Uvod u raspodijeljene sustave

1. Definicija, obilježja i vrste raspodijeljenih sustava

Definicija, obilježja i vrste raspodijeljenih sustava:

- "Skup neovisnih računala koji korisniku izgleda kao jedan cjeloviti sustav."
- "Sustav u kojem programske i sklopovske komponente umreženih računala komuniciraju usklađuju svoje aktivnosti isključivo razmjenom poruka."

Sloj raspodijeljenog sustava:

- Prikriva činjenicu da su procesi i sredstva raspodjeljen na više računala
- Pivezivanje i suradnju aplikacija, sustava i uređaja
- Između OS i aplikacijskih programa
- Iznad mrežnog ispod aplikacijskog

Razlozi za raspodijeljene sustave:

- Inherentna raspodijeljenost
    - Korisnika, informacija, sredstava
- Funkcionalno odvajanje:
    - Različite namjene, mogućnosti (korisnik-davatelj, proizvođač- potrošač)
- Opterećenje
- Pouzdanost i raspoloživost
- Cijena, troškovi, održavanje..

Vrste rasus:

1. Raspod računalni sustavi:
    - Grozd, nakupina (cluster)
    - Splet, cloud
2. Sveprisutni sustavi (pervasive system):
    - Nternet of things
    - Machine to machine (M2M)
3. Raspod informacijski sustavi:
    - Poslovni sustavi

- Transakcijski sustavi

4. Sustavi za pružanje informacijskih i komunikacijskih usluga:
    - Usluge, aplikacije…

Obilježja rasus:

- Paralelne aktivnosti:
    - autonomne komponente sustava istodobno izvode više aktivnost
- komunikacija razmjenom poruka:
    - komponente razmjenjuju podatke, ne vade ih iz memorije
- Dijeljenje sredstava:
    - Zajedničkim sredstvima pristupa više komponenata
- Nema globalnog stanja:
    - Niti jedan proces ne zna stanje svih ostalih
- Nema globalnog vremenskog takta:
    - Ograničenost vremenskog usklađivanja

2. Zahtjevi na rasus (otvorenost, transparentnost, skalabilnost i kvaliteta)

1. **Otvorenost:**
    - pruža usluge sukladno normiranim pravilima te definiranoj sintaksi i semantici
    - Norma:
        i. Široko prihvaćena u industriji (***de facto standard***) ili zastupana od tijela (***de jure standard***)
        ii. Dobro definirana
        iii. Javno dostupna
    - Pretpostavka za :
        i. Međudjelovanje
        ii. Prenosivost
        iii. proširljivost

2. **Transparentnost:**
    - prikrivanje odabranih značajki raspodijeljenog sustava
    - Transparentnost pristupa:
        i. Prikrivanje razlika u pristupu resursima (razliciti OS, DB…)

- Lokacijska transparentnost:
    i. Prikrivanje lokacije sredstva
- Migracijska transparentnost:
    i. Prikrivanje promjene lokacije: promjena lokacije sredstva ne utječe na način dostupa sredstvu
- Relokacijska transparentnost:
    i. Prikrivanje premještanja sredstva tijekom njegove uporabe

- Replikacijska transparentnost:
    i. Prikrivanje više istovrsnih sredstava ili više preslika nekog sredstva, što zahtijeva primjenu istog naziva za sve replike
- Konkurencijska transparentnost:
    i. Prikrivanje istodobne uporabe istog resursa od strane više korisnika: zajednička/dijeljena uporaba sredstva uz očuvanje konzistentnost
- Transparentnost na kvar:
    i. Prikrivanje kvara: otkrivanje kvara i obnavljanje sustava nakon kvara nije uočljivo korisnicima
    ii. Problem otkrivanja kvara: veliko opterećenje može se očitovati kao kvar (npr. nema odgovora u očekivanom vremenu)

3. **Skalabilnost:**
    - sposobnost razmjerne prilagodbe veličini (broj korisnika – količina sredstva), rasprostranjenosti (lokalno, regionalno, globalno, …) i načinu upravljanja (jedna ili više administrativnih domena)
    - kako bi se za više korisnika održale performanse treba:
        i. više dijelova (koliko ?)
        ii. prostorno raspod (gdje?)
        iii. koji komuniciraju asinkron (kako ?)
    - Tehnike koje omogućuju skalabilnost sustava:
        i. Prikrivanje kašnjenja u komunikaciji (radi nešto korisno dok čekaš odg)
        ii. Višestrukost `raspodijeljena baza podataka`
        iii. Replikacija (problem konzistentnost originala i kopije)

4. **Kvaliteta usluge:**
    - Quality of Service (skupni naziv za)
        i. Performanse
        ii. Pouzdanost/raspoloživost
        iii. Ukupni trošak vlasništva

```
response time - period od slanja zahtijeva do primitika

throughput - mjeri promet na serveru (bit/s)

availabilty (usluga dostupna u trenutku t)

Quality of Experience - zadovoljstvo korisnika
```

Oblikovanje raus

- Kakve funkcijske zahtjeve treba ostvariti – **ŠTO** sustav treba raditi?
- Kakve nefunkcijske zahtjeve treba ostvariti – **KAKO** sustav treba raditi (kakva se kvaliteta usluge zahtijeva)?
- Temelji li se sustav na otvorenim rješenjima?
- Kakav je stupanj transparentnosti potreban i kako utječe na složenost, performanse i troškove sustava?
- Kakva je skalabilnost sustava potrebna s motrišta veličine, rasprostranjenosti i upravljanja?

3. Arhitektura rasus-a

Programska arhitektura:

- Logička organizacija sustava, programske komponente, njihova org i interakcija

Sustavska arhitektura:

- Centralizirana ili decentralizirana

Kako predočiti rasus?

1. <u>Slojevita arhitektura</u>
    - U središtu aplikacijski sloj
    - Aplikacijski programi te usluge koje im pružaju niži slojevi
    - Funkcionalnost određuje aplikacijski sloj (najviši) a niži slojevi pružaju uslugu
    - Točka međudjelovanja dva susjedna sloja je Service Access Point (SAP) a samo međudjelovanje opisuje se Service Primitives (SP)
    - Service Primitives :
        i. *request* - korisnik upućuje davatelju
        ii. *indication* – izdaje ga davatelj usluge kako bi od korisnika zatražio izvršenje određene funkcije ili ga izvijestio da je primio zahtjev od drugog korisnika

iii. *response*- korisnik kao odgovor na indikaciju

iv. *confirm* – davatelj usluge kao odgovor na request

- Redosljed SP-a je prilikom uspostave asocijacije:
  1. Korisnik (klijent) nižem sloju upućuje *request* za spajanjem na drugi proces
  2. Niži sloj prosljeđuje drugom korisniku *indication* kojom traži to spajanje
  3. Taj korisnik vraća *response* ako može provesti zatraženo
  4. Niži sloj vraća *confirm* do prvog korisnika ako je uspješno provedeno spajanje

2. Objektna arhitektura: (komponentama)
   - Objekti su dijelovi sustava s dobro definiranim sučeljem mikrousluge --> dobro definirana sucelja - komunikacija i suradnje mikroservisa
   - Mehanizam komunikacije, usklađivanja i suradnje objekata
3. Arhitektura temeljena na podacima:
   - Procesi komuniciraju putem repozitorija
   - Jedna komponenta objavljuje podatke u repo a druga čita
4. Arhitektura temeljena na događajima:
   - Događaji prenose podatke
   - Jedna komponenta se preplati na neki podatak a kad druga komponenta objavi podatak (događaj) onda se taj podatak dostavlja pretplaćenoj komponenti (događaj)

4. Osnovni modeli raspodijeljene obrade

1. Klijent poslužitelj
   - Problemi sinkrone komunikacije
   - Ako je poslužitelj „zauzet“ onda svi ostali zahtjevi čekaju
   - Asimetrija dolaznih i odlaznih podatka moguća
2. Ravnopravni sudionici (peer):
   - Može obaviti funkciju poslužitelja i klijenta
   - P2P komunikacija tako da na app sloju povezuju u prekrivajuću mrežu nad stvarnom mrežnom topologijom
   - Svaki čvor plaća sudjelovanje tako da nudi dio sredstava ostalim čvorovima
   - *Decentralizirani rasus*
   - *Samoorganizirajuća mreža čvorova*

3. Pokretni kod i programski agenti:
    - **Process migration:**
        - Klijent raspolaže kodom međutim nedostaju mu sredstva za izvedbu, on šalje kod na server te se tamo izvrši kod (remote evaluation) i vraća mu se rezultat
    - **Code on demand:**
        - Klijent nema kod, već po potrebi šalje serveru zahtjev, server iz repozitorija dohvaća kod, šalje ga klijentu gdje se izvršava lokalno i potom se briše
        - Primjer java appletA
    - **Active message:**
        - Poruka sa kodom ide od čvora do čvora di bi se kod izveo na određenom
        - Analogija pokretnog objekta u objektnoj arhitekturi
    - **Programski agent:**
        - program koji obavlja neki posao za vlasnika a raspolaže svojstvima kao što su inteligencija, samostalnost, proaktivnost itd.
    - **Pokretni agent:**
        - Programski agent koji predstavlja korisnika i samostalno se kreće između čvorova u mreži
        - Smanjen mrežni promet jer se sve izvodi lokalno na poslužitelju/ klijentu
        - Moguće je na različitim čvorevima obavljati dio posla i na kraju dobiti rezutat obrade
        - Programska infrastruktura potrebna za izvedbu i izvršavanje agenata naziva se agentskom platformom (engl. agent platform) i mora se nalaziti n svakom čvoru koji udomljuje agente

5. Web kao case-study

WWW je stvoren i razvija se kao otvoreni sustav s transparentnim pristupom i konkurencijskom transparentnosti. Usluge pruža velikom broju korisnika predočenih klijentima sukladno normiranim pravilima te definiranoj sintaksi i semantici, uz očuvanje konzistentnosti sredstva.

- Transparentnost pristupa i konkurencijska transparentnost su ostvarene njegovim osnovnim postavkama. Sama zamisao „hiperteksta“ i normiranog jezika za označavanje (HTML) ključna je za transparentnost pristupa, dok je „konkurencija“ korisnika, tj. klijentskih zahtjeva, inherentna modelu klijent-poslužitelj

- Lokacijska transparentnost postiže se simboličkim sustavom imena gdje DNS vodi brigu o pretvorbi u IP adrese
- Migracijska transparentnost korisnik nema pojma je li promjenjena ip adresa dok nije promijenjen simbolički naziv
- Replikacijska transparentnost npr. Više posluzitelja iza simboličkog naziva gdje proxy usmjerava na pripadajući fizički poslužitelj

Procesi i komunikacija preza 2

Međuprocesna komunikacija (interprocess communication IPC) realizirana je prosljeđivanjem poruka na mrežnom sloju tako što se podaci pakiraju u oblik prikladan za prijenos mrežom.

2.1 Obilježja komunikacije

Pravila komunikacije među procesima definiraju se komunikacijskim protokolom. Prvo važno obilježje komunikacije opisuje vrstu komunikacijskog protokola koji može biti *konekcijski* ili *beskonekcijski*.

Konekcijski protokol kad procesi prvo izmjene kontrolne poruke (uspostave konekciju logično) i pritom razmjene pravila, parametre komunikacije i tek onda kreće prijenos podataka.(TCP)

Beskonekcijski protokol ako se ne ne izmjenjuju kontrolne poruke prije slanja podataka. (UDP)

Perzistentna komunikacija:

- garantira isporuku poruke premda primatelj nije aktivan u trenutku nastanka i slanja poruke.
- Pretpostavlja postojanje posrednika koji pohranjuje poruku do njene isporuke.

Tranzijentna komunikacija:

- Garantira isporuku poruke samo ako su pošiljatelj i primatelj istovremeno aktivni

Sinkrona komunikacija:

- Pošiljatelj je blokiran dok ne primi potvrdu od primatelja
- Može biti blokiran sve do primitka potvrde o tome da je:
    1. sloj raspodijeljenog sustava primio zahtjev te se dalje brine o isporuci poruke
    2. primatelj uspješno primio zahtjev nakon čega će započeti obradu
    3. primatelj uspješno primio i obradio poruku te uz potvrdu šalje i rezultate
- *pull* načelo (dovlači odgovor s servera)
- zbog potencijalno dugog čekanja na odgovor definiramo *listener* i tada imamo *push* načelo

Asinkrona komunikacija:

- pretpostavlja da nakon slanja poruke pošiljatelj nastavlja obradu bez čekanja na povratnu informaciju, te se poruka jednostavno pohranjuje u izlazni spremnik pošiljatelja nakon čega operacijski sustav pošiljateljskog računala vodi računa o njenoj isporuci.

2.2 Obilježja procesa

Def. Proces je program u izvođenju na jednom od fizičkih ili virtualnih procesora računala.

Dva osnovna obilježja procesa:

1. vremenska (ne)ovisnost
    - vremenski ovisni procesi moraju istovremeno biti aktivni za realizaciju komunikacije
    - primjer SMS

2. ovisnost o referenci sugovornika
    - anonimnost komunikaicije
    - Proces je ovisan ako mora znati jedinstveni identifikator, tj.adresu udaljenog procesa s kojim želi komunicirat (client-server)
    - Primjer neovisnog procesa sustavi objava-preplata li dijeljenom podatkovnom prostoru

Tablica 2. Obilježja raspodijeljenih procesa

1.	vremenska ovisnost	vremenska neovisnost
2.	ovisnost o referenci "sugovornika"	neovisnost o referenci "sugovornika"
3.	iterativni poslužitelj	konkurentni poslužitelj
4.	*stateless* poslužitelj	*stateful* poslužitelj

Poslužiteljski procesi se u raspodijeljenim sustavima najčešće izvode višedretveno. Dretve se izvode konkurentno unutar jednog procesa te dijele isti adresni prostor. Višedretvenost rješava probleme koje izazivaju metode sa svojstvom blokiranja procesa (npr. Accept za poslužiteljski socket TCP-a).

Uobičajene zadaće klijenta:

- GUI
- Otvaranje mrežne konekcije prema serveru
- Primanje podatak

Uobičajene zadaće servera:

- Primanje zahtjeva
- Obrada podataka i rad sa bazama podataka

Poslužitelj:

- Iterativni:
    - Istovremeno samo jedan zahtjev
    - Neadekvatan za rasus
- Konkurentni:
    - Istodobno više zahtjeva
    - Složeniji, višedretvenost
- Stateless:
    - Ne pamti stanje klijentskih zahtjeva
    - Npr. Web poslužitelj (svaki HTTP je zasebna jedinica i neovisna o drugima)
- Stateful:
    - Obrnuto od stateless
    - Npr. Poslužitelj za pohranjivanje datoteka koji pohranjuje kopiju datoteke i radi tablicu kojom povezuje klijenta, kopiju i original datetoeku

3. Sloj raspodijeljenog sustava za komunikaciju među procesima

Programski posrednički sloj ili međuoprema (engl. *middleware*) je programski sustav na aplikacijskom sloju koji se nalazi između transportnog sloja i aplikacije. Pojednostavljenje oblikovanja razvoja aplikacija te se izdaje u obliku library-a.

Međuoprema za komunikaciju udaljenih procesa

- Nalazi se na sloju rasus
- Implementira protokole među raspod procesima

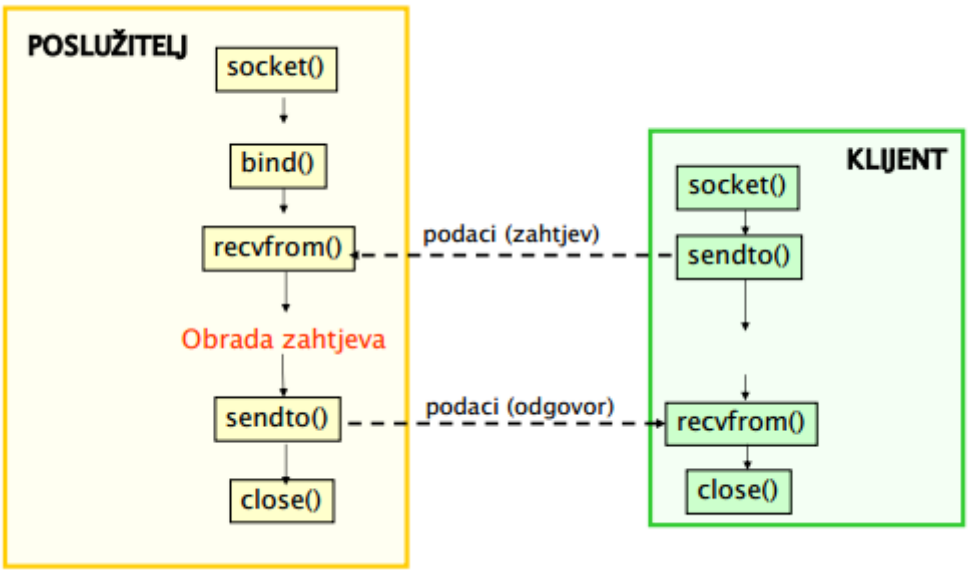
3.1 Komunikacija korištenjem priključnica

Priključnica (socket):

- Pristupna točka koja šalje podatke u mrežu i iz koje čita primljene podatke
- Viši nivo apstrakcije od komunikacijske točke koju OS koristi za pristup funkcijama transportnog sloja
- Veže se uz adresu i port

User Datagram Protocol (UDP):

- Nespojna usluga (bezkonekcijski protokol)
- Prenosi datagrame
- Poslani datagrami spremaju se privremeno u izlazni spremnik povezan s vratima iz izvorišne adrese na klijentu (ne blokirajuća operacija slanja jer se dalje odvija bez čekanja na potvrdu)
- Kod primanja moguće blokirajuće i neblokirajuće (blokirajuće kad je spremnik iz kojeg se čita prazan)
- Vremenska ovisnost procesa (server mora bit aktivan)
- Tranzijentna komunikacija
- Asinkrona komunikacija



Kako implementirati poslužitelja koji koristi UDP *socket*?

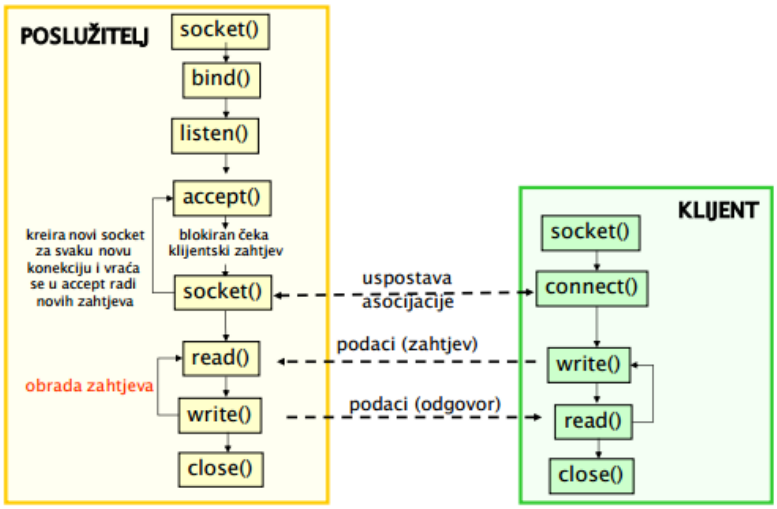
1. Kreirati *socket* poslužitelja
```
DatagramSocket serverSocket;
serverSocket = new DatagramSocket( PORT );
```
2. Kreirati paket (prazan, priprema za primanje)
```
byte[] rcvBuf = new byte[256];
DatagramPacket packet =
    new DatagramPacket(rcvBuf,rcvBuf.length);
```
3. Čekati korisnički paket (blokira proces do klijentskog zahtjeva!)
```
serverSocket.receive( packet );
```
4. Obraditi pristigli paketa i po potrebi poslati odgovor klijentu
5. Zatvoriti poslužiteljski *socket*
```
serverSocket.close();
```

Kako implementirati klijenta koji koristi UDP *socket*?

1. Kreirati *socket*
```
DatagramSocket clientSocket;
clientSocket = new DatagramSocket();
```
2. Kreirati paket (prazan, priprema za primanje paketa iz mreže)
```
byte[] sendBuf = new byte[256];
DatagramPacket packet =
new DatagramPacket(sendBuf, sendBuf.length, destAddress, destPort);
```
3. Slanje paketa
```
clientSocket.send( packet );
```
4. Po potrebi obrada i čekanje odgovora
5. Zatvoriti *socket*
```
clientSocket.close();
```

Transmission Control Protocol (TCP):

- Konekcijski protokol
- Pouzdan prijenos paketa uz kontrolu retransmisije
- Dva socketa (klijent + server)
- Client-server
- Vremenska ovisnost
- Klijent mora znati ID servera
- Tranzijentna komunikacija
- Sinkrona komunikacija
- Pull načelo



- *Socket:* kreira komunikacijsku točku, operacijski sustav rezervira sredstva koja će omogućiti slanje i primanje podataka pomoću odabranog transportnog protokola
- *Bind:* povezuje adresu sa socketom
- *Listen:* omogućuje os rezerviranje sredstava za max broj konekcija
- *Accept:* poslužitelj prima zahtjev za otvaranjem konekcije od strane klijenta (connect). Poslužitelj stvara novi socket koji se koristi za komunikaciju s klijentom. Originalni socket se koristi za primanje novih zahtjeva. Blokirajuća metoda !!

Kako implementirati poslužitelja koji koristi TCP *socket*?

1. Kreirati *socket* poslužitelja
```
ServerSocket serverSocket;
serverSocket = new ServerSocket( PORT );
```
2. Čekati korisnički zahtjev (proces je blokiran do klijentskog zahtjeva) i kreirati kopiju originalnog *socketa*
```
Socket copySocket = serverSocket.accept();
```
3. Kreirati I/O stream za komunikaciju s klijentom
```
DataInputStream is= new DataInputStream(copySocket.getInputStream());
DataOutputStream os=new DataOutputStream(copySocket.getOutputStream());
```
4. Komunikacija s klijentom
5. Zatvoriti kopiju *socketa*
```
copySocket.close();
```
6. Zatvoriti poslužiteljski *socket*
```
serverSocket.close();
```

Kako implementirati klijenta koji koristi TCP *socket*?

1. Kreirati klijentski *socket*
```
clientSocket = new Socket( address, port );
```
2. Kreirati I/O stream za komunikaciju s poslužiteljem
```
is = new DataInputStream(clientSocket.getInputStream() );
os = new DataOutputStream( clientSocket.getOutputStream() );
```
3. Komunikacija s poslužiteljem
```
//Receive data from server:
    String line = is.readLine();
//Send data to server:
    os.writeBytes("Hello\n");
```
4. Zatvoriti *socket*
```
clientSocket.close();
```

Višedretveni poslužitelj:

- Za implementaciju konkurentnog poslužitelja, potrebno je kreirati novu dretvu za svaki klijentski zahtjev tako da postoji jedna po konekciji
- Model koordinator/radna dretva (engl. dispatcher/worker model)
    - Koordinator prima zahtjeve i komunicira s klijentom
    - Ograničiti broj dretvi na strani poslužitelja *thread pool* (višak se ili odbauje ili sprema u red čekanja *backlog*)
    - Glavna ddretva samo prima konekcije i svaku (do MAX_KON) pokreće u svojoj dretvi

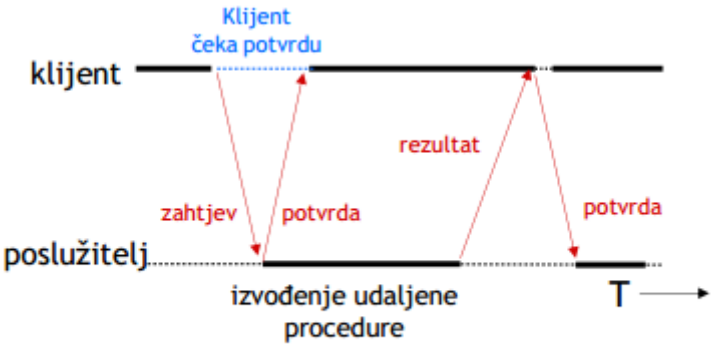
3.2 Poziv udaljene procedure/metode

Remote Prodcedure Call (RPC)

- omogućuje pozivanje i izvođenje procedura na udaljenom računalu.
- RPC je transparentan jer pozivajuća procedura nije svjesna da se proces izvodi na udaljenom računalu
- Koristi se posebna komponenta *stub* (klijent je blokiran za vrijeme izvođenja
- Koraci poziva:
    1. Klijent poziva udaljenu proceduru
    2. Stub pakira parametre i ID procedure i poziva OS na klijentu
    3. OS klijenta šalje poruku na server
    4. OS servera predaje poruku stubu servera
    5. Stub servera raspakira pakete, izvadi parametre itd. i poziva proceduru
    6. Procedura vraća rezultat serverskom stubu
    7. Stub servera pakira rez u poruku i poziva OS
    8. Os servera šalje poruku OS-u klijenta
    9. OS klijenta predaje poruku stubu
    10. Stub raspakira rez i predaje klijentskom procesu
- Postupak „pakiranja“ parametara ili rezultata u poruku je *marshalling*
- Postupak „raspakiravanja“ odnosno čitanja podataka iz poruke je *unmarshalling*
- Ukoliko se prenose objekti tada se cijeli objekt zapiše u poruku (call by value) a ne preko referenca jer reference imaju smisla samo u adresnom prostoru procesa

Asinrkoni RPC:

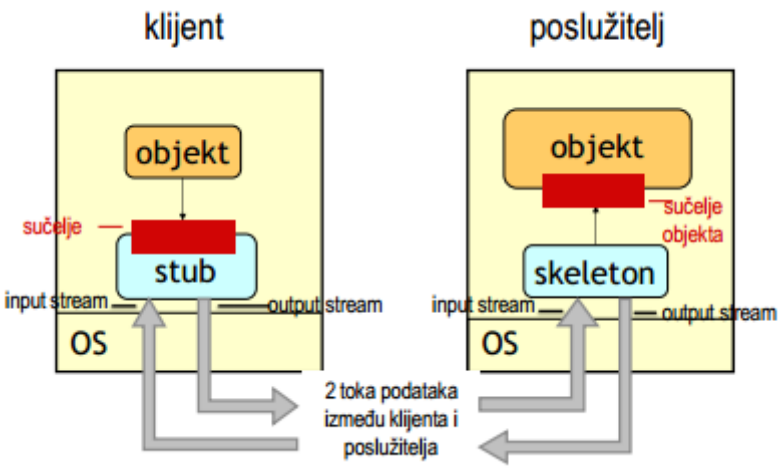
- Kako bi se smanjilo beskorisno čekanje



Slika 3.12. Odgođeni sinkroni RPC

Poziv udaljene metode (engl. Remote Method Invocation, RMI)

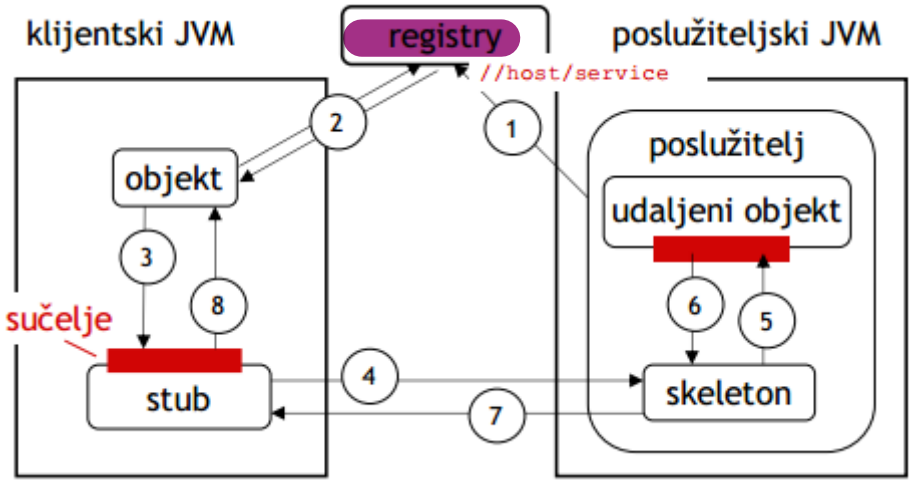
- RPC za objektne jezike jer se poziva metoda udaljenog objekta
- Sustav mora nuditi uslugu za registriranje i pronalazak udaljenih objekata (directory service)
- Ideja == RPC (samo s objektima)
- Skeleton == stub na serveru
- Stub u ovom slučaju implementira sučelje udaljnog objekta ali samo kao posrednik koji poziv metode i parametre pakira u oblik prikladan za slanje mrežom i šalje do skeletona koji dalje preusmjerava do objekta



-
- Za razliku od RPC ovdje mogu postojati reference na objekte i prenose se kao parametar udaljene metode (call by reference)
- Obiježja komunikacije poziva udaljene procedure/metode:
  - Klijent server
  - Vremenska ovisnost
  - Klijent mora znat id servera
  - Tranzijentna komunikacija
  - Sinkrona komunikacija (osim kod asinkronog RPC)
  - Pull načelo pokretanja komunikacije

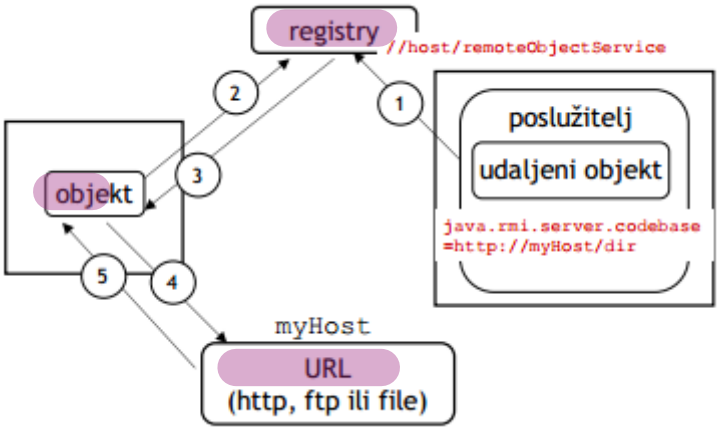
Java RMI (Remote Method Invocation)

- Transparentnost kao najvažnije svojstvo
- Referenca na udaljeni objekt istovjetna referenci na lokalni objekt s tom razlikom da svi udaljeni objekti moraju implementirati sučelje *java.rmi.Remote*
- Sučelje udaljenog objekta implementira stub (proxy) u adresnom prostoru klijentskog računala, a klase stub i skeleton se generiraju iz implementacije, a ne iz sučelja udaljenog objekta.
- Pri pozivu metode svi se parametri (primitivi i objekti) moraju serijalizirati te moraju implementirati sučelje *Seriazable*
- U slučaju udaljenog objekta prenosi se njegova referenca koja je jedinstvena u raspodijeljenom sustavu, a sadrži adresu računala, broj vrata i identifikator udaljenog objekta



- Koraci komunikacije:
  1. Poslužiteljski objekt se registrira pod odabranim imenom (npr. //host/service).
  2. Klijent od registrya traži referencu na udaljeni objekt pomoću registriranog imena.
  3. Klijent poziva metodu stuba koji je dostupan na klijentskom računalu.
  4. Stub serijalizira parametre i šalje ih skeletonu.
  5. Skeleton deserijalizira parametre i poziva metodu udaljenog objekta.
  6. Udaljeni objekt vraća rezultat izvođenja metode skeletonu.
  7. Skeleton serijalizira rezultat i šalje ga stubu.
  8. Stub deserijalizira rezultat i dostavlja ga klijentu.

- S obzirom da se stub može mijenjati bog promjene implementacije udaljenog objekta, klijent mora pribavi najaktualniji stub u svoj adresni prostor



-
- Redoslijed akcija za dinamičko učitavanje klase stuba na stranu klijenta:
  1. Poslužitelj definira codebase udaljenog objekta i registrira taj udaljeni objekt pod odabranim imenom. (Codebase definira URL s kojega se Javine klase mogu učitavati u JVM.)
  2. Klijent od registrya traži referencu na udaljeni objekt koristeći registrirano ime.
  3. Registry vraća podatke o klasi stuba. Ako se klasa stuba može pronaći na klijentskoj strani učitava se lokalna verzija klase. U suprotnom će klijent učitati klasu koristeći definirani codebase.
  4. Klijent traži klasu stuba koristeći codebase.
  5. Klasa stuba se dostavlja klijentu. Klijent može pozivati metode udaljenog objekta koristeći primljeni stub

3.3 Komunikacija porukama

preza 4

Komunikacija porukama spada u važnu skupinu programske infrastrukture za komunikaciju raspodijeljenih procesa poznatu pod imenom message-queuing systems ili Message-Oriented Middleware (MOM). Omogućuje perzistentnu i asinkronu komunikaciju preko posrednika koji pohranjuje poruke, a da pri tom izvor i odredište ne moraju biti istovremeno aktivni.

Osnovna ideja:

- Preko posrednika tj. postoji rep na primatelju
- Pošiljatelju se garantira isporuka na rep ali ne i do primatelja
- Na rep se dodaje metodom *put* a čita sa *get i poll*
- *Get* briše kopiju iz repa ali blokira primatelja ako je rep prazan
- *Poll* provjerava stanje repa te ovisno o tome isporuči ili ne poruku
- Pošiljatelj i primatelj su vremenski neovisni
- Svaki rep ima jedinstveno ime (tzv. Adresa repa) neovisno o transportnoj adresi te je potrebna usluga koja povezuje ime s adresom (DNS analogija)
- Na posredniku postoi queue manager proces koji upravlja skupom repova

Obilježja komunikacije:

- Vremenska neovisnost
- Sender mora znati id odredište tj. repa
- Perzistentna komunikacija
- Asinkrona komunikacija
- Pull načelo (primatelj provjerava postoji li poruka za njega)
- Moguće i push načelo preko notify()

3.4 Komunikacija na načelu objavi-preplati

- Omogućuje asinkronu komunikaciju između publishera i subscribersa
- Pretplata se može zamisliti kao predložak (engl. template) obavijesti
- Razmjena se odvija preko posrednika, *brokera*
- Povoljno utječe na skalabilnost sustava
- Obavijesti (notifications) , preplate (subscriptions) i odjave preplta(unsupscription)
- Prikladno za sustave di se promjena vrijednosti odnosno događaj ne može predvidjeti
- Matching property:
    - Preplata prekriva obavijest ako zadovoljava sve uvjete definirane preplatom
    - Ako preplata prekriva obavijest onda se mora isporučiti
- Najraširenije vrste preplata:
    - Preplata na kanal
        - omogućuje tematsko grupiranje obavijesti i klasificira svaku obavijest
        - Kanal se može smatrati logičkom poveznicom među objavljivačima i pretplatnicima
        - Kanali se mogu organizirati hijerarhijski (vrijeme u svijetu, europi, hrvatskoj..)
    - Preplata na sadržaj
        - Složenije i fleksibilnije preplate
        - Definiraju se uvjeti nad atributima
        - Obavijest je najčešće točka u višedimenzionalnom prostoru(senzor…)
        - Preplata je potprostor višedimenzionalnog prostora

Arhitektura usluge objavi-pretplati:

- Centralizirano :
  - 1 posrednik sa cjelokupnim skupom pretplatnika i publishera
  - Posrednik ima sve informacije o svemu u sustavu
  - Najteže smisliti algoritam
  - Glavni nedostatak je skalabilnost i nepouzdanost rješenja jer ako ispadne taj jedan čvor ode sve u k
- Raspodijeljeno
  - Mreža posrednika od kojih svaki obrađuje dio klijenata te je zadužen za podskup pretplatnika koji su direktno na nj spojeni
  - Skalabilnost na više poslužitelja
  - Direktno se poslužuju lokalni objavljivači i pretplatnici te se usmjeravaju obavijesti drugim posrednicima
  - Održava se tablica usmjeravanja:
    - Svaka poruka sadrži zaglavlje s podacima o izvoru i odredištu poruke te prenosi obavijest, pretplatu ili odjavu pretplate
    - Kontrolne poruke (preplate i odjave) mijenjaju tablicu
  - Osnovne metode usmjeravanja:
    - **Preplavljivanje**:
      - Svaka primljena poruka se prosljeđuje svima
    - **Filtriranje poruka:**
      - Samo zainteresiranima

Obilježja modela objavi-pretplati su sljedeća:

1. vremenska neovisnost: objavljivači i pretplatnici ne moraju istovremeno biti aktivni, posrednik pohranjuje poruku

2. objavljivač ne mora znati identifikator pretplatnika (anonimnost), o tome se brine posrednik

3. komunikacija je perzistentna

4. asinkrona komunikacija: objavljivač šalje poruku i nastavlja obradu neovisno o odgovoru od strane odredišta

5. pokretanje komunikacije na načelu push: objavljivač šalje poruku posredniku koji je prosljeđuje pretplatnicima bez prethodnog eksplicitnog zahtjeva

6. personalizacija primljenog sadržaja: filtriranje objavljenih poruka prema pretplatama

7. proširivost sustava: dodavanje novog objavljivača ili pretplatnika ne utječe na ostale strane u komunikaciji

8. skalabilnost: raspodijeljena arhitektura

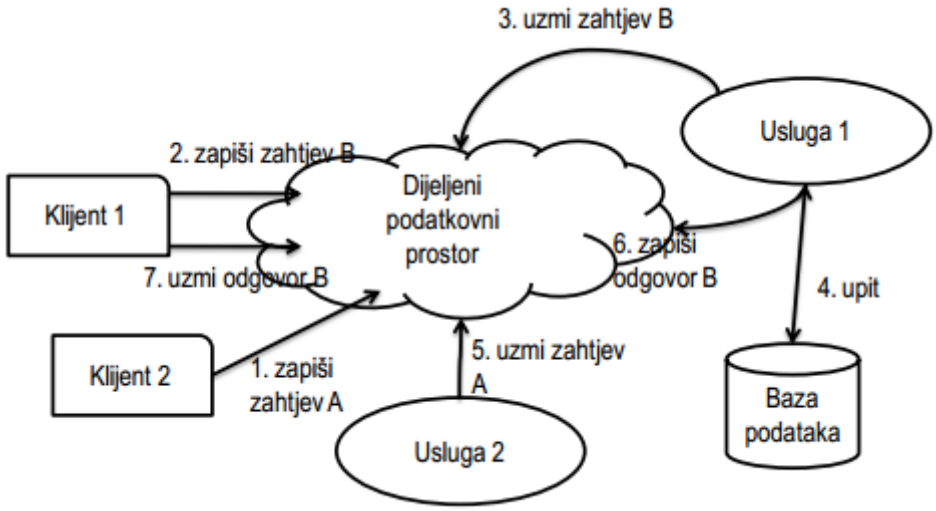
3.5 Dijeljeni podatkovni prostor

Temelji se na dijeljenoj perzistentnoj memoriji u koju se pohranjuju podaci (tzv. Tuple).

Slično kao i objavi-preplati iz memorije se čitaju podaci koji odgovaraju predlošku s razlikom što odredišta eksplicitno čitaju podatke a ne aktivna isporuka.

Operacije podržane:

- Write(t) dodaj tuple t
- Read(s) vraća tuple t koji odgovara template s
- Take(s) vraća tuple t koji odgovara template s i briše ga iz prostora



Obilježja komunikacije pomoću dijeljenog podatkovnog prostora su:

1. vremenska neovisnost: procesi ne moraju istovremeno biti aktivni radi komunikacije, dijeljeni prostor pohranjuje poruku

2. anonimna komunikacija (temelji se na sadržaju podataka)

3. komunikacija je perzistentna

4. asinkrona komunikacija: proces dodaje podatak u podatkovni prostor i nastavlja obradu

5. pokretanje komunikacije na načelu pull: proces eksplicitno šalje zahtjev za čitanje podatka iz dijeljenog podatkovnog prostora

4. Arhitekture web- aplikacija

preza 3

4.1 HTTP protokol

- Zahtjev i odgovor
- Port 80
- GET, PUT, DELETE, POST, HEAD i OPTION - znaš
- TRACE- dijagnostika
- CONNECT- za buduću uporabu?
- Zaglavlje prazan red tijelo
- TCP konekcija
- 1xx- Informative , 2xx- Success, 3xx- Redirecting, 4xx Client error, 5xx – Server error

4.2.1 HTML 5

- Najvažnija svojstva HTML-a 5 su:
    - Manje koda u zaglavlju
    - Više semantike u oznakama (header, nav, section…)
    - Oznake za multimediju (video, audio)
    - Geolokacija u JavaScriptu
    - Nove vrste podataka u formama
    - Spremnik podataka za offline rad
    - Drag
    - ...

4.3 CSS

Za uključivanje stila koristi se oznaka *link* unutar *head*. U oznaci imamo 3 atributa:

1. *Rel*- označava relaciju između HTML-ovog dokumenta i povezanog dokumenta, za stilove se koristi vrijednost *stylesheet*
2. *Type*- specificira vrstu dokumenta prema standardu MIME, za stilove se koristi text/cs
3. *Href* – specificira lokaciju pomoću URL-a

Prilikom dohvata resursa, najprije se učita HTML i zatim prkeo link-a novim GET zahtjevom i CSS. Tek nakon što se i on dohvati , korisniku se prikazuje stranica u pregledniku.

- /* komentar ide ovdje*/
- Selektor mogu odabrati razne oznake:
    - Oznake:
        - h1 {…} pravilo vrijedi za sve h1
    - atribut id:
        - #prozor {…} svi atributi sa id vrijednošću prozor
    - Atribut class:
        - .center {….}

4.4 Priručna spremišta (cache)

U rasus se koristi za spremanje kopije resursa na mjestu povoljnijem za njegovo korištenje (bliže..).

Ideja:

- Kod prvog dohvaćanja resursa, spremi ga u cache
- Kod svakog slijedećeg zahtjeva za resursom vraćaj iz memorije ne preko mreže
- Dobrobit za klijenta, server i mrežu

Smještaj spremišta može biti:

- Na klijentu:
  - web-preglednik sprema sadržaj na disk klijentskog računala
- na strani servera:
  - kopija izvornog resusa (npr. HTML-ov dokument) ili izračunati rezultat (npr. odgovor web-tražilice na neki popularni upit) se spremi za kasniju dostavu klijentu. Ako neki klijent zatraži isti resurs onda mu pružatelj usluge može odmah proslijediti traženi sadržaj i ne mora ponovno izvršavati kôd koji generira rezultat. Na taj se način smanjuje opterećenje izvornog poslužitelja pružatelja usluge
- u mreži:
  - poslužitelj posrednik (engl. caching proxy)

Postoje 2 modela za upravljanje spremištem:

1. **Model roka trajanja**
2. **Model validacije (valjanosti dokumenta)**

4.4.1 Model roka trajanja

- Server definira koliko je resurs valjan/ svjež
- Dobar kada znamo kad će se i koliko često resurs mijenjati ili kada se rijetko mijenja
- Zaglavlje *Expires* u HTTP (Expires: Sat, 15 Oct 2011 11:49:28 GMT)
  - problem ako satovi nisu sinkronizirani na serveru i klijentu
  - *Cache-Control* zaglavlje (max age=6497)

4.4.2 Model validacije

- Jeli spremljena inačica i dalje svježa
- *Last-Modified* i *Etag*
  - Etag koristi hash (MD5) (klinet šalje zahtjev sa zaglavljem If-None-Match: hash)

4.4.3 Posrednički poslužitelj

- filtriranje informacija - npr. blokiranje nepoželjnog sadržaja za djecu ili članove organizacije;
- privremeno spremište - ubrzavanje posluživanja tj. manje korištenje mreže između
- posredničkog poslužitelja i poslužitelja usluga
- anonimiziranje klijenta – poslužitelj usluga vidi klijenta kao posrednički poslužitelj, a ne krajnjeg klijenta;
- bilježenje korištenja mreže - npr. u organizaciji se može bilježiti koji korisnik/skupina korisnika je koliko koristila mrežu;
- provjera sadržaja - npr. provjera na viruse, analiza informacija koje se šalju iz organizacije itd

Prema smještaju:

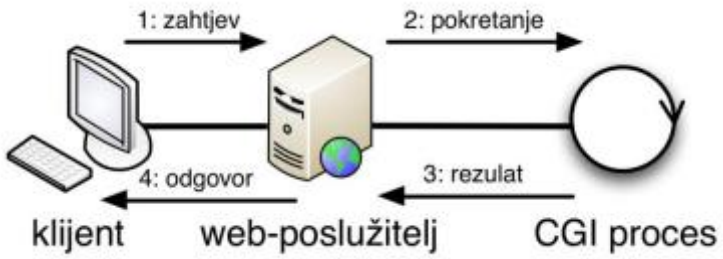
- lokalnoj mreži klijenta (forward proxy)
  - smanjuje promet između lokalne i vanjske mreže
- javnoj mreži (open proxy)
  - isključivo za anonimiziranje klijenata
- lokalnoj mreži na poslužiteljskoj strani (reverse proxy):
  - enkripcija ili komprimiranje podataka – rasterećuje se poslužitelj usluge (npr. webposlužitelj);
  - uravnoteženje opterećenja – posrednički poslužitelj bira na koji će poslužitelj usluga usmjeriti zahtjev klijenta i na taj način više poslužitelja usluga mogu paralelno obrađivati zahtjeve klijenata
  - privremeno spremanje izračunatih podataka – posrednički poslužitelj može svježe resurse spremiti u svoje priručno spremište i njih slati kao odgovor te na taj način rasteretiti poslužitelja usluge;
  - sigurnosne postavke – može dozvoliti pristup samo nekim uslugama i na taj način poslužitelj usluge ne mora biti toliko siguran jer ga štiti posrednički poslužitelj koji može osiguravati SSL komunikaciju između klijenta i posredničkog poslužitelja, a komunikacija između posredničkog poslužitelja i poslužitelja usluga ne treba biti zaštićena.

4.5 Web-aplikacije

Def. *„Web-aplikacije su ostvarene na web-poslužiteljima i koriste alate kao što su baze podataka, JavaScript (Ajax ili Silverlight), PHP (ili ASP.Net) kako bi se omogućile funkcionalnosti koje nisu standardne web-stranice ili web-obrasci."*

4.5.1 CGI (Common gateway Interface)

Jednostavno sučelje za pokretanje eksternih programa iz web-poslužitelja na platformski i programski neovisan način. Kod svakog zahtjeva se pokreće novi proces, a podaci između poslužitelja i procesa šalju se preko varijabli okoline i tokova podataka. Nakon svake obrade proces se gasi.

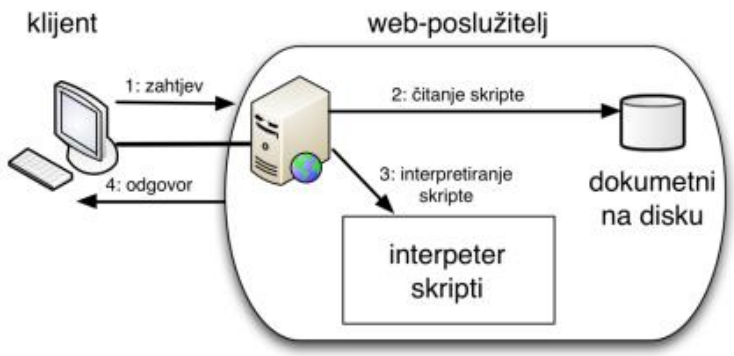


Nedostatak je e što se kod svakog zahtjeva pokreće novi proces i nakon obrade gasi što je zahtjevno za resurse (procesorsko vrijeme i memorija).

Tablica 5. Usporedba web-aplikacija i web-usluga

web-aplikacije	web-usluge
interakcija korisnik-program	interakcija program-program
statička integracija komponenti	moguća dinamička integracija komponenti
monolitna usluga	moguće je sastavljanje usluga od jednostavnijih

4.5.2 Poslužiteljske skripte (server-side scripts)



- za svaki zahtjev se NE pokreće novi proces
- PHP, ASP, JSO, Ruby on Rails…

4.6 Arhitekture web-aplikacija

Imaju višeslojnu arhitekturu:

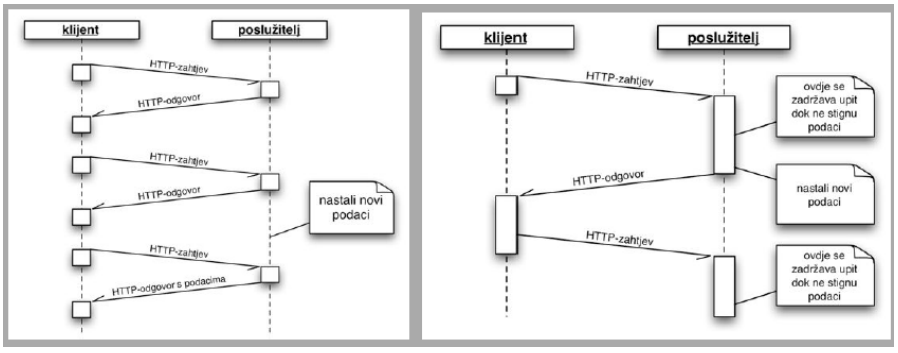
- Presentation layer (PL)
- Bussiness logic layer (BLL)
- Data access layer (DAL)
- Zadnjih godina PL se preselio na klijenta pa se AJAX-om mijenja GUI a podaci se prenose npr. JSON-om ili XML-om

4.8 AJAX

- Tehnika koja služi za dinamičko dohvaćanje podataka sa servera
- AJAX omogućuje da se dijelovi stranice mijenjaju (ažuriraju asinkrono)
- Kako su pravila korištenja AJAX-a u preglednicima različita, potrebna je provjera vrste preglednika i izvršavanje kôda prilagođenog tom pregledniku

Problem chat-a:

- Jedan korisnik u browseru šalje poruku drugom korisniku koji koristi drugi browser
- Jedini način je AJAX preko servera ALI
- Poslužitelj može samo slat odgovor na zahtjev, ne može inicirati komunikaciju
- Poll (prozivanje):
  - Klijent periodički šalje zahtjeve i prima prazne odgovore ako nema ništa pametnog
  - Slaba iskoristivost mreže (puno praznih zahtjeva)
- Long poll (dugo prozivanje):
  - Klijent pošalje zahtjev ali server čeka sa odgovorom dok ne dobije podatke
  - Problem uvijek otvorene konekcije prema serveru (dok se čeka odg)



5. Web usluge

Pod pojmom web-usluge mogu se podrazumijevati dvije stvari: web-usluga koja se temelji na tehnologijama SOAP, WSDL, UDDI ili web-usluga koja je zapravo usluga/aplikacija na nekom web sjedištu npr. Facebook.

Obični web koriste ljudi dok su web usluge namijenjene za programe. Web services (WS) omogućuje lakše korištenje weba umjesto RPC/Java RMI…

Osnovna ideja:

- komunikacija se temelji na XML-u
- koristi već postojeću internetsku infrastrukturu i protokole
- usluge su dostupne putem Interneta
- omogućuje integraciju između različitih aplikacija
- ne ovisi o programskom jeziku ili zatvorenoj tehnologiji jedne tvrtke
- omogućuje otkrivanje usluga koje nude te aplikacije
- usluge su slabo povezane
- temelji se na industrijskim standardima

Web usluga je program koji :

- možemo identificiran URI-jem,
- komunicira s klijentskim programima putem Weba,
- ima sučelje (API) opisano standardima web-usluga i
- omogućuje korištenje neovisno o platformama i programskim jezicima

Osnovna skupina standarda:

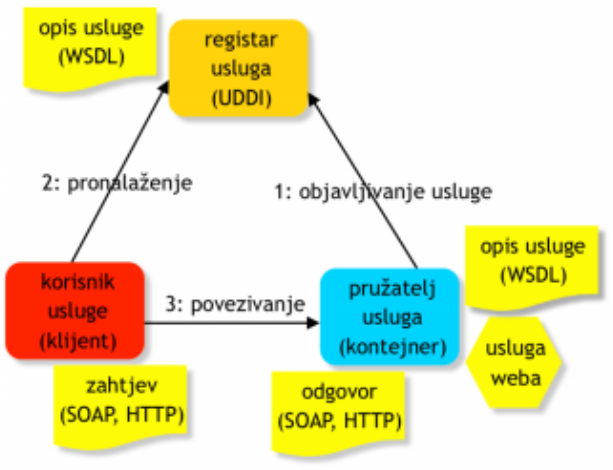
- WSDL (Web Services Definition Language) – opisuje uslugu,
- SOAP (Simple Object Access Protocol) – format poruke
- UDDI (Universal Description, Discovery and Integration) – za otkrivanje usluga

Druga generacija standarda omogućuje korištenje u poslovnom okruženju koje mora biti sigurno i fleksibilno:

- WS-Coordination – protokol za koordinaciju raspodijeljenih aplikacija,
- WS-Transaction – podrška za transakcije,
- BPEL4WS (Business Process Execution Language) – jezik za formalnu specifikaciju poslovnih procesa i interakcijskih protokola,
- WS-Security – sigurnosni protokol (TLS, integritet, privatnost, …),
- WS-ReliableMessaging – za pouzdano slanje i primanje podataka,
- WS-Policy – za uređivanje pravila i prava,
- WS-Attachments – za slanje dodataka u zahtjevima i odgovorima,
- WS-Addressing – adresiranje usluga i poruka

Arhitektura web-usluge se sastoji od 3 entiteta:

- pružatelja usluge,
- registra usluga i
- korisnika usluge



- najprije davatelj registrira uslugu u registru preko UDDI i šalje opis u WSDL
- korisnik dohvaća opis iz registra i dalje preko SOAP i HTTP komunicira sa davateljem

klijent-poslužitelj	web-usluge
unutar tvrtke	između tvrtki
ograničeno na skup programskih jezika	neovisne o programskom jeziku
proceduralno	temelji se na porukama
obično ograničeno na određeni transportni sloj	jednostavno se koristi različitim transportnim mehanizmima
jako povezani dijelovi	slabo povezane usluge
učinkovita obrada (prostor/vrijeme)	relativno neefikasna obrada

web-aplikacije	web-usluge
interakcija korisnik-program	interakcija program-program
statička integracija komponenti	moguća dinamička integracija komponenti
monolitna usluga	moguće je sastavljanje usluga od jednostavnijih

5.1.1 RPC Poziv udaljene procedure

- kritiziran jer usko povezuje klijenta i uslugu
- usluga usko vezana uz implementacijski jezik

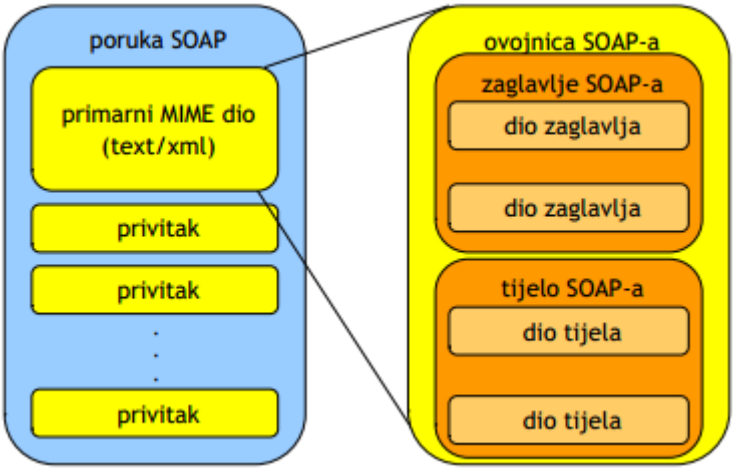
5.1.2 Usluge temeljene na porukama/dokumentima

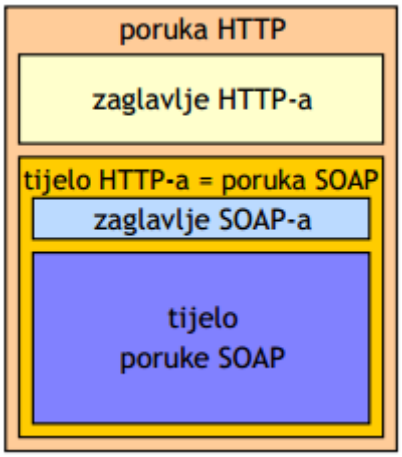
- SOA (Service Oriented Architecture)
- Implementacija temeljena na ugovorima propisanim WSDL-om (što se prenosi i kako?)

5.1.3 Usluge temeljene na prijenosu prikaza stanja resursa

- HTTP se koristi→ jasno sučelje (GET, POST…)
- Može koristiti WSDL i SOAP
- RESTful

5.2 Protokol SOAP

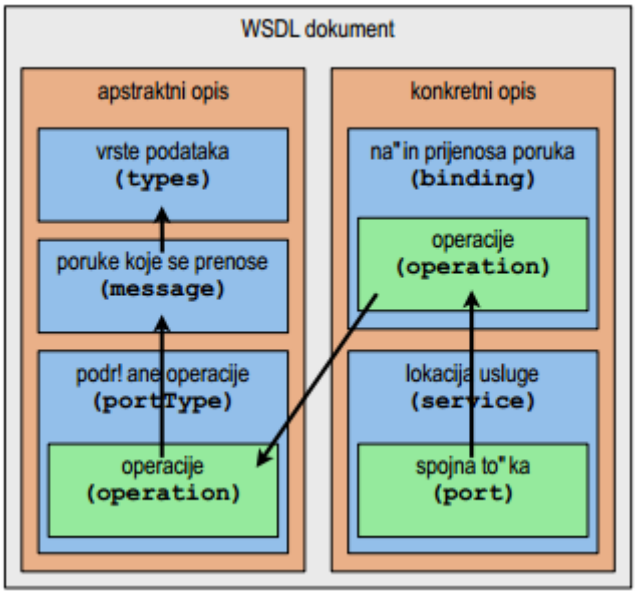
- Simol Object Access Protocol omogućuje komunikaciju sa web uslugom
- **Poziv udaljenih procedura (RPC):**
  - Prijenos serijaliziranih parametara i rezultata
  - Dobro definirana sučelja i tipovi podataka
  - Prilagodni kod može bit generiran automatski
- **Razmjena dokumenata/poruka:**
  - Koriste se XML dokumentu
- 

- 

5.3 Jezik WSDL

Opis web-usluge se sastoji od dva dijela koji se međusobno referenciraju i stoga su povezani:

- apstraktnog opisa
- konkretnog opisa.



1. types: definira vrste podataka neovisne o platformi i jeziku (koristi se XML
Schema),
2. message: definiraju ulazne i izlazne poruke koje se mogu koristiti kao parametri
usluge,
3. operation: predstavlja jednu operaciju/metodu/proceduru koja je definirana u
usluzi, a sastoji se od definicija ulaznih, izlaznih i iznimnih poruka koje se mogu
razmjenjivati korištenjem ove operacije,
4. portType: koristi poruke (pod 2) kako bi opisao sve operacije koje pruža usluga.
1. binding: definira kako je konkretna implementacija povezana s operacijama u
apstraktnom opisu i definira format u kojem će se poruke prenositi (protokol i
elemente) i
2. service: definira URI na kojem je usluga isporučena tj. na kojoj adresi se može
pozvati usluga (taj URI je definiran u spojnoj točki – oznaci port).

5.3.1 Apstraktni opis

Interakcija klijent-server sastoji se od razmjene poruka. Server prihvaća poruku i može vratiti poruku ili baciti iznimku. Svaka poruka pripada nekoj vrsti, a svaka vrsta se sastoji od niza podataka.
Trebamo:

1. opisati sve vrste podataka u svim porukama
2. navesti sve poruke koje se mogu koristiti a svaka je poruka predstavljena nizom podataka
3. opisati svaku metodu/operaciju/proceduru kao kolekciju ulaznih, izlaznih i iznimnih poruka

Navedeni opisi moraju biti platformski i jezično neovisni.

Apstraktan opis se sastoji od 4 važna elementa:

1. *types* : definira vrste podataka neovisne o platformi i jeziku (koristi se XML Schema)
2. *message*: definiraju ulazne i izlazne poruke koje se mogu koristiti kao parametri usluge
3. *operation*: predstavlja jednu operaciju/metodu/proceduru koja je definirana u usluzi, a sastoji se od definicija ulaznih, izlaznih i iznimnih poruka koje se mogu razmjenjivati korištenjem ove operacije
4. *portType*: koristi poruke (pod 2) kako bi opisao sve operacije

5.3.2 Konkretni opis

Svaka usluga mora imati jedinstven URI preko kojeg joj se pristupa. Ima definiran protokol i format i te su definicije sadržane u konkretnom opisu koji se sadrži od 2 dijela:

1. *binding* :
    - definira kako je konkretna implementacija povezana s operacijama u apstraktnom opisu i definira format poruka. Definira transportne protokole i stil
2. *service* :
    - definira URI na kojem je usluga (definiran u spojnoj točki –port)

5.3.3 Element WSDL-a

- **definitions**
    - osnovni element
- **types**
    - opis podatka pomoću XML-a
    - samo kad se koriste podaci koji nisu osnovni
- **message**
    - opis jednosmjerne poruke
    - definira ime poruke
    - sastoji se od dijelova **part**
    - svaki dio se referencira na vrste podataka iz types
- **portType**
    - definira operacije
    - reference na poruke koje mogu biti:
        - parametri
        - rezultati
- **binding**
    - kako će se poruke iz operacije prenositi
    - (HTTTP GET, POST, SOAP, SMTP)
    - Stil definira vrstu poruke:
        - **Rpc**
            - zahtjev će imati omotač u kojem će pisati naziv funkcije koja se poziva
        - **document**
            - zahtjev i odgovori će imati "obične" XML dokumente
    - poruke mogu koristiti dvije vrste pakiranja poruka:
        - **literal**
        - **encoded**
- **sevices**
    - definira lokaciju URL
- **import**

5.4 UDDI

UDDI (Universal Description, Discovery, and Integration) je protokol koji definira na koji način se mogu objaviti i otkriti web-usluge. Obično se UDDI registar koristi unutar neke organizacije. Registar se sastoji od 3 dijela:

- **imenika** (engl. white pages) – koji sprema adrese, kontakte i identifikatore,
- **poslovnog imenika** (engl. yellow pages) – koji ima kategorizaciju područja, usluga i proizvoda te lokacije i
- **tehničke informacije** (green pages) – koji sadrži tehničke informacije o uslugama (dokumente u WSDL-u).

Ne postoji centralni registar na Internetu, već svaka koristi organizacija vlastiti.

5.7 Web usluge temeljene na prikazu stanja resursa (REST)

Representational State Transfer je arhitekturni stil a ne API ili standard.

- Klijent šalje zahtjev za resursom (id resursa i vrsta podatak koji se očekuje)
- Odgovor sadrži prikaz resursa i niz parametara koji ga opisuju
- dohvat može uzrokovati promjenu stanja
- upiti su neovisni jedan o drugom

Metode koje se koriste:

- **GET**:
  - Dohvat resursa
  - Sigurna
  - Idempotentnost
  - cachale
- **POST:**
  - Slanje na obradu ili dodavanje u kolekciju
- **PUT:**
  - Srvaranje ili obrada postojećeg resursa
  - Idempotentnost
- **DELETE:**
  - Idempotentnost

Postupak izrade ovakve usluge:

1. Potrebno je identificirati entitete kojima se želi omogućiti pristupanje (npr. lista korisnika, podaci o korisnicima);

2. Za svaki resurs treba definirati shemu identifikatora pomoću URL-a. Shema treba završavati imenicama, a ne glagolima jer glagoli označavaju radnju. Radnja nije resurs. Npr. http://localhost:8080/REST2011/rest/persons za popis korisnika i http://localhost:8080/REST2011/rest/person/id za pojedinog korisnika s identifikatorom id;

3. Za svaki resurs treba odrediti da li je moguće samo dohvaćati podatke o resursu (korištenje metode GET) ili je potrebno i manipulirati s njima (korištenje metoda POST, PUT i DELETE);

4. Svi resursi koji se dohvaćaju pomoću metode GET za tu metodu moraju imati svojstvo sigurnosti tj. obrada metode GET na poslužitelju ne smije mijenjati stanje resursa;

5. Svaki resurs treba vratiti što manje detalja tj. treba stavljati poveznice na druge resurse u slučaju da klijent želi detalje resursa ili saznati povezanost s drugim resursima;

6. Usluge treba dizajnirati tako da se postepeno (kroz više upita koji se otkrivaju preko poveznica tj. URL-ova) otkrivaju podaci o resursima. Nije dobro sve staviti u jedan veliki dokument;

7. Za svaku metodu svakog resursa potrebno je definirati formate upita i odgovora. Ako se koristi XML onda je dobro za te opise koristiti XML Schemau;

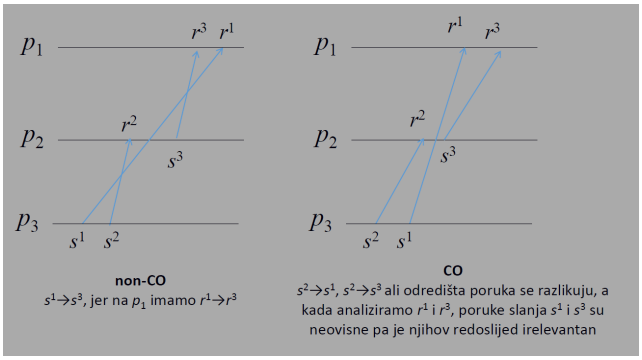
8. Opisati kako se web-usluge mogu koristiti npr. WSDL-om ili HTML-om.

5.8 Service Oriented Architecture (SOA)

- Bitno svojstvo je razdvajanje interesa (separation of concerns)
- Svaka se usluga može nadograđivati na druge usluge i na taj način stvarati kompleksne usluge
- SOA nije vezana za konkretnu tehnologiju iako je najčešće kao web-usluga

Svojstva uslužno orijentirane arhitekture su:

- slaba povezanost usluga – jedna usluga ne ovisi o tehnologiji implementacije druge usluge,
- korištenje uslužnih ugovora – vlasnici usluga i klijenata trebaju koristiti i ugovore da bi se riješio pravni segment,
- dogovor o komunikaciji (opis usluge – WSDL, XML Schema, policy, pravni dokumenti),
- autonomnost – klijent može biti autonoman,
- usluga ima kontrolu nad logikom koju enkapsulira,
- apstrakcija – dogovara se sučelje, a ne implementacija,
- izvana se vidi samo ono što je u ugovoru,
- ponovna iskoristivost – usluga se može iskoristiti kao dio druge usluge,
- uslugu mogu koristiti druge usluge,
- mogućnost slaganja u složene usluge,
- usluge bez stanja su skalabilne
- mogućnost otkrivanja usluga



6 Model Raspodijeljenog Sustava preza 5

6.1 Osnovni model

Osnovni model raspodijeljenog sustava se sastoji od skupa autonomnih i asinkronih procesa p1, p2,… pn koji komuniciraju razmjenom poruka putem komunikacijske mreže. Cij je kanal d pi do pji samo to.

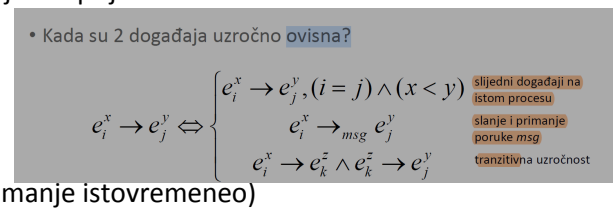
1. Ako se procesi izvode na različitim procesorima onda su međusobno autonomni i asinkroni
2. Procesi ne dijele memorijski prostor
3. Zbog kašnjenja paketa pri prijenosu na fizičkom liku, neminovno se javlja kašnjenje poruka pri komunikaciji procesa
4. Procesi ne koriste jedinstveni zajednički sat te je problem sinkronizacije vremena

Izvođenje procesa = akcije tijekom vremena. Akcije su:

- Unutarnji događaji (npr. obrada podataka na procesu koja proces dovodi u novo stanje)
- Slanje poruke
- Primanje poruke

Uzročna relacija označava uzročnu ovisnost 2 događaja tijekom raspodijeljenog izvođenja (→). Vrijedi i tranzitivna uzročnost a→b i b→c – a→c.

Uzročna neovisnost ako ne možemo sa sigurnošću reći koji je bio prije.



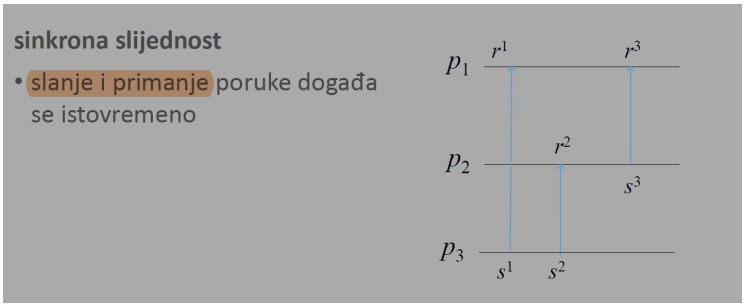
Slije isporuka poruka :

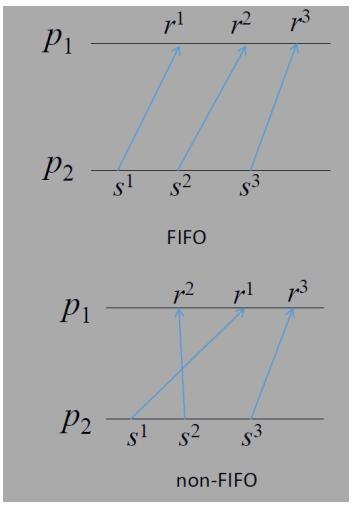
- FIFO
- Non-FIFO
- Kanal koji osigurava sinkronu slijednost (slanje i primanje istovremeneo)
- -||- uzročnu slijednost non-CO CO

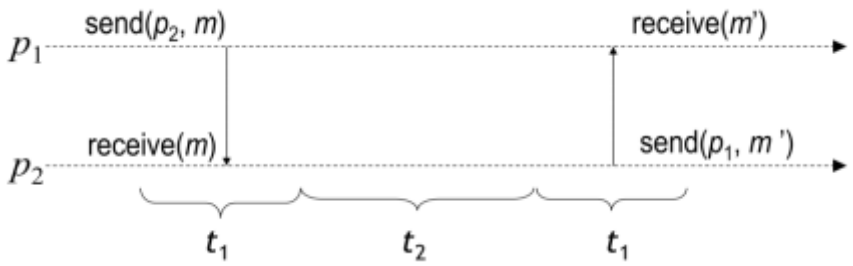
Globalno stanje raspodijeljenog sustava određeno je stanjem pojedinih procesa i komunikacijskih kanala. Stanje procesa određeno je stanjem lokalne memorije i izvođenjem unutarnjih događaja. Stanje kanala određeno je skupom primljenih i poslanih poruka. Izvođenje bilo kojeg događaja mijenja lokano stanje procesa/kanala, ali istovremeno i globalno stanje raspodijeljenog sustava.

Osnovni model rassu proširuje se kao sinkroni ili asinkroni model sustava:

- Sinkroni:
  - Pojednostavljenje (isključivo) pretpostavka da svi procesi izvode korakre istovremeno
  - Svi procesi imaju sinkronizirana vremena lokalna
  - Poznata gornja vremenska granica za
    - Izvođenje prijelaza nekog procesa
    - Trajanje prijenosa poruke kanalom

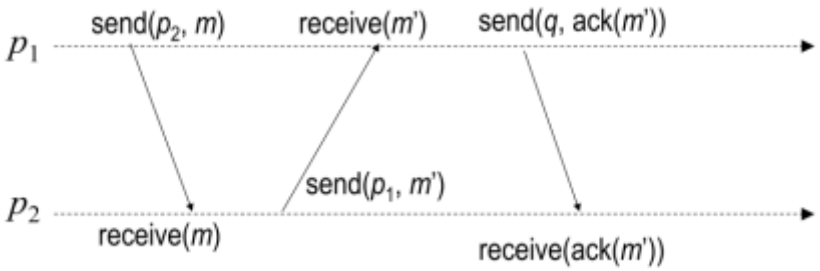






- Inicijalno svi procesi u početnom stanju i svi kanali prazni
- Prva faza-proces generira poruke i postavlja ih na izlazne kanale
- Druga faza- tren stanje+ poruka == prijelaz

- Asinkroni:
  - Akcije u proizvoljnom slijedu
  - Kompliciran te se najčešće koristi sinkroni za objašnjavanje
  - Pretpostavke:
    - Procesi NEMAJU sinkronizirana lokalna vremena
    - Ne postoji gornja granica vremenska za izvođnej prijelaza nekog procesa no uvijek je konačno
    - Ne postoji vremenska gornja granica za prijenos poruke kanalom



6.2 Sinkroni model

Lokalnost- svaki proces poznaje samo svoje neposredne susjede, ulazne i izlazni ali NEMA centralnog procesa koji poznaje sve o svemu u topologiji.

Model procesa:

$v_i \in V$ se modelira kao uređena četvorka: ($states_i$, $start_i$, $msgs_i$, $trans_i$)
$states_i$ – skup mogućih stanja procesa
$start_i$ – skup početnih stanja, $start_i \subset states_i$, $start_i \neq 0$
$msgs_i$ – funkcija za generiranje poruka koja određuje izlaznu poruku za svakog susjeda na temelju trenutnog stanja procesa, tj. $msgs_i : states_i \times out\text{-}nbrs_i \rightarrow M_i \subset M \cup \{null\}$
$trans_i$ – funkcija prijelaza, određuje sljedeće stanje na temelju trenutnog stanja i primljenih poruka od ulaznih susjeda

Algoritmi se izvode nad sinkronim modelom u koracima. Inicijalno svi procesi u poč stanju a svi kanali prazni. Korak ima 2 faze:

1. Faza: primjeni msgs za svaki proces v na temelju tren. Stanja. Poruke se postavljaju na izlazne kanale i prosljeđuju se susjedima

2. Faza: primjeni trans za svaki proces v, koja na temelju tren. Stanja i primljenih poruka određuje sljed. Stanje . briše sve poruke na kanalima.

Mjera složenosti algoritama:

1. **Vremenska složenost**
    - Mjeri se brojem izvedenih koraka
2. **Komunikacijska složenost**
    - Mjeri se broj pripremljenih i poslanih poruka na kanalima

Primjer 1. Odabir vođe u sinkronom prstenu

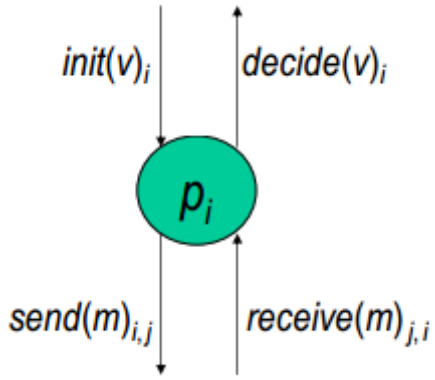
- Svi procesi su jednaki osim UID-a koji su usporedivi (< >)
- Proces random može odabra0ti UID i zna za svoje susjede
- Jednosmjerna komunikacija (kazaljka na satu)
- Procesi ne znaju veličinu n
- Vođa je najveći UID
- Susjedu se šalje UID, ako je veći od vlastitog prosljijedi, ako je == proglasi se vođom, inače 0
- Vremenska složenost je O(n)
- Komunikacijska složenost je O(n^2)
- Leader šalje posebnu poruku *halt* kako bi ostalima javio da je algoritam gotov onda je složenost O(2n)

Primjer 2. Odabir vođe u usmjerenoj mreži

- Svi parovi procesa su na konačnoj udaljenosti
- Svaki proces ima UID
- Temelji se na preplavljivanju
    - Svaki proces ima UID koji se random odabire
    - Svaki proces zna diameter(G) mreže G
- U svakom koraku šalje svim susjedima max vrijednost koju ima zapisanu (inicijalno svoju)
- Vremenska složenost je Idiametter(G) a komunikacijska je *diameter(G) x |E|*

6.3 Asinkroni model

- Svaki se proces i svaki kanal definira ulazno-izlaznim automatom koji definira svaku komponentu rasus njenu interakciju sa ostalim komponentama
- Primitak ulazne vrijesnosti za varijablu v je *init(v)*, odluka *decide(v)* je izlaz
- Proces pi komunicira sa ostalim procesima pomoću *send(m)ij i recieve(m)ji*

- 

- I/O automat *A* se definira kao uređena četvorka (*sig*(*A*), *states*(*A*), *start*(*A*), *trans*(*A*)) i sastoji se od sljedećih komponenti:
  - *sig*(*A*) je signatura automata *A*, sig(*A*)={ *in*(*A*), *out*(*A*), *int*(*A*) }, je skup koji definira ulazne, izlazne i unutarnje događaje automata *A*
  - *states*(*A*) definira skup mogućih stanja automata
  - *start*(*A*) je skup početnih stanja automata, *start*(*A*) ≠ 0 i podskup je skupa *states*(*A*)
  - *trans*(*A*) je funkcija prijelaza koja za svako stanje *s* iz skupa *states*(*A*) i svaki ulazni događaj π iz skupa *in*(*A*) definira prijelaz (*s*, π, *s'*) ∈ *trans*(*A*) gdje je *s'* ∈ *states*(*A*) novo stanje automata

Primjer 1: odabir vođe u asinkronom prstenu

- Prsten je modeliran pomoću 2 vrste I/O automata
  - Jedan za proces
  - Drugi za kanal
- FIFO kanal
- Svaki proces ima ulazni spremnik od max n poruka
- Signatura je definirana sljedećim ulaznim i izlaznim događajima:
  - in(A): $receive(v)_{i-1,i}$, gdje je *v* UID
  - out(A): $send(v)_{i,i+1}$; $leader_i$

  Skup mogućih stanja $states_i$ = { *u*, *send*, *status*}
  - *u* je UID, inicijalno UID za proces p_i
  - *send* je FIFO queue UID-ova veličine *n*, inicijalno sadrži UID za proces p_i
  - *status* ∈ {*unknown*, *chosen*, *reported*}, inicijalno *unknown*

  Funkcija prijelaza $trans_i$:
  - $send(v)_{i,i+1}$ – preduvjet: *v* je 1. element iz *send*, posljedica: briši *v* iz *send*
  - $leader_i$ – preduvjet: *status* = *chosen*, posljedica: *status* := *reported*
  - $receive(v)_{i-1,i}$

    if v > u: add v to send

    if v = u: then status := chosen

    if v < u: do nothing

7 SINKRONIZACIJA PROCESA U VREMENU

Def. Sinkronizacija procesa u vremenu označava koordinaciju njihova izvođenja i međudjelovanja u svrhu skladnog rada cjelokupnog sustava tijekom vremena.

Za ispravni rad nekog raspodijeljenog sustava neophodno je osigurati usklađeno djelovanje njegovih raspodijeljenih procesa. Četiri osnovna razloga za primjenu sinkronizacije procesa u raspodijeljenoj okolini su:

1. uporaba dijeljenih sredstava, - ako 2 procesa pokusaju pristupiti
2. usuglašavanje vremenskog redoslijeda akcija, - ako ovise procesi medusobno
3. nadgledanje i upravljanje zadaćama skupa procesa te -
4. uspostava suradnje skupa procesa - p2p za dijeljenje datoteka

7.1 Primjena sata u raspodijeljenoj okolini

- Javljaju se slijedeći problemi:
  - Satovi imaju različita odstupanja
  - Vrijednosti satova nisu usklađene
  - Satovi imaju različiti takt
- Koriste se dva pristupa:
  - Fizički sat
  - Logički sat- bitan redoslijed a ne točni vremenski trenutci

7.1.1 Usklađivanje fizičkih satova

- Dva najčešće korištena algoritma:
  - Criatianov- satovi se usklađuju stvarnim vremenom
    1. Korisnik šalje zahtjev poslužitelju. Zahtjev putuje t_z vremenskih jedinica do poslužitelja.
    2. Poslužitelj prima zahtjev te unutar sljedećih t_p vremenskih jedinica obrađuje primljeni zal i šalje odgovor klijentu.
    3. Odgovor putuje t_O vremenskih jedinica do klijenta, a sadrži točno vrijeme T_3 na poslužite trenutku primanja zahtjeva i točno vrijeme T_4 na poslužitelju u trenutku slanja odgovora.
    4. U zadnjem koraku klijent prima odgovor te na osnovi vremenskih trenutaka koje je prim poruci i izmjerenih vremenskih trenutaka T1 i T4 pomiče svoje lokalno vrijeme za $T_3 + t_O - T_4 \approx T_3 + \frac{t_Z + t_O}{2} - T_4 = T_3 + \frac{(T_4 - T_1) - (T_3 - T_2)}{2} - T_4 = \frac{(T_2 - T_1) - (T_3 - T_4)}{2}$.
    - Problem je neusklađenosti satova na poslužitelju i na računalu
    - Nedovoljno precizan jer vrijeme dolaska signala do servera i odlazak sa servera je aproksimiran
    - Sinkronizacija je dobra ako je RTT = T4-T1 puno manje od preciznosti koja se želi postići

- Berkley – međusobno se usklađuju

1. Upravitelj šalje svoje trenutno vrijeme preostalim procesima u raspodijeljenoj okolini te od njih traži da mu pošalju svoja vremena.
2. Procesi primaju zahtjev, određuju razliku svog trenutnog lokalnog vremena u odnosu na vrijeme primljeno u poruci te zatim upravitelju šalju odgovor u kojem je zapisana razlika izračunatih vremena.
3. Upravitelj prima odgovore te računa pomak za satni mehanizam svakog pojedinog procesa u raspodijeljenoj okolini pa i sebe samog te zatim izračunate pomake prosljeđuje odgovarajućim procesima.

7.1.2 Usklađivanje logičkih satova

- Logički sat je funkcija koja vrši mapiranje sa skupa događaja na oznake vremena uz očuvanje konzistentnosti
- Odnosno ako a→b tada je T(a) < T(b), ukoliko vrijedi i obrat onda pričamo o strogoj konzistentnosti

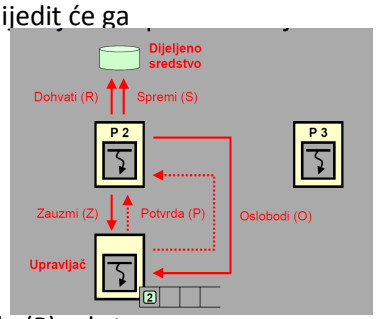
7.2 Sinkronizacija tijeka izvođenja procesa

7.2.1 Mehanizam semafora

- Semafor je proces koji sadrži spremnik s konačnim brojem znački i repom.
- Značke FIFO
- Kada ostane bez znački (sve je podijelio) sve zahtjeve sprema u rep
- Tijek komunikacije (D P V) :
  1. Proces šalje zahtjev semaforu za dohvat (D) jedne ili više znački,
  2. Ako u spremniku semafora postoji traženi broj znački, semafor će procesu predati značke (P),
  3. Ukoliko spremnik ne posjeduje dovoljan broj znački, zahtjev će se staviti u rep čekanja i
  4. Nakon završetka obrade, proces će vratiti semaforu preuzete značke slanjem poruke vrati (V)

7.2.2 Mehanizam razmjene događaja

- Posrednik događaja je proces sa 2 spremnika:
  - Spremnik s objavljenim događajima
  - Spremnik preplata
- Tijek komunikacije sastoji se od :
  - Proces posredniku šalje svoju preplatu (S) koju on sprema u spremnik
  - Neki drugi proces objavljuje događaj (P) posredniku koji ga sprema
  - Ako posrednik ima spremljenu preplatu za objavljeni događaj, proslijedit će ga procesu pretplatniku dojave(N)



7.3 Međusobno isključivanje procesa

7.3.1 Isključivanje putem središnjeg upravljača

- FIFO
- Tijek komunikacije:
  1. Proces šalje zahtjev za zauzimanjem dijeljenog sredstva (Z)
  2. Proces ostvaruje pristup dijeljenom sredstvu nakon primitka potvrde (P) od strane središnjeg upravljača. Proces obavlja akciju dohvaćanja (R) i/ili spremanja (S) dijeljenog sredstva

3. Nakon završetka obrade, proces otpušta zauzeto sredstvo slanjem poruke oslobodi (O) središnjem upravljaču.

7.3.2 Raspodijeljeno isključivanje

- Svaki proces ima vlastiti rep čekanja a svi međusobno razmjenjuju informacije potrebne za usklađivanje stanja svih repova
- Razmjenjuju se 2 vrste poruka:
  - Zahtjev za pristup:
    - Uključuj logičku oznaku trenutka kojeg prosljeđuje svim drugim procesima
  - Potvrda prihvaćanja zahtjeva
- Svi šalju svima, bira se onaj sa najmanjim vremenom (najprije je došao) tj. ostali ga potvrđuju, kada on odradi svoje šalje svima potvrdu o kraju i tako daje…

7.3.3 Isključivanje zasnovano na primjeni prstena

- „značka“ za dodijelu pristupa
- Algoritam:
  1. Čekaj poruku sa značkom od lijevog susjeda (tad imaš pristup)
  2. Ako si zainteresiran onda čitaj/spremaj
  3. Proslijedi poruku sa značkom desno od sebe
- Ako svaki proces 1 pristupi za cijeli krug potrebno je T= N*(Tz+Tp+To) +N*Tt gdje je:
  - Tz→vrijeme slanja zahtjeva
  - Tp →vrijeme primitka poruke odgovora
  - To → vrijeme obrade
  - Tt → vrijeme prijenosa poruke sa značkom

8 KONZISTENTNOST I REPLIKACIJA PODATAKA

8.prezu si preskocio
7.preza je na pola
9. preza je ovdje

Replikacija označava stvaranje i održavanje istovrsnih kopija. Razlozi mogu biti razni. Više kopija == veća pouzdanost. Problem nastaje kada se jedna od replika (ili original) promijeni.

Postupci očuvanja konzistentnosti su složeni jer treba postići:

- Transparentnost pristupa podacima
- Transparentnost lokacijsku i migracijsku
- Konkurencijsku Transparentnost
- Transparentnost na kvar

8.1 Modeli konzistentnosti podataka

Skup procesa: { p,q,r,s,…}, skup podataka {A,B,C,D,…}, skup lokacija {x,y,z,w,…}.

Svaki proces može neki podatak zapisati negdje npr. proces p upisuje podatak A na lokaciju x operacijom pisanja W → W(x,A).

Modeli konzistentnosti podataka govore o tome kako slijed upisa podataka na istu ili različite lokacije od strane jednog ili više procesa, „vide“ drugi procesi koji očitavaju te lokacije.

8.1.1 Stroga konzistentnost

- Odgovara idealnoj situaciji kada je promjena na jednoj lokaciji momentalno vidljiva na svim
- U stvarnosti, promjene će na drugim lokacijama biti vidljive tek nakon komunikacijskog kašnjenja

8.1.2 Slijedna konzistentnost

- Zahtijeva da svi procesi moraju slijed izvođenja operacija čitanja i pisanja u vremenu vidjeti na jednak način
- Bitno je da su zajednički učitani elementi u istom redoslijedu, npr. A, B , C i A,B,D,C su slijedno konzistentni

nekonzistentno
A=>B
B=>A

8.1.3 Povezana konzistentnost

- Oblik slabije slijedne konzistentnosti kod koje se pozornost obraća samo potencijalno povezanim operacijama koje mogu biti u uzročno posljedičnoj vezi i takve operacije se moraju vidjeti jedna za drugom

A=>B=>C
A=>C=>B

8.1.4 Konzistentnost redoslijeda upisivanja

- Ukoliko je redoslijed upisivanja podataka od jednog procesa vidljiv jednako ostalim procesima
- Npr. proces x upisuje redom A, B. U ostalim procesima mora biti očuvan taj redoslijed (može biti i npr. A, C, B)
- FIFO
- Postiže se dodavanjem jedinstvenih oznaka zahtjevu za upisivanjem npr. Wp1(x,C)..

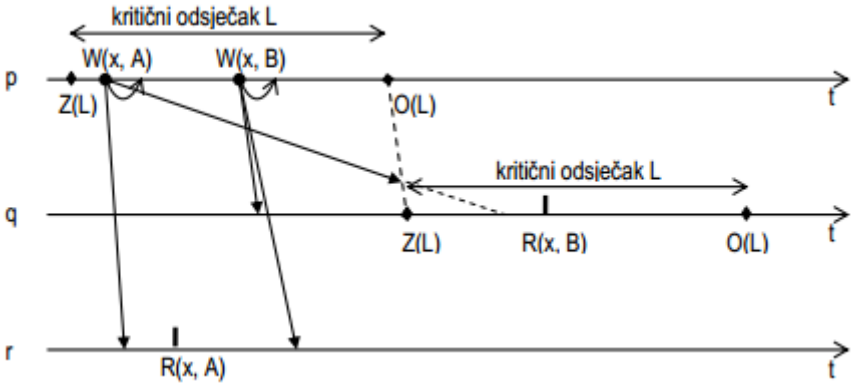
8.1.5 Slaba konzistentnost

- ostvaruje se primjenom sinkronizacijskih varijabli kojima se upravlja usklađivanjem replika podatkovnog objekta

- pri završetku upisivanja podataka proces pokreće sinkronizaciju prema ostalim proceisma preko varijable Sync(S) kako bi svi vidjeli zadnje upisanu vrijednost (zbog mogućeg kašnjenja bi neki procesi mogli vidjeti prije upisanu vrijednost)

8.1.6 Konzistentnost otpuštanja i zauzimanja kritičnog odsječka

- ulazak u KO određen je operacijom zauzimanja dok je izlazak određen operacijom otpuštanja
- konz otpuštanja odgovara modelu kod kojeg se prije izlaska iz KO sve lokalne promjene prosljeđuju svim replikama



-
- konz zauzimanja odgovara modelu kod kojeg se nakon ulaska u KO preuzimaju sve promjene podataka i uspostavlja konz.

8.1.7 Usporedba modela konzistentnosti

- mogu se usporediti sa 2 motrišta:
  - **složenosti ostvarivanja** : Opisuje koliko je zahtjevno ostvariti programsku potporu za uspostavu modela konzistentnosti
  - **složenosti primjene**: Opisuje koliko je složeno koristiti takav model
  -

Složenost ostvarivanja	Model konzistentnosti	Složenost primjene
najveća	stroga konzistentnost	najmanja
↑	slijedna konzistentnost	↓
	povezana konzistentnost	
	slaba konzistentnost	
	konzistentnost otpuštanja	
najmanja	konzistentnost zauzimanja	najveća

8.2 Organizacija sustava replika i vrste replika

Glavni problemi replikacije:

1. Odabir lokacije gdje će se smjestiti replike
   a. Želimo li povećanje pouzdanosti i raspoloživosti ili
   b. Poboljšanje performansi
2. Postavljanje replika na te lokacije
3. Održavanje konzistentnosti

Replike, ovisno o lokaciji dijelimo na:

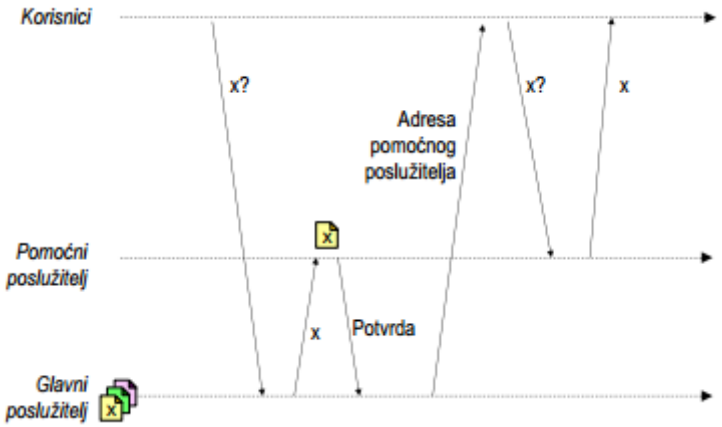
- Trajne
- Poslužiteljske
- Korisničke

8.2.1 Trajne replike

- Početni skup postavljenih replika na skupu replikacijskih servera povezani lokalnom mrežom (grozd računala)
- Statična organizacija (obavlja admin)
- Primjer korištenja ove vrste replika kod weba je prilikom raspodjele opterećenja između poslužitelja na istoj geografskoj lokaciji i raspodjele opterećenja zrcaljenjem (engl. mirroring) replika na različitim geografskim lokacijama
- Pristup takvim serverima preko raspoređivača zahtjeva

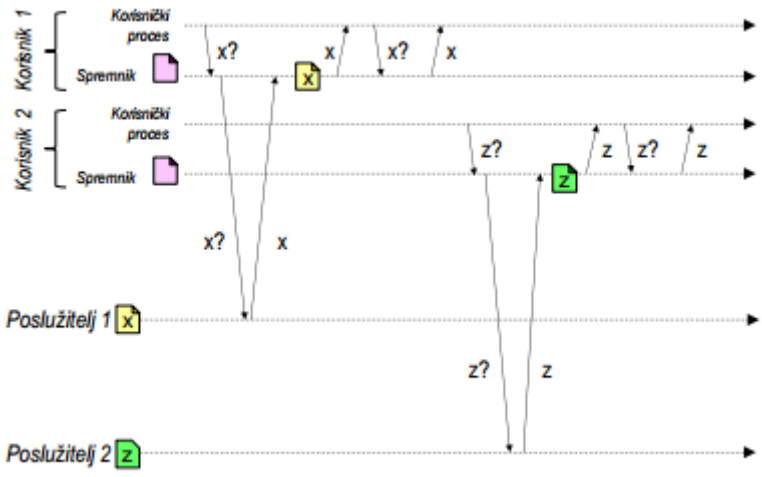
8.2.2 Serverske replike

- Dinamički, u real-time stvorene replike prilikom povećanja potražnje za nekim resursom



- Povećano vrijeme odziva

8.2.3 Korisničke replike



- programi koriste lokalni spremnik za odrzavanje konzistentnosti s posluziteljem

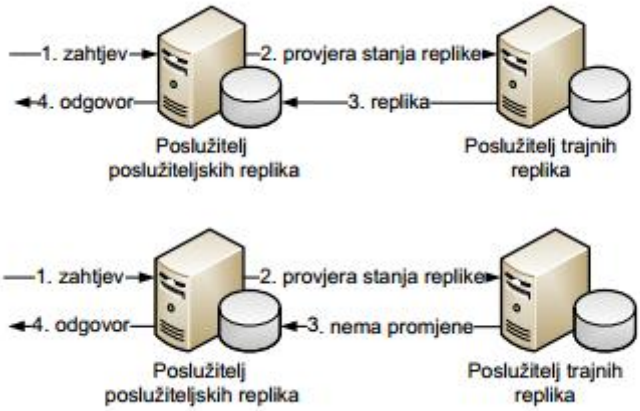
- cache browsera

8.3 Metode održavanja konzistentnosti replika

Razlikujemo pull i push metode ovisno dali se traži ili prosljeđuje.

8.3.1 Dohvaćanje promjena sadržaja replika

- Klijent dohvaca resurs, najprije se provjerava je li podatak iz cache-a validan sve do vrha
- Učinkovitost se računa:
  - Ako je fr(frekv. Čitanja) > fw(frekv.pisanja) gornji slučaj fw a donji fr-fw
  - Ako je fr<=fw, donji slučaj →0 a gornji fr



8.3.2 Prosljeđivanje promjena sadržaja replika

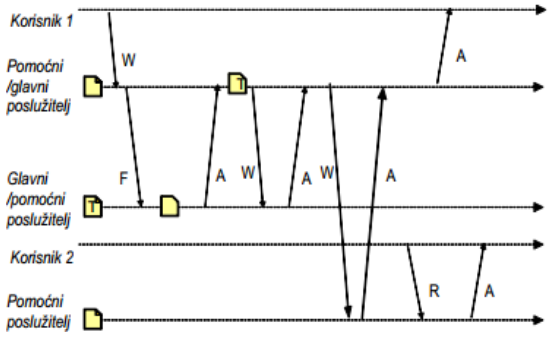
- Uvijek se čita samo iz lokalnog spremnika
- Kad nastupi promjena, tranji server šalje niz hijerarhiju :
  - Novi sadržaj (cijela ili samo dio)
  - Operaciju za promjenu sadržaja
  - Obavijest o promjeni.
    - Zatim se koristi pull
- Metoda push je učinkovitija kada je fr>fw jer bi sa pull pri svakom čitanju bespotrebno provjeravali je li doščo do promene
- Nedostatak je što server trajnih replika mora imati adrese svih poslužitelja replika te opise stanja njihovih replika

8.4 Protokoli za ostvarivanja operacija čitanja i upisivanja replika

U praksi je čest slučaj gdje korisnik/klinejt inducira promjenu na replikama. Postoje protokoli kojo omogućuju ostvarivanje čitanja i pisanja replika od strane korisnika. Dijele se na protokole sa aktivnom i pasivnom replikacijom.

8.4.1 Pasivna replikacija

- Razlikuju glavnu (trajnu) repliku od ostalih te se operacije pisanja provode nad tom replikom dok se operacije čitanja provode nad bilo kojom replikom (najčešće najbližom)

1. **Udaljeno obnavljanje stanja replike:**
   - Pretpostavlja statičnost glavne replike
   - Korisnik piše na glavnu repliku i onda se prosljeđuju promjene dalje
   - Uspostava slijedne konzistentnosti
   - Povećano vrijeme izvođenja operacije
2. **Lokalno obnavljanje stanja replike:**
   - Dinamična glavna replika
   - 

- uzastopno vrijeme pisanja u kratkom vremenu na domacinu
- korisnik moze pristupiti lokalnoj replici neovisno o trajnoj