučelja (prva metoda u sučelju je queryInterface() , vraća oznaku ako sučelje nije podržano).
- Loše: upravljanje memorijom, kompatibilnost (MSWindows).

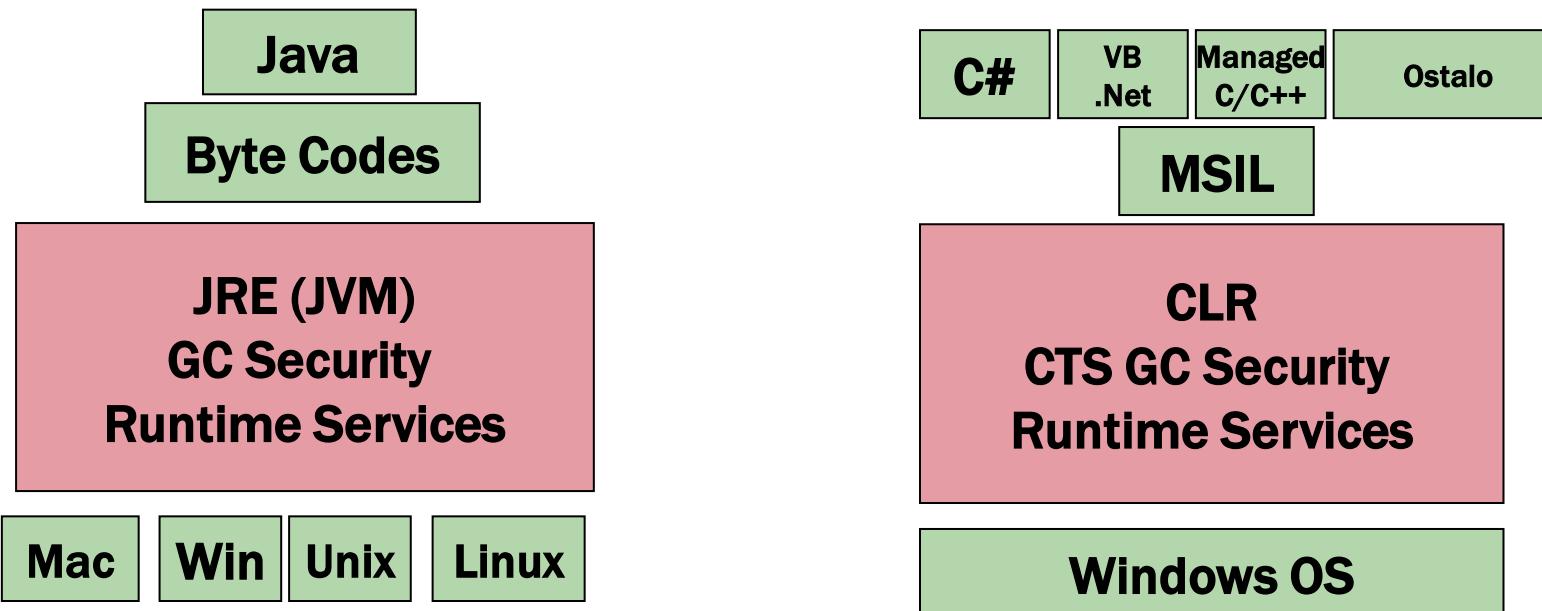


Primjer: Arhitektura Java i .Net



Usporedba JVM i CLR-a

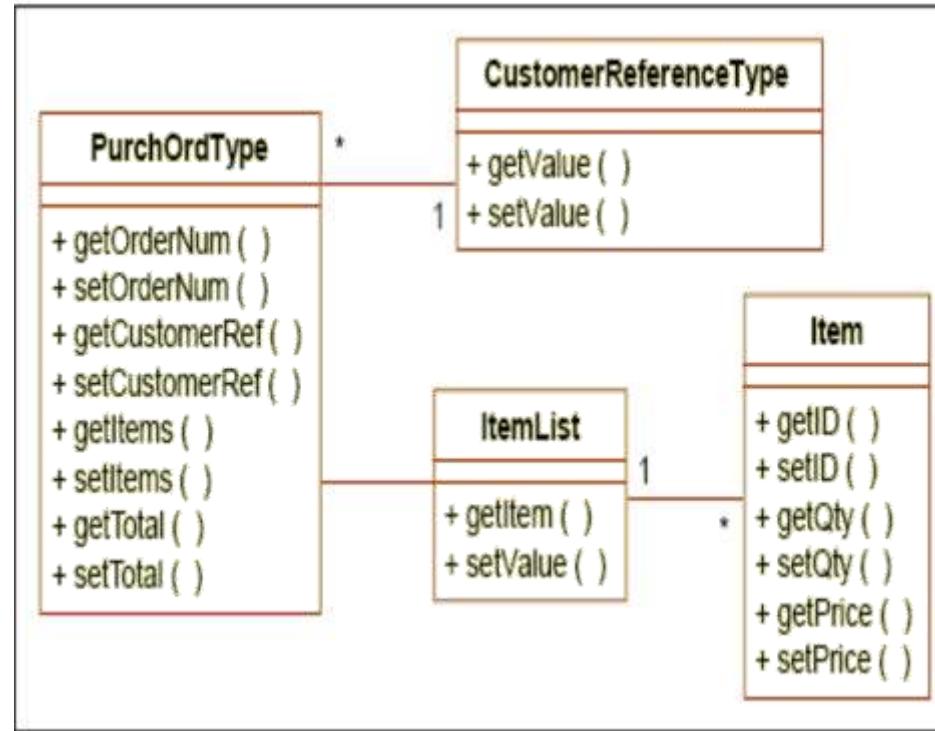
- Kako odabrat?■ netehnički parametri presudni!





Primjer komponentnog modela

- Komponentama su pridruženi entiteti i pravila (npr. obuhvaća poslovni model)

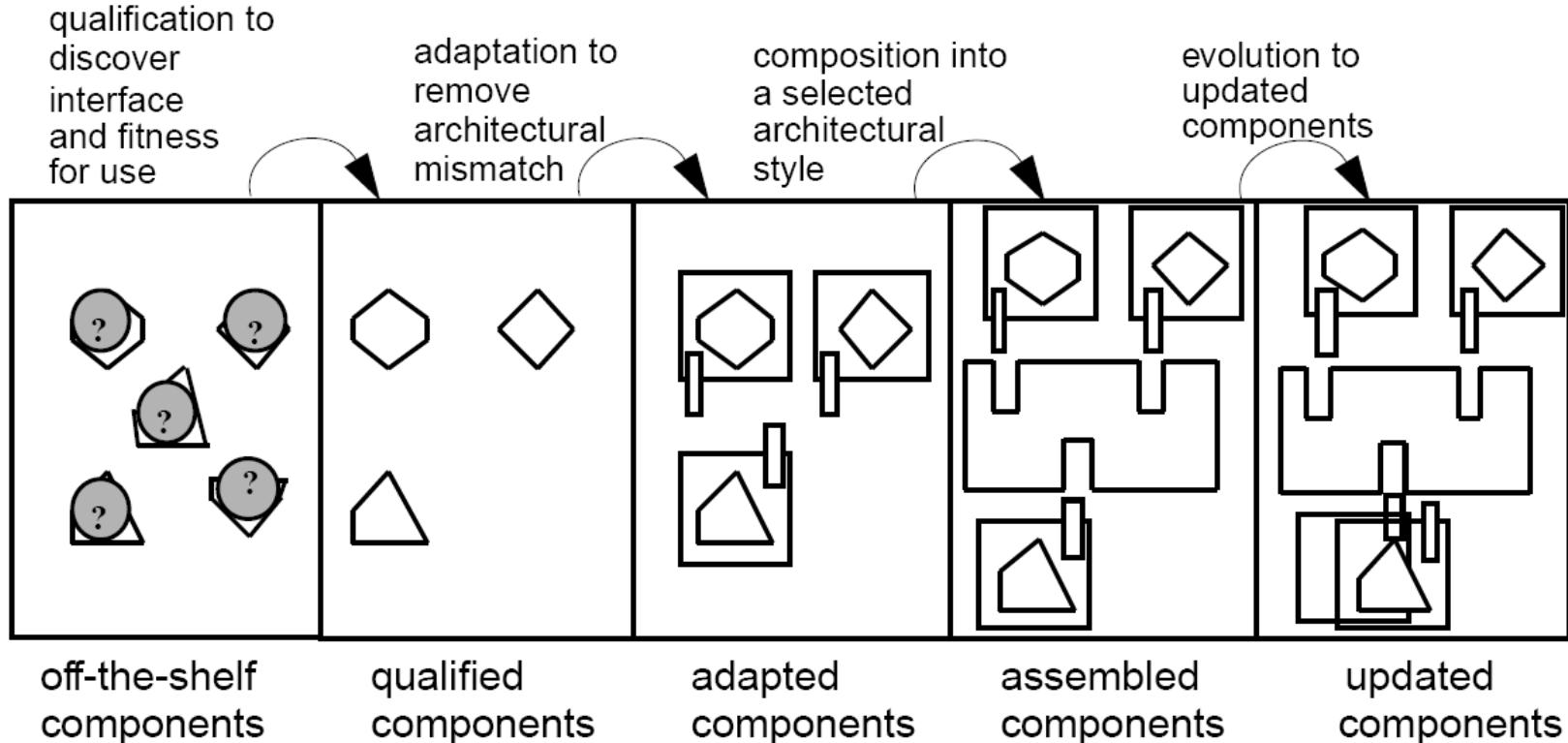


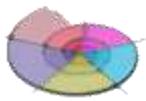
- Usluga je građena od niza komponenti koje osiguravaju poslovnu funkciju predstavljenu uslugom

Koraci oblikovanja zasnovanog na komponentama

activities/

transformations





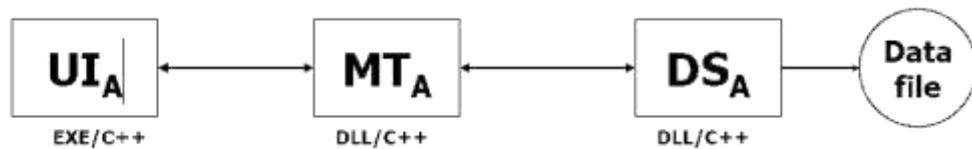
Primjer komponentnog razvoja



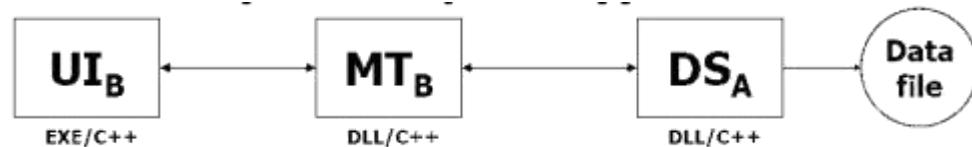
- Npr. razvoj programske potpore sustava naučivanja
- Komponente:
 - korisničko sučelje (UI)
 - poslovna logika (MT)
 - pristup podacima (DA)
- Komponente mogu imati inačice
- ⇒ inačice aplikacije tijekom razvoja
 - 1 stavka
 - više stavaka
 - GUI
 - jednostavan zapis podataka → Baza
 - klijet-poslužitelj CORBA,



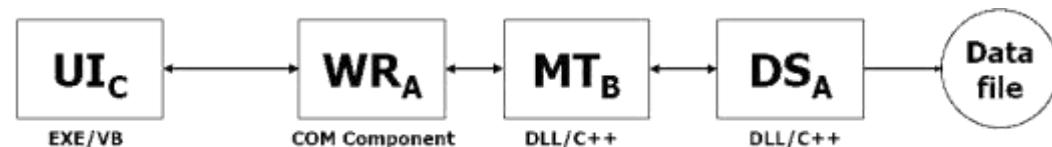
- 1 stavka



- Više stavaka



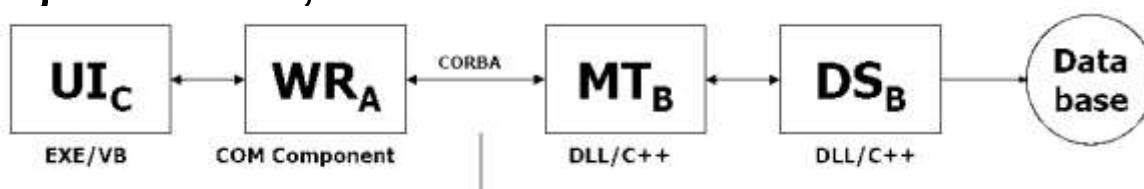
- Gui



- Jednostavan zapis podataka → Baza



- Klijet-poslužitelj CORBA,



Rekapitulacija principa ponovnog korištenja

- U fokusu arhitekture zasnovane na komponentama je njihovo ponovno i višestruko korištenje.
- Princip višestrukog korištenja u oblikovanju programske potpore manifestirao se tijekom vremena kroz:
 - ponovno korištenje konzistencije (programski jezici).
 - ponovno korištenje fragmenata rješenja (knjižnice).
 - ponovno korištenje dijelova arhitekture (arhitektturni obrasci – engl. *architectural patterns*).
 - ponovno korištenje arhitekture (radni okviri – engl. *frameworks*).
 - ponovno korištenje cjelokupne arhitekture sustava.



Objektno usmjerena arhitektura



- U procesu oblikovanja programske potpore teži se povećanju **razine apstrakcije**.
- Objektno usmjerena arhitektura povećava razinu apstrakcije uvođenjem koncepta razreda i njihovih različitih odnosa te dinamičkih akcija i reakcija njihovih instanci (objekata).
- Još višu razinu apstrakcije moguće je postići uporabom **radnih okvira** (engl. *frameworks*) koji predlažu skup apstraktnih razreda (vidi klijent-poslužitelj radni okvir) primјeren pojedinoj primjeni.
- Slijedeća, viša razina apstrakcije obuhvaća posredničku i uslužnu arhitekturu, tu uporabu gotovih komponenata.
- Visoka razina apstrakcije postiže se uporabom gotovih **programskih obrazaca** (engl. *Design patterns*).
- Mnogi drže da se najviša razina apstrakcije postiže na razini korisnika koji bi mogli oblikovati primjenski program.
 - već danas postoje radna okruženja za modeliranje na toj razini
 - engl. *Domain specific modeling*

Trendovi

■ Microsoft:

- staro: Visual Basic, C++, COM, DCOM, DLLs, ...
- novo: .NET, VB.NET, C#, ASP.NET
- prednost - interoperabilnost

■ Java:

- trend EJB, J2EE, JAWS , JAX,
- prednost: jasnoća i kvaliteta

■ CORBA:

- web i XML podrška
- trend





Sadašnje stanje



- U programskom inženjerstvu postoje programske utvrde (*engl. software fortress*)
 - programski sustavi namijenjeni općoj namjeni upravljanju od strane organiziranje grupe pojedinaca
 - rade u uskoj sprezi na pružanju jednoznačne i smislene funkcionalnosti u neprijateljskom okruženju
- Osnovni tipovi: utvrde poslovnih aplikacija, utvrde prezentacijskog sučelja, utvrde Web servisa, utvrde ugovornih odnosa, servisne utvrde, pravne utvrde,....
- Različiti neovisno razvijeni sustavi
- Složenost odnosa
- Tehnološki ratovi (Windows ↔ Linux, .NET ↔ Java, ...)
- Nepovjerenje
- Nered podataka
- Povjerenje u Internet – sigurnost
- Bojazan promjene naslijednih aplikacija (*engl. legacy application*)

Budućnost?

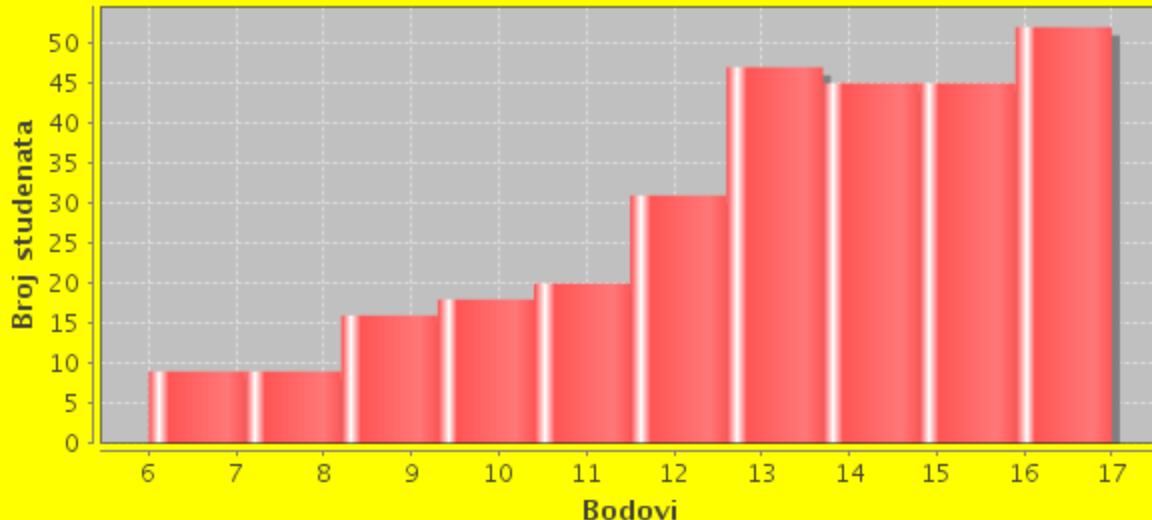
- Arhitektura programske podrške je više od strukture, ponašanja i pridruženih podataka
 - jednakо važni upravljački, ugovorni i financijski aspekti
- Napredne tehnike oblikovanja, povećanje udjela arhitekturnih informacija, kompleksniji gradivi blokovi ⇒ usmjerenost na arhitekturu
- Usmjereno na ljude
 - današnja razina arhitekture naglašava utjecaj ljudskog čimbenika
 - Ispravan rad sustava;
 - Izrada u okviru troškova;
 - Omogućavanje timskog rada;
 - Pomoći održavanju sustava;
 - Razumijevanje sustava za sve dionike.

Diskusija

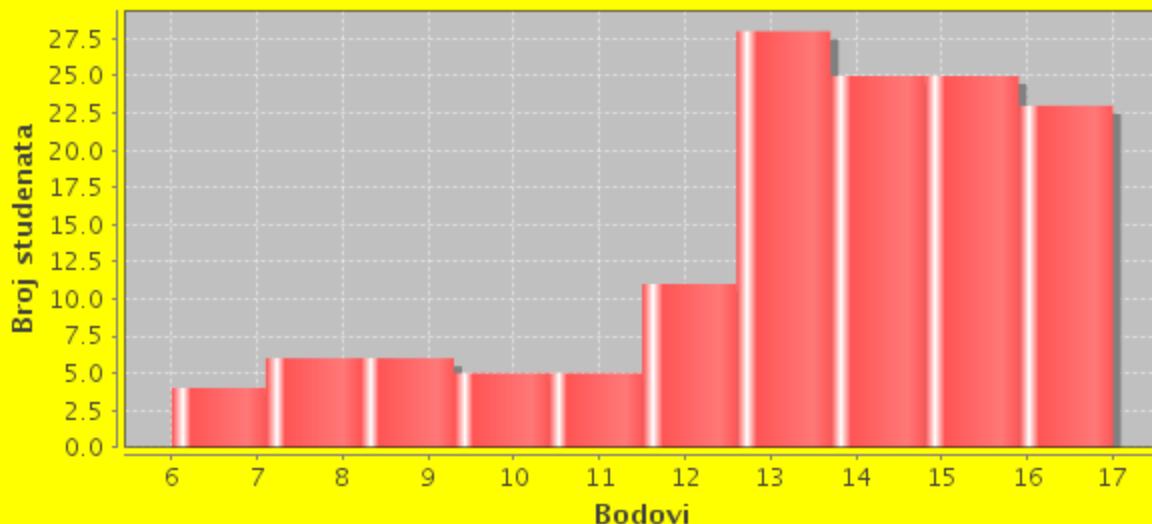
-
-
-
-
-

Teorijski dio

Raspodjela bodova



Raspodjela bodova





Problemski dio

- <1bod
 - 27
 - 38
- >3 bod
 - 52
 - 48

