

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1560
**SINKRONIZACIJSKI I KOMUNIKACIJSKI
MEHANIZMI ZA SUSTAVE ZASNOVANE NA
USLUGAMA**

Saša Rešković

Zagreb, rujan 2005.

Zahvaljujem mentoru prof. dr. sc. Siniši Srbljiću na savjetima i pomoći koju mi je pružio u stručnom i osobnom razvoju.

Na brojnim savjetima i pomoći pri pisanju diplomskog rada zahvaljujem mr. sc. Andri Milanoviću.

Također, zahvaljujem dipl. ing. Danielu Skrobi i kolegi Davoru Čapaliji na pomoći pri ostvarenju praktičnog dijela diplomskog rada.

Sadržaj

| | |
|--|-----------|
| Sadržaj | 3 |
| 1 Uvod | 5 |
| 2 Postojeći mehanizmi komunikacije i sinkronizacije | 7 |
| 2.1 Mehanizmi ugrađeni u operacijske sustave | 8 |
| 2.1.1 <i>Natjecanje za računalna sredstva</i> | 8 |
| 2.1.2 <i>Kritični odsječci</i> | 9 |
| 2.1.3 <i>Binarni i opći semafor</i> | 12 |
| 2.1.4 <i>Potpuni zastoj</i> | 14 |
| 2.1.5 <i>Monitori</i> | 15 |
| 2.1.6 <i>Komunikacija među procesima</i> | 16 |
| 2.1.7 <i>Mehanizmi za komunikaciju ugrađeni u operacijski sustav Windows</i> | 17 |
| 2.1.8 <i>Mehanizmi za sinkronizaciju ugrađeni u operacijski sustav Windows</i> | 19 |
| 2.2 Mehanizmi u raspodijeljenim sustavima | 21 |
| 2.2.1 <i>Pozivi udaljenih procedura i poziv metoda udaljenih objekata</i> | 22 |
| 2.2.2 <i>Komuniciranje razmjenom poruka</i> | 23 |
| 2.2.3 <i>Vrste komunikacije porukama</i> | 24 |
| 2.2.4 <i>Prolazna komunikacija zasnovana na porukama</i> | 26 |
| 2.2.5 <i>Sustavi zasnovani na repovima poruka</i> | 28 |
| 3 Računalni sustavi zasnovani na uslugama | 31 |
| 3.1 Raspodijeljeni računalni sustavi | 31 |
| 3.2 Raspodijeljeni sustavi zasnovani na uslugama | 33 |
| 3.2.1 <i>Načela računarstva zasnovanog na uslugama</i> | 35 |
| 3.2.2 <i>Podjela usluga prema složenosti</i> | 37 |
| 3.2.3 <i>Zrnatost i suradnja usluga</i> | 38 |
| 3.2.4 <i>Svojstva arhitekture zasnovane na uslugama</i> | 40 |
| 3.2.5 <i>Osnovna Web Services arhitektura</i> | 41 |
| 3.2.6 <i>XML</i> | 43 |
| 3.2.7 <i>SOAP</i> | 44 |
| 3.2.8 <i>WSDL</i> | 45 |
| 3.2.9 <i>UDDI</i> | 47 |
| 3.3 Načela programiranja na veliko i programiranja na malo | 49 |
| 4 Okolina raspodijeljenog sustava zasnovanog na uslugama | 51 |
| 4.1 Raspodijeljeni sustav zasnovan na suradnji i natjecanju | 51 |

| | |
|---|-----------|
| 5 Ostvareni komunikacijski i sinkronizacijski mehanizmi..... | 54 |
| 5.1 Osnovna arhitektura ostvarenih mehanizama | 54 |
| 5.2 Binarni i opći semafor..... | 55 |
| 5.2.1 <i>Stvaranje i uništavanje.....</i> | 55 |
| 5.2.2 <i>Sučelje binarnog i općeg semafora.....</i> | 56 |
| 5.2.3 <i>Način zauzimanja semafora</i> | 57 |
| 5.2.4 <i>Okolina binarnog i općeg semafora.....</i> | 58 |
| 5.2.5 <i>Svojstva binarnog i općeg semafora</i> | 59 |
| 5.3 Ostvarenje binarnog i općeg semafora..... | 60 |
| 5.3.1 <i>Mrežne usluge i serijalizacija razreda u XML.....</i> | 61 |
| 5.3.2 <i>Sučelje mrežne usluge binarnog semafora.....</i> | 62 |
| 5.3.3 <i>Spremnik instanci binarnih semafora</i> | 64 |
| 5.3.4 <i>Instanca binarnog semafora</i> | 65 |
| 5.3.5 <i>Ostvarenje mehanizma dojave</i> | 67 |
| 5.3.6 <i>Razredi općeg semafora</i> | 68 |
| 5.4 Spremnik poruka..... | 69 |
| 5.4.1 <i>Stvaranje i uništavanje.....</i> | 70 |
| 5.4.2 <i>Sučelje spremnika poruka</i> | 70 |
| 5.4.3 <i>Načini dohvaćanja poruka</i> | 71 |
| 5.4.4 <i>Vrste komunikacije porukama</i> | 72 |
| 5.4.5 <i>Okolina spremnika poruka</i> | 73 |
| 5.5 Ostvarenje spremnika poruka | 74 |
| 5.5.1 <i>Sučelje mrežne usluge spremnika poruka</i> | 75 |
| 5.5.2 <i>Spremnik instanci spremnika poruka</i> | 75 |
| 5.5.3 <i>Instanca spremnika poruka</i> | 76 |
| 5.5.4 <i>Ostvarenje mehanizma dojave</i> | 77 |
| 5.5.5 <i>Format poruke.....</i> | 78 |
| 6 Zaključak..... | 80 |
| Literatura | 82 |

1 Uvod

Zbog sve većih zahtjeva koje aplikacije trebaju zadovoljiti one postaju sve složenije i zahtijevaju složenije računalne sustave. Razvoj mikroprocesora u osamdesetim godinama prošlog stoljeća i brzih lokalnih računalnih mreža omogućio je izgradnju raspodijeljenih računalnih sustava povezivanjem jeftinih računala u lokalne mreže. Raspodijeljeni računalni sustavi imaju nekoliko prednosti pred centraliziranim. Jeftiniji su i mogu jednostavno biti nadograđeni kad se za to pojavi potreba te podržavaju veći broj korisnika. Glavni nedostatak raspodijeljenih računalnih sustava je njihova raznorodnost zbog koje je programiranje primjenskih programa za njih vrlo složen zadatak. Zbog toga su razvijeni posrednički sustavi koji skrivaju složenost nudeći dobro definiran skup usluga.

Danas razvoj raspodijeljenih programa postaje sve važniji. Višejezgrevni procesori su postali dostupni širokom krugu korisnika, a sve je više računala povezano u lokalne mreže. Zbog toga se postavljaju novi zahtjevi pri razvoju primjenskih programa. S jedne strane postoji potreba za programima koji se koriste u sustavima za rad u stvarnom vremenu ili za znanstvena istraživanja. Njihova svojstva moraju biti prilagođena za brzu i učinkovitu obradu podataka. Za razliku od toga, u poslovnom je svijetu bitno i vrijeme izgradnje poslovnih sustava i međusobna suradnja poduzeća koja koriste različite računalne sustave i primjenske programe. Zbog toga primjenski programi koji surađuju trebaju međusobno biti labavo vezani. Jedan od načina na koji se to može postići je računarstvo zasnovano na uslugama.

Kod razvoja raspodijeljenih programa potrebni su složeniji mehanizmi komunikacije i sinkronizacije nego kod programa koji se izvode na jednom računalu. U sustavima zasnovanim na uslugama posrednički sustav treba ponuditi odgovarajuće usluge za komunikaciju i sinkronizaciju. Zbog toga su za potrebe izgradnje raspodijeljenog sustava zasnovanog na suradnji i natjecanju ostvareni komunikacijski i sinkronizacijski mehanizmi kao mrežne usluge. Njihovo ostvarenje je opisano u ovom radu.

Na početku rada su opisani postojeći mehanizmi komunikacije i sinkronizacije u operacijskim sustavima i raspodijeljenim sustavima. U trećem poglavlju navedena su načela računarstva zasnovanog na uslugama. Objasnjeno je korištenje mrežnih usluga i skup

standarda koji se koristi za njihovu izvedbu. U završnom dijelu trećeg poglavlja objašnjeni su koncepti programiranja na veliko i programiranja na malo te su navedene njihove glavne razlike. U četvrtom poglavlju opisan je raspodijeljeni sustav zasnovan na suradnji i natjecanju za čije su potrebe izvedeni komunikacijski i sinkronizacijski mehanizmi. U petom poglavlju opisani su opći i binarni semafor te spremnik poruka, prikazani su primjeri njihova korištenja i objašnjena je izvedba. Na kraju rada vrednovani su ostvareni mehanizmi i iznesen je zaključak.

2 Postojeći mehanizmi komunikacije i sinkronizacije

Operacijski sustav ima dvije glavne funkcije. Prva funkcija je skrivanje složenosti sklopolja računala od korisnika odnosno razvijatelja primjenskih programa. Skrivanje složenosti se postiže upotrebom dobro definiranog sučelja (engl. *Application Programming Interface, API*) koje omogućuje pristup sklopolju na višem nivou apstrakcije. Druga funkcija operacijskog sustava je omogućavanje djelotvornog iskorištavanja procesorskog vremena, memorije, ulazno-izlaznih naprava i ostalih računalnih sredstava. Primjenski programi raznoliko troše sredstva pa se zbog povećanja njihove iskoristivosti u operacijskim sustavima najčešće može izvoditi više primjenskih programa u istom vremenskom razdoblju. Operacijski sustavi koji omogućuju istovremeno izvođenje više programa nazivaju se višezadačni.

Primjenski programi se često mogu podijeliti na više podzadataka koji su ostvareni kao procesi. Jedan proces može čitati podatke s ulaza, drugi ih obrađivati, a treći prikazivati rezultate korisniku. Zbog toga trebaju postojati mehanizmi komunikacije između procesa, pomoću kojih će proces koji čita ulaz moći slati pročitane podatke drugom procesu na obradu, a on će obrađene podatke poslati procesu zaduženom za prikaz rezultata. Pojedini zadaci se izvode različitim brzinama pa je potrebno osigurati mehanizme sinkronizacije procesa jer se oni moraju izvršiti u točno određenom redoslijedu i ne smiju pristupati zajedničkim podacima istovremeno. Danas je većina primjenskih programa izvedena pomoću više dretvi jer je postupak stvaranja dretve nekoliko puta brži od stvaranja novog procesa. Sve dretve nekog procesa dijele zajedničku memoriju pa je komunikacija među njima lako ostvariva. Ipak, za pristup zajedničkim sredstvima dretve kao i procesi trebaju koristiti mehanizme međusobne sinkronizacije. Zbog toga su u sve operacijske sustave ugrađeni mehanizmi sinkronizacije dretvi i procesa te mehanizmi komunikacije između procesa.

S razvojem računalnih mreža počeli su se pojavljivati raspodijeljeni računalni sustavi. U njima je sinkronizacija i komunikacija procesa složenija jer se oni mogu izvoditi na različitim računalima koja često nemaju jednako sklopolje ili operacijski sustav. Zbog toga se i podaci koje razmjenjuju mogu drugačije interpretirati i raspodijeljeni programi toga moraju biti svjesni. Tako dva računala koja komuniciraju trebaju znati da li je tekst koji

razmjenjuju kodiran ASCII sustavom znakova ili UTF-8 sustavom, kakav prikaz cijelih brojeva u komunikaciji koristi drugo računalo te kako su prikazane složenije strukture podataka koji se koriste u raspodijeljenom programu. Ovakvi problemi se rješavaju uvođenjem posredničkog sustava koji predstavlja sloj između raspodijeljenog programa i operacijskog sustava. Raspodijeljenim programima on nudi mehanizme komunikacije i sinkronizacije koji su jednaki na svim računalima s posredničkim sustavom.

2.1 Mehanizmi ugrađeni u operacijske sustave

Kod implementacije složenih programskih sustava često je potrebno ostvariti suradnju dva ili više procesa. Da bi procesi međusobno surađivali moraju razmjenjivati podatke pa se kod izgradnje složenih računalnih sustava javljaju tri problema. Slanje podataka drugim procesima odnosno implementacija mehanizama komunikacije među procesima računalnog sustava je prvi problem koji se mora riješiti. Drugi problem je usklađivanje aktivnosti procesa tako da dva procesa koja se natječu za ista sredstva to ne čine istovremeno. Treći problem je osiguravanje redoslijeda izvođenja zadataka, tako da proces koji obrađuje podatke ili ih prikazuje korisniku, mora čekati da ti podaci stvarno postanu dostupni. Kod višedretvenih procesa prisutna su samo druga dva problema, a prvi nestaje jer dretve dijele zajednički adresni prostor pa je ostvarenje komunikacijskih mehanizama jednostavno.

2.1.1 Natjecanje za računalna sredstva

Kad dvije dretve (ili dva procesa) istovremeno žele pristupiti zajedničkom sredstvu one se natječu za njega. Dretvi koja je prva započela sa upotrebom nekog sredstva treba omogućiti da završi sa započetom aktivnošću prije nego što neka druga dretva može upotrijebiti isto sredstvo. Ako dvije dretve istovremeno koriste zajedničko sredstvo, npr. dijeljenu memoriju, onda je moguć slijed događaja u kojem će jedna dretva započeti s izmjenom podataka i prije nego što uspije završiti započetu aktivnost operacijski sustav može procesorsko vrijeme dodijeliti drugoj dretvi. Ona tada čitanjem sadržaja dijeljene memorije dolazi do neispravnih i nedosljednih podataka što uzrokuje neispravan rad programa. Zbog toga su potrebni mehanizmi koji će osigurati međusobno isključivanje dretvi kod pristupanja zajedničkim sredstvima.

2.1.2 Kritični odsječci

Dijelovi kôda koji pristupaju zajedničkim sredstvima nazivaju se kritični odsječci. Ostatak kôda sačinjavaju nekriticni odsječci. Da bi se riješio problem natjecanja za sredstvima (engl. *race condition*) potrebno je zadovoljiti sljedeće uvjete:

1. Dvije dretve ne smiju istovremeno biti u kritičnom odsječku.
2. Ne smiju se raditi pretpostavke o brzini rada procesora i broju procesora u računalnom sustavu.
3. Dretve koje nisu u kritičnom odsječku ne mogu zaustaviti izvođenje ostalih dretvi.
4. Sve dretve koji čekaju na ulazak u kritičan odsječak moraju i dočekati u konačnom vremenu.

Najjednostavniji način da se ostvari međusobno isključivanje je onemogućavanje prekida. Ova metoda ima i dva bitna nedostataka. Radi samo na jednoprocesorskim sustavima. Drugi nedostatak je dopuštanje svim primjenskim programima da omogućavaju i onemogućavaju prekide. Ako jedan od tih programa onemogući, a nakon toga zbog greške ponovno ne omogući prekide, onda operacijski sustav više ne može preuzeti kontrolu pa ovaj pristup nije dobar za izradu primjenskih programa. Ipak, za programiranje jezgre operacijskog sustava postupak se često koristi. Tako se mogu onemogućiti prekidi dok se obavljaju neke operacije nad jezgrinim internim strukturama.

Sljedeći pokušaj rješenja problema može biti upotreba zajedničke varijable koja signalizira da li je ulazak u kritični odsječak siguran. Ova metoda ne rješava problem već ga samo premješta na drugi dio kôda jer je pristup zajedničkoj varijabli novi kritični odsječak. Ako jedna dretva (proces) pročita sadržaj varijable i zaključi da može ući u kritičan odsječak, a operacijski sustav prekine izvođenje trenutne dretve i nastavi s drugom koja baš u tom trenutku pokuša ući u kritičan odsječak, onda će i ona zaključiti da može ući u kritični odsječak. Nakon toga će obje dretve u zajedničku varijablu zapisati vrijednost koja signalizira zabranu ulaza i nastaviti s izvođenjem pa će se u kritičnom odsječku naći obje dretve istovremeno što krši prvo pravilo. Problem se ne rješava ni uvođenjem novih varijabli, po jedne za svaku dretvu jer se i za takav pokušaj može dogoditi sličan događaj zbog kojih neki od gornja četiri uvjeta neće biti zadovoljen.

Zajednička se varijabla može pokušati iskoristiti na još jedan način. Ona može sadržavati redni broj dretve koja smije sljedeća uči u kritičan odsječak. Pri izlaska iz kritičnog odsječka dozvoljava se pristup sljedećoj dretvi. Ova metoda zadovoljava prvi uvjet jer samo jedna dretva može uči u kritični odsječak i tek na izlasku iz njega mijenja sadržaj zajedničke varijable. Ipak, ni ovo rješenje ne zadovoljava sve uvjete jer dretve koje nisu u kritičnom odsječku mogu zaustaviti izvođenje ostalih dretvi. Ako neka dretva na koju je došao red trenutno ne treba pristupiti zajedničkom sredstvu, onda će ono biti nedostupno i za ostale dretve. Tek kad dretva koja ima pravo pristupa promijeni vrijednost zajedničke varijable, ostale dretve mogu doći na red. Zbog toga najsporija dretva određuje brzinu cijelog programa što nije učinkovito. Drugi problem se javlja kad neki proces stane izvan kritičnog odsječka. Kad na njega dođe red on nikad neće uči i pravo ulaska se nikad neće dodijeliti sljedećem procesu. Zbog toga ni četvrti uvjet nije zadovoljen.

Rješenje koje zadovoljava sva četiri uvjeta ponudio je nizozemski matematičar T. Dekker. On je predložio sljedeći algoritam koji omogućuje međusobno isključivanje dviju dretvi označenih s D_I i D_J :

```

Dok je ( 1 ) {
    ZASTAVICA[I] = 1;
    Čitati varijablu ZASTAVICA[J];
    Dok je ( ZASTAVICA[J] != 0 ) {
        Ako je ( Pravo != I ) {
            ZASTAVICA[I] = 0;
            Dok je ( PRAVO != I ) {
                Čitati varijablu PRAVO;
            }
            ZASTAVICA[I] = 1;
        }
        Čitati varijablu ZASTAVICA[J];
    }
    Kritičan odsječak;
    PRAVO = J;
    ZASTAVICA[I] = 0;
    Nekritični odsječak;
}

```

Dretva D_I prvo najavi svoj ulazak u kritični odsječak postavljanjem varijable $ZASTAVICA[I]$ na 1. Nakon toga provjerava da li je dretva D_J postavila svoju zastavicu.

Ako to nije učinila, onda dretva D_I slobodno može nastaviti s izvođenjem. Inače, varijabla PRAVO određuje koja dretva ima prednost, a koja mora odustati.

Petersonov postupak je pojednostavljenje Dekkerova algoritma. Za dvije dretve označene s D_I i D_J algoritam je sljedeći:

```

Dok je ( 1 ) {
    ZASTAVICA[I] = 1;
    NE_PRAVO = I
    Pročitati varijablu ZASTAVICA[J];
    Pročitati varijablu NE_PRAVO;
    Dok je ( NE_PRAVO == I && ZASTAVICA[J] == 1 ) {
        Pročitati varijablu ZASTAVICA[J];
    }
    Kritičan odsječak;
    ZASTAVICA[I] = 0;
    Nekritični odsječak;
}

```

Zapisivanje u varijablu NE_PRAVO je nedjeljiva operacija pa ako obje dretve u kratkom vremenskom razdoblju u nju upišu svoju vrijednost, onda će ona dretva koja je zadnja upisala svoju vrijednost čekati.

Postojanje algoritama za međusobno isključivanja dvije dretve ukazuje na postojanje algoritama za međusobno isključivanje proizvoljnog broja dretvi koji zadovoljava sva četiri uvjeta. Ono se može ostvariti primjenom Lamportova protokola za proizvoljan broj dretvi. Program dretve D_I izgleda ovako:

```

Dok je ( 1 ) {
    ULAZ[I] = 1;
    Čitati ZADNJI_BROJ;
    BROJ[I] = ZADNJI_BROJ + 1;
    ZADNJI_BROJ = BROJ[i];
    ULAZ[I] = 0;
    Za ( J = 0; J < N; J++ ) {
        Čitati varijablu ULAZ[J];
        Dok je ( ULAZ[J] == 1 ) {
            Čitati varijablu ULAZ[J];
        }
        Čitati varijablu BROJ[J];
        Dok je ( Broj[I] != 0 && (Broj[J] < Broj[I] || Broj[J] ==
            Broj[I] && J < I ) {

```

```

    Čitati varijablu Broj[J];
}

}

Kritični odsječak;
BROJ[I] = 0;
Nekritični odsječak;

}

```

Na početku svaka dretva dobije broj dolaska. Ako više dretvi pokuša istovremeno ući u kritični odsječak, onda se po dodijeljenom broju dolaska odredi koja dretva ima prednost, tj. koja je došla prije. Kad više dretvi dobije jednak broj dolaska, redoslijed njihova ulaska je unaprijed određen njihovim rednim brojevima. Više dretvi može dobiti isti broj dolaska ako jedna dretva uzme broj i prije nego što uveća varijablu ZADNJI_BROJ neka druga dretva uzme broj.

Nedostatak navedenih metoda je što one troše procesorsko vrijeme dok čekaju na oslobođanje nekog računalnog sredstva. Operacijski sustavi zbog toga u svojim izvedbama mehanizama za sinkronizaciju ne dodjeljuju procesorsko vrijeme dretvama koje čekaju na računalna sredstva.

2.1.3 Binarni i opći semafor

Međusobno isključivanje procesa je riješilo problem istovremenog pristupa sredstvima, ali na taj način nije moguće odrediti redoslijed pristupanja tim sredstvima. Ako dva procesa dijele zajedničku memoriju koja služi kao međuspremnik, onda oni pomoću nje mogu razmjenjivati podatke. Prvi proces je proizvođač i on pohranjuje podatke u međuspremnik. Drugi proces čita podatke iz međuspremnika pa on ima ulogu potrošača. U općenitom slučaju proizvođač i potrošač rade različitim brzinama. Problem nastaje kada proizvođač želi pohraniti podatak u međuspremnik koji je već popunjen i kad potrošač želi pročitati podatak iz međuspremnika koji je prazan. Da bi se riješio problem pisanja u popunjenoj međuspremnik ili problem čitanja iz praznog međuspremnika, potrebno je voditi računa o redoslijedu kojim procesi pristupaju međuspremniku. Potrošač ne smije pristupiti međuspremniku ako prije njega proizvođač nije stavio neki podatak u njega. Kad je međuspremnik pun, proizvođač se privremeno mora zaustaviti sve dok potrošač iz njega ne pročita jedan ili više podataka.

Za rješenje problema proizvođača i potrošača Dijkstra je 1965. g. predložio korištenje cjelobrojnog brojača koji se naziva opći semafor. On može poprimiti cjelobrojne vrijednosti veće ili jednake 0 i manje od neke unaprijed zadane gornje granice. Postoje dvije operacije koje se mogu obaviti sa semaforom. Prva je ispitivanje semafora kojom se vrijednost brojača smanjuje za jedan ako je veća od nule. Kad vrijednost brojača postane jednak nuli, izvođenje sljedeće dretve koja pokuša ispitati semafor privremeno se zaustavlja. Druga operacija je postavljanje semafora. Ako postoje dretve čije izvođenje je privremeno zaustavljeno kad su ispitivale semafor, onda se jedna od njih nastavlja izvoditi nakon postavljanja semafora, inače se brojač uveća za jedan. Obje operacije semafora moraju biti nedjeljive pa su obje funkcije zapravo kritični odsječci. Semafor se koristi da ograniči broj dretvi koje pristupaju nekom sredstvu u jednom trenutku ili da se ograniči broj sredstava kojima u nekom trenutku jedna dretva može pristupiti. Opći semafor čiji brojač može poprimiti samo dvije vrijednosti se naziva binarni semafor. On se najčešće koristi za ostvarivanje kritičnih odsječaka. Kad dretva pristupa kritičnom odsječku ispituje binarni semafor, a kad izlazi iz kritičnog odsječka ona postavlja semafor.

Problem proizvođača i potrošača se može riješiti uvođenjem dva opća semafora. Početna vrijednost prvog semafora je veličina međuspremnika, dok je drugi semafor postavljen na 0 što označava da za potrošača još nema poruka. Prije svakog dodavanja podataka proizvođač ispita prvi semafor i ako je međuspremnik pun, onda će on čekati dok se ne oslobodi mjesto. Ako međuspremnik nije pun, onda se podatak slobodno može zapisati. Tada se postavi drugi semafor da se označi da je stigla nova poruka. Kad potrošač želi pročitati novu poruku on ispita drugi semafor. Ako je međuspremnik prazan potrošač će čekati sve dok proizvođač ne stvori novu poruku. Nakon što pročita poruku potrošač će smanjiti vrijednost prvog semafora operacijom ispitivanja. Ako je međuspremnik ostvaren kao polje fiksne veličine, onda proizvođač i potrošač nikada neće pristupati istim memorijskim lokacijama pa sam pristup međuspremniku nije kritičan odsječak. Ipak, u složenijem slučaju gdje je npr. međuspremnik ostvaren pomoću vezane liste, dodavanje i brisanje utječe i na okolne elemente pa dio kôda koji pristupa listi treba biti ostvaren kao kritični odsječak.

2.1.4 Potpuni zastoj

Sa štićenjem kritičnih odsječaka riješeni su problemi izgubljenih i neispravnih podataka, a problem redoslijeda izvođenja dretvi se može riješiti upotrebom semafora. Ipak, korištenjem navedenih metoda sinkronizacije javlja se novi problem. Ako dvije ili više dretvi koristi ista sredstva, onda se može dogoditi potpuni zastoj. Svaka dretva tada čeka na oslobađanje nekog sredstva koje koristi neka druga dretva. Dretva koja čeka ne može osloboditi sredstva koja je zauzela pa u takvoj situaciji nijedna dretva neće nastaviti s izvođenjem. Potpuni zastoj se može definirati na sljedeći način:

Za skup procesa se kaže da je u potpunom zastaju ako svaki od procesa u skupu čeka na događaj koji samo neki drugi proces iz istog skupa može uzrokovati.

Da bi se definirali uvjeti nastajanja potpunog zastoja potrebno je definirati pojам računalnog sredstva. Računalno sredstvo može biti neka ulazno-izlazna naprava, dio memorije, neki objekt operacijskog sustava (npr. datoteka, proces, itd.) ili zapis u bazi podataka. Postoje dvije vrste sredstava: ona koja se mogu oduzeti od procesa koji ih posjeduje bez ikakvih posljedica (engl. *preemptable resource*) i ona kojima se to ne može učiniti (engl. *nonpreemptable resource*). Primjer prve vrste sredstva je radna memorija računala, čiji sadržaj se može u trenutku dok se proces ne izvodi prebaciti na disk, a memorija se onda može dodijeliti nekom drugom procesu. Uredaj za snimanje DVD-ova predstavlja primjer druge vrste sredstava. Kad počne snimanje DVD-a ono se ne može prekinuti bez posljedica na uspješnost operacije snimanja.

Postupak korištenja računalnih sredstava može se podijeliti na tri koraka:

1. Zahtjev za korištenje sredstva
2. Korištenje sredstva
3. Oslobađanje sredstva

Ako sredstvo nije dostupno kad ga neki proces zatraži, onda on čeka na njegovo oslobađanje ili kao rezultat dobije kôd pogreške te nakon toga čeka neko vrijeme prije ponovnog pokušaja. Što će se točno dogoditi razlikuje se za svaki operacijski sustav i korištene sistemske pozive.

Coffman i ostali (1971.) su pokazali da za nastajanje potpunog zastoja moraju biti zadovoljena četiri uvjeta:

1. Neko sredstvo u istom trenutku može upotrebljavati samo jedan proces (dretva), tj. korištenje sredstva je međusobno isključivo.
2. Proces (dretva) drži dodijeljeno sredstvo dok čeka drugo sredstvo.
3. Procesu (dretvi) se sredstvo ne može oduzeti. On ga sam mora otpustiti na kraju korištenja.
4. Mora postojati kružni niz od dva ili više procesa (dretvi) u kojem svaki proces čeka na sredstvo koje drži sljedeći proces.

Problem potpunog zastoja može se pokušati riješiti na više načina. Najjednostavniji način je zanemarivanje problema. Iako na prvi pogled ovo izgleda kao loša praksa, često je mogućnost nastanka potpunog zastoja toliko mala da nije isplativo smanjiti učinkovitost nekog sustava pogotovo ako je mogućnost pogrešnog rada sustava zbog nekih drugih razloga puno veća od mogućnosti nastanka potpunog zastoja. Tako u većini operacijskih sustava postoji mala mogućnost nastanka potpunog zastoja ako nekoliko procesa pokušava istovremeno stvoriti nekoliko novih procesa, otvoriti nekoliko datoteka ili želi zauzeti neko sredstvo koje im nudi operacijski sustav. Pošto tih sredstava može biti samo ograničen broj postoji mogućnost nastanka potpunog zastoja.

Ipak, u većini slučajeva potreban je aktivan pristup u rješavanju problema potpunog zastoja. Ako je broj potrebnih sredstava unaprijed poznat, onda je moguće korištenjem neke od formalnih metoda kao što je bankarov algoritam (engl. *banker's algorithm*) potpuno ukloniti mogućnost nastajanja potpunog zastoja. U praksi se često ne zna kakvi će biti budući zahtjevi aplikacije za sredstvima. Tada se može pokušati ukloniti neki od četiri uvjeta nastajanja potpunog zastoja čime se i onemogućuje njegovo nastajanje.

2.1.5 Monitori

Iako su semafori vrlo korisni mehanizmi oni nisu jednostavnii za korištenje. Krivi redoslijed ispitivanja semafora može lako dovesti do potpunog zastoja. Općenito problemi koji nastaju upotrebom semafora nastaju zbog njegovih dvaju svojstava: svaki semafor ispituje samo jedan jednostavan uvjet i ispitivanje nekog semafora povezano je sa zauzećem

sredstva kojeg semafor štiti. Zbog toga nije moguće ispitati više semafora i nakon toga obaviti rezervaciju sredstava tek ako se ustanovi da su sva tražena sredstva slobodna. Te je probleme analizirao C. A. R. Hoare i predložio načine njihova rješavanja. On je ustanovio da bi bilo prikladno mehanizme suradnje dretvi objediniti u prikladne nakupine funkcija za rješavanje nekih cjelovitih problema. Cijela se nakupina nadzire objedinjenim nadzornim programom koji je on nazvao monitorom.

Monitor se sastoji od struktura podataka i funkcija koje djeluju nad strukturu podataka. Izvođenje funkcija monitora mora biti pojedinačno, tj. pozivanje funkcija je ulazak u kritični odsječak. Sve funkcije jednog monitora na svom početku ispituju poseban binarni semafor koji se naziva monitorski semafor, a po završetku one postavljaju taj semafor. Unutar monitora dretve ispituju uvjete za daljnje napredovanje. Ako su oni ispunjeni, onda dretva napušta monitor i time omogućuje ostalim dretvama ulazak u monitor. Ako uvjeti nisu ispunjeni, onda se zaustavlja izvođenje dretve unutar monitora i čeka na ispunjenje uvjeta. Zaustavljanje izvođenja jedne dretve unutar monitora popraćeno je propuštanjem sljedeće dretve u monitor, iako je zaustavljena dretva prividno ostala unutar monitora. Dretva koja je ušla u monitor može djelovati na ispunjenje nekog uvjeta zbog kojeg je prekinuto izvođenje dretve unutar monitora i na taj ju način ponovno pokrenuti pa se obje dretve mogu istovremeno naći unutar monitora. Zbog toga su monitorske funkcije ostvarene tako da se ponovno pokretanje dretve zaustavljene u monitoru obavlja uvijek na njihovom kraju pa dretva koja je ponovno pokrenuta može zauzeti sredstva bez mogućnosti nastanka potpunog zastoja. Kod oslobađanja sredstava, monitor osigurava da se ponovno pokrene najviše jedna dretva koja čeka unutar monitora.

2.1.6 Komunikacija među procesima

Dvije ili više dretvi lako međusobno razmjenjuju podatke uz pomoć odgovarajućih mehanizama sinkronizacije kao što su semafori jer se nalaze u istom adresnom prostoru. Da bi dva procesa komunicirala moraju koristiti neki mehanizam za komunikaciju ugrađen u operacijski sustav. Najjednostavniji način prenošenja podataka je zapisivanje podataka u datoteku koju onda drugi proces može pročitati. Ipak, u operacijske sustave je najčešće ugrađeno nekoliko metoda koje su prikladnije za korištenje. Jedna od najčešćih metoda koja je ostvarena u većini operacijskih sustava je korištenje dijeljene memorije. Česti mehanizmi su i repovi poruka te cijevi (engl. *pipes*).

2.1.7 Mehanizmi za komunikaciju ugrađeni u operacijski sustav Windows

Operacijski sustav Windows podržava sljedeće mehanizme komunikacije među procesima:

- Clipboard:

Clipboard je međuspremnik za podatke koji omogućuje pohranjivanje proizvoljnih podataka u jednom ili više oblika. Tako se npr. tekst može pohraniti kao formatirani i neformatirani tekst. Najčešće korisnik odabire podatke koje želi prenijeti nekom drugom procesu, te aktivno sudjeluje u samoj komunikaciji eksplicitno zadavajući naredbe procesima. Proces koji prima podatke može po želji odabrati jedan od pohranjenih formata podataka koji smatra najprikladnjim.

- COM

Primjenski programi koji koriste OLE (engl. *object linking and embedding*) imaju mogućnost korištenja podataka koji su nastali u različitim aplikacijama. Tako se npr. u tekstualni dokument može ubaciti tablica. Kad korisnik poželi urediti sadržaj tablice on to obavlja iz istog programa. Program za uređivanje teksta surađuje s tabličnim kalkulatorom koji zna obrađivati tablice i ta suradnja je ostvaren pomoću COM (engl. *Component Object Model*) tehnologije.

- Kopiranje podataka (engl. Data Copy)

Kopiranje podataka iz jednog procesa u drugi postiže se upotrebom Windows poruke WM_DATACOPY. Procesi koji komuniciraju moraju znati format podataka koje razmjenjuju. Podaci koji se šalju ne smiju sadržavati pokazivače jer u drugom adresnom prostoru oni gube svoje značenje. Da bi proces mogao koristiti ovaj mehanizam on mora posjedovati prozor (engl. *window*) i mora znati identifikator prozora (engl. *window handle*) procesa kojem se podaci šalju.

- DDE

DDE (engl. *Dynamic Data Exchange*) je protokol koji omogućuje razmjenu podataka u raznim formatima. Primjenski programi mogu koristiti DDE protokol za jednokratno slanje podataka drugim procesima ili za kontinuiranu razmjenu podataka. Formati podataka su jednaki kao i formati podataka za clipboard. Zbog toga se ova metoda komunikacije može smatrati proširenjem funkcionalnosti clipboarda. Kao i kod clipboarda korisnik je najčešće odgovoran za pokretanje komunikacije, ali se ona nakon toga može nastaviti bez izravnog sudjelovanja korisnika. Moguće je definirati i dodatne tipove podataka za komunikaciju u čvrsto povezanim aplikacijama. Za komunikaciju DDE protokolom nije nužno da se oba procesa izvršavaju na istom računalu.

- Mapiranje datoteka

Mapiranje datoteke (engl. *File Mapping*) omogućuje procesima da datoteku koriste kao dio memorije unutar njihovog adresnog prostora. Procesi mogu koristiti uobičajene operacije nad pokazivačima za pregledavanje i mijenjanje sadržaja datoteke. Kad više procesa pristupi mapiranoj datoteci svaki od njih dobije pokazivač na memoriju u svom adresnom prostoru. Zbog toga može doći do gubitka ili uništenja podataka pa procesi koji istovremeno pristupaju mapiranoj datoteci moraju koristiti neki od mehanizama sinkronizacije. Ovaj je način komunikacije među procesima učinkovit i omogućava korištenje sigurnosnih mehanizama ugrađenih u Windows operacijski sustav za kontrolu pristupa mapiranoj datoteci.

- Pretinci za poruke

Pretinci za poruke (engl. *mailslots*) omogućavaju jednosmjernu komunikaciju između procesa. Proses koji stvara pretinac je poslužitelj pretinca za poruke (engl. *mailslot server*), dok je proces koji zapisuje podatke u pretinac i na taj način šalje podatke poslužitelju korisnik pretinca za poruke (engl. *mailslot client*). Nove poruke se stavljaju na kraj spremnika i tu su pohranjene dok ih poslužitelj pretinca za poruke ne pročita. Dvosmjerna se komunikacija može ostvariti pomoću dva pretinca za poruke. Poslužitelj i korisnik pretinca za poruke ne moraju se izvoditi na istom računalu, već se mogu izvoditi i na računalima povezanim mrežom.

- Cijevi

Cijevi (engl. *pipes*) omogućuju dvosmjernu komunikaciju između procesa. Dijele se na dvije vrste: neimenovane i imenovane. Neimenovane cijevi služe za komunikaciju između dva procesa koja međusobno surađuju. Najčešće jedan proces koristi cijevi da preusmjeri svoj standardni izlaz tako da može komunicirati sa svojim roditeljskim procesom. Za dvosmjernu komunikaciju potrebno je stvoriti dvije neimenovane cijevi. Neimenovane cijevi ne omogućuju razmjenu podataka između procesa koji se ne izvode na istom računalu i procesa koji su međusobno logički nepovezani. Za komunikaciju procesa koji se izvode na različitim računalima ili za komunikaciju logički nepovezanih procesa koriste se imenovane cijevi. Poslužitelj koji stvara imenovanu cijev daje joj dobro poznato ime ili to ime na neki način dojavlji svojim korisnicima. Oni mogu čitati iz cijevi ili pisati u nju. Imenovane cijevi su korisne za komunikaciju između dva procesa za koje se u vrijeme programiranja ne zna da li će se izvoditi na istom računalu.

Dva mehanizma komunikacije u operacijskom sustavu Windows koja ovdje nisu opisana su pozivanje udaljenih procedura i komunikacijske priključnice (engl. Windows sockets) i ubrajaju se u mehanizme za komunikaciju u raspodijeljenim sustavima.

2.1.8 Mehanizmi za sinkronizaciju ugrađeni u operacijski sustav Windows

U operacijskom sustavu Windows svi objekti mogu biti u dva osnovna stanja: signaliziranom i nesignaliziranom. Zbog toga se svi objekti mogu koristiti za sinkronizaciju, ali četiri objekta su predviđena samo za tu namjenu. Objekti za sinkronizaciju su: događaj, objekt za međusobno isključivanje, opći semafor i vremenski alarm. Za svaki od objekata definirano je kad se nalazi u kojem stanju:

- Događaj

Događaj (engl. *event*) je sinkronizacijski objekt čije se stanje se može postaviti u signalizirano pozivom posebne funkcije `SetEvent()`. Postoje dvije vrste događaja. Kod prve se vrste sinkronizacijski objekt u nesignalizirano stanje vraća pozivom funkcije `ResetEvent()`. Njenim se korištenjem više dretvi

koje čekaju neki događaj može istovremeno obavijestiti. Kod druge se vrste stanje mijenja u nesignalizirano čim je jedan objekt koji čeka na događaj signaliziran.

- Objekt za međusobno isključivanje

Objekt za međusobno isključivanje (engl. *mutex*) koristi se za ostvarivanje kritičnih odsječaka. Ako se nijedan proces (dretva) ne nalazi u kritičnom odsječku, onda je objekt za međusobno isključivanje u signaliziranom stanju, inače je u nesignaliziranom stanju.

- Opći semafor

Opći se semafor (engl. *semaphore*) ostvaren u operacijskom sustavu Windows ne razlikuje od modela općeg semafora koji je prethodno opisan. On može poprimiti vrijednost od 0 do neke gornje granice koja se zadaje prilikom njegovog stvaranja. Kad je vrijednost brojača općeg semafora nula on je u nesignaliziranom stanju, inače je u signaliziranom stanju.

- Vremenski alarm

Vremenski alarm (engl. *waitable timer*) je objekt čije stanje postaje signalizirano u određenom vremenskom trenutku. Postoje dvije vrste alarma. Prva vrsta je sinkronizacijski alarm koji ostaje signaliziran dok se neka dretva koja je čekala na njemu ponovno ne počne izvoditi. Druga vrsta ostaje signalizirana dok se ne pozove funkcija `SetWaitableTimer()` kojom se ponovno pokreće mjerjenje vremena. Svaka od ove dvije vrsta alarma može još biti periodička, ako se aktivira ponovno nakon svakog isteka vremena.

Za ispitivanje da li je neki od objekata za sinkronizaciju u signaliziranom stanju koristi se nekoliko funkcija. One se mogu podijeliti u dvije skupine. Funkcije prve skupine čekaju da jedan objekt postane signaliziran, dok funkcije druge skupine mogu čekati dok barem jedan od više objekata ne postane signaliziran ili dok svi ne postanu signalizirani. Funkcije prve skupine su: `SignalObjectAndWait` i `WaitForSingleObject`. Prva funkcija postavlja objekt u signalizirano stanje i nakon toga čeka da neki drugi objekt postane signaliziran. Druga funkcija samo čeka dok objekt ne postane signaliziran. Funkcije druge

skupine su: `WaitForMultipleObjects` i `MsgWaitForMultipleObjects`. One primaju polje signalizacijskih objekata i mogu čekati dok jedan od objekata ne pređe u signalizirano stanje ili dok svi ne prijeđu. Sve ove funkcije mogu prekinuti s čekanjem nakon zadanog vremenskog razdoblja, a ostvarene su tako da dok čekaju ne troše procesorsko vrijeme.

Funkcije `WaitForSingleObjectEx`, `WaitForMultipleObjectsEx` i `MsgWaitForMultipleObjectsEx` razlikuju se od istovjetnih verzija bez sufiksa *Ex* po mogućnosti korištenja sa asinkronim pozivima funkcija. Njihovo čekanje može prekinuti završetak nekog prije započetog asinkronog poziva, npr. za obavljanje neke operacije nad datotekama. Ipak, njihova je primjena ograničena jer se većina problema koji zahtijevaju asinkrone pozive funkcija može riješiti i uvođenjem dodatnih dretvi.

Osim navedenih mehanizama koji se mogu koristiti za sinkronizaciju procesa i dretvi u *Windows* operacijski sustav ugrađen je i učinkovit mehanizam za sinkronizaciju dretvi pomoću objekata kritičnog odsječka. Za korištenje objekta tipa kritični odsječak potrebno je pozivom na funkciju `InitializeCriticalSection` inicijalizirati strukturu podataka kritičnog odsječka koja je tipa `LPCRITICAL_SECTION`. Nakon toga je za ulazak u kritični odsječak potrebno pozvati funkciju `EnterCriticalSection`, sa strukturom koja opisuje željeni kritični odsječak kao parametrom, a za izlazak funkciju `LeaveCriticalSection` sa istim parametrom.

2.2 Mehanizmi u raspodijeljenim sustavima

Umrežavanje računala i mrežni operacijski sustavi omogućili su da se sredstva dijele učinkovitije, te da se jeftino izgrade napredni računalni sustavi. Raspodijeljeni sustavi korisniku izgledaju kao virtualni uniprocesori i postojanje više računala je transparentno tj. skriveno od njega. U raspodijeljenim sustavima komunikacija među procesima je ključna komponenta svakog raspodijeljenog programa. Ona je često zasnovana na mehanizmima slanja poruka koji su ostvareni u prijenosnom sloju mreže. Njihovo je slanje složenije za izvedbu komunikacije nego jednostavan pristup dijeljenoj memoriji ili korištenje nekog drugog mehanizma ostvarenog u operacijskim sustavima pa se pojavila potreba za razvojem komunikacijskih mehanizama namijenjenih izgradnji raspodijeljenih programa. Osim komunikacije, za raspodijeljene programe bitna je suradnja između procesa i njihova

sinkronizacija. Sinkronizacija je potrebna iz istih razloga kao i kod programa koji se izvode na jednom računalu, da bi se spriječilo istovremeno pristupanje dijeljenim sredstvima, ali je to u raspodijeljenim sustavima znatno složeniji problem.

Prema OSI modelu primjenski sloj omogućuje pristup mreži na visokoj razini apstrakcije, ali to nije dovoljno za većinu raspodijeljenih aplikacija. Zbog toga se u primjenskom sloju izgrađuju posebni programi koje se nazivaju posrednički sustavi. Oni pružaju niz usluga koje olakšavaju komunikaciju i sinkronizaciju raspodijeljenih programa. Često postoje ostvarenja nekog posredničkog sustava namjenjena različitim računalnim platformama pa se raspodijeljeni programi koji koriste taj posrednički sustav mogu izvoditi na računalima s različitim operacijskim sustavima i s različitim sklopoljem.

2.2.1 Pozivi udaljenih procedura i poziv metoda udaljenih objekata

Problem komunikacije u raspodijeljenim programima dugo je postojao dok Birrell i Nelson 1984. g. nisu ponudili novi i jednostavniji način komunikacije. Njihov prijedlog su bili pozivi udaljenih procedura (engl. *remote procedure call, RPC*). Osnovna ideja je bila jednostavna: pozvati proceduru koja se nalazi na drugom računalu na jednak način kao što se pozivaju lokalne procedure. Podaci se šalju kao parametri i vraćaju kao rezultat poziva funkcije. Zbog ove jednostavnosti pozivi udaljenih procedura su se pokazali kao dobro rješenje za pisanje korisnik-poslužitelj aplikacija.

Kad neki proces pozove udaljenu proceduru on zapravo poziva lokalnu proceduru koja se naziva spojni modul (engl. *client stub*). Koristeći parametre i naziv funkcije on oblikuje poruku (engl. *parameter marshaling*) koja se prenosi preko komunikacijskog sredstva. Poruku će operacijski sustav na poslužiteljskoj strani proslijediti spojnom modulu (engl. *server stub*). On će lokalno pozvati proceduru s parametrima koje je dobio u poruci. Rezultat poziva funkcije će se nakon toga opet pohraniti u poruku koja će se poslati natrag pozivatelju koji se za to vrijeme nije izvodio. Nakon što spojni modul pozivatelja dobije odgovor on će vratiti rezultat korisniku.

Iako postupak sa strane korisnika izgleda kao poziv lokalne procedure, poziv udaljene procedure nije u potpunosti transparentan. Korisnik često mora znati da poziva udaljenu

proceduru jer će se ona dulje izvoditi zbog prevođenja parametara, oblikovanja poruka i komunikacije preko mreže. Drugi razlog su ograničenja za parametre udaljenih procedura. Tako je, npr. slanje pokazivača kao parametra udaljenih funkcija teško izvedivo. Pošto se procesi izvode u različitim adresnim prostorima to bi zahtijevalo slanje ne samo pokazivača već i podataka na koje oni pokazuju. To nekad može biti vrlo neučinkovito jer može zahtijevati slanje velike količine podataka preko mreže. Drugi problem koji se javlja pri pokušaju prenošenja pokazivača je da se u nekim jezicima kao što su C i C++ ne može znati da li pokazivač pokazuje na samo jedan objekt ili na polje objekata nepoznate duljine. U navedenim se programskim jezicima slanje pokazivača kao parametra udaljene procedure ne može ostvariti.

Objektno orijentirana paradigma je postajala sve popularnija pa se javila potreba za razvojem još jednog načina komunikacije koji je prikladniji za korištenje u objektno orijentiranim jezicima. Tako je kao nadogradnja pozivanja udaljenih procedura nastao novi način komunikacije – poziv metoda udaljenih objekata (engl. *remote object invocation*, ROI). Osnovna je ideja slična kao u pozivu udaljenih procedura. Svaki objekt osim podataka posjeduje i metode kojim može vršiti operacije nad podatcima. Te metode su dobro definirane sučeljem objekta koje može biti dostupno na različitim operacijskim sustavima i može se koristiti iz različitih programskih jezika. Ulogu oblikovanja parametara u poruku preuzeli su lokalni zastupnik (engl. *client proxy*) na korisničkoj strani i okvir objekta (engl. *server skeleton*) na strani poslužitelja.

2.2.2 Komuniciranje razmjenom poruka

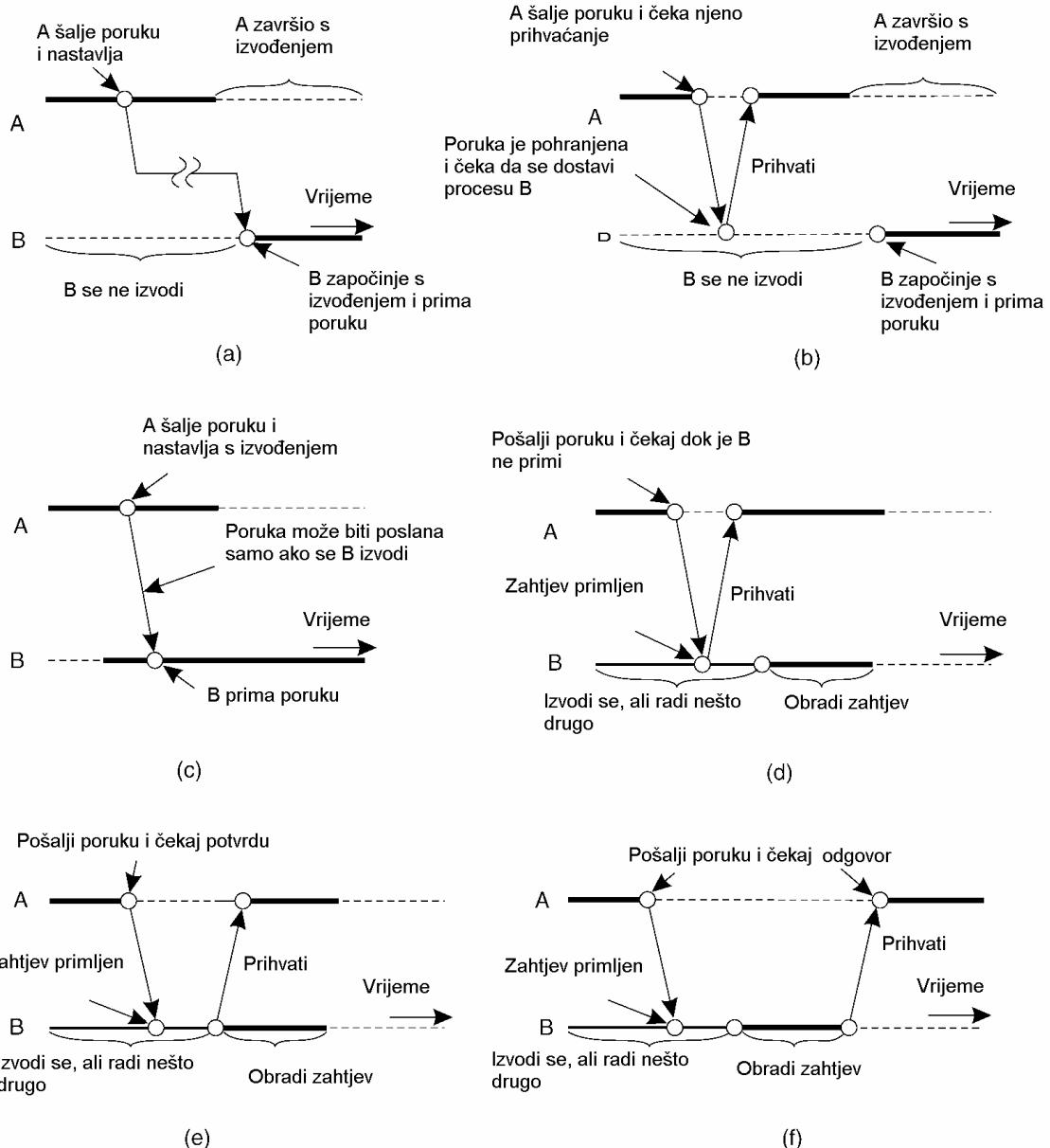
Pozivanje udaljenih procedura i pozivi metoda udaljenih objekata omogućuju transparentnu komunikaciju među procesima u raspodijeljenom sustavu. Komunikacija pomoću ova dva mehanizma je sinkrona, pa su oni pogodni za aplikacije koje koriste model korisnik-poslužitelj. U mnogim raspodijeljenim aplikacijama komunikacija ne koristi taj model, a za neke primjene je potrebna asinkrona komunikacija. Drugi problem se javlja kad se ne može pretpostaviti da se aplikacija s kojom se želi komunicirati u tom trenutku izvodi. Navedeni se problemi mogu riješiti pomoću komunikacije porukama, ali standardni mrežni protokoli, kao što su UDP i TCP, nisu dovoljni pa se uvodi posrednički sustav zasnovan na razmjeni poruka (engl. *message-oriented middleware*, MOM).

2.2.3 Vrste komunikacije porukama

Ako se primatelj poruke u trenutku slanja ne izvodi, onda poruke moraju biti pohranjene sve dok ne budu isporučene. Komunikacija u kojoj su poruke pohranjene u neki međuspremnik sve dok se ne isporuče ja postojana komunikacija (engl. *persistent communication*). Poruka se može privremeno pohraniti na korisničkom računalu ili nekom poslužitelju. Primjer postojane komunikacije porukama je elektronička pošta. Poruke koje su poslane nekom korisniku neće se izgubiti ako on ima ugašeno računalo. One se pohranjuju na poslužitelju, koji je pod nadzorom davatelja internetskih usluga, sve dok ne budu isporučene korisniku.

Drugi oblik komunikacije s porukama zahtijeva da se obje aplikacije koje međusobno komuniciraju izvode u trenutku slanja poruke. Takva komunikacija se naziva prolazna ili nepostojana komunikacija (engl. *transient communication*). Ako neki od poslužitelja ne može proslijediti poruku sljedećem poslužitelju ili primatelju, onda se poruka gubi. Zbog toga se kod nepostojane komunikacije porukama dostava ne jamči.

Osim podjele po stalnosti, komunikacija porukama može se podijeliti po sinkronosti na sinkronu i asinkronu. Sinkrona komunikacija porukama zahtijeva da se oba procesa koja komuniciraju istovremeno izvode i pošiljalac čeka na rezultat. Kod asinkrone komunikacije pošiljalac nastavlja s izvođenjem čim pošalje poruku. Kad proces primatelj primi poruku on će poslati rezultat pošiljatelju.



Slika 1: Različite vrste komunikacije porukama

U praksi postoje različite kombinacije navedenih vrsta komunikacije i različiti načini njihove izvedbe. Ukupno postoji šest kombinacija koje su prikazane na slici 1. Tako je u postojanoj asinkronoj komunikaciji poruka privremeno pohranjena u međuspremniku na posiljaocu ili nekom od poslužitelja koji se koristi za komunikaciju. Ova vrsta je prikazana na slici a). Kod postojane sinkrone komunikacije, poruka se pohranjuje samo na primateljevom računalu. Pošiljalac čeka dok se poruka ne spremi u primateljev međuspremnik. Aplikacija kojoj je poruka namijenjena u tom se trenutku ne mora izvoditi. Na slici b) prikazana je

postojana sinkrona komunikacija. Ponekad se i slučaj kad pošiljalac čeka dok se poruka ne pohrani na nekom poslužitelju izravno povezanim s primateljem svrstava u postojanu sinkronu komunikaciju.

Prolazna asinkrona komunikacija omogućuje aplikaciji koja šalje poruku da nastavi s izvođenjem čim se poruka pohrani u lokalni međuspremnik. Ako se primalac poruke ne izvodi u trenutku kad mu poruka treba biti isporučena, tada se poruka gubi. Primjer ovakve komunikacije je protokol UDP. Na slici c) je prikazan primjer ovakve komunikacije. Prolazna sinkrona komunikacija se dijeli na tri podvrste. Na slici d) je prikazan najjednostavniji oblik u kojem pošiljalac čeka dok se poruka ne pohrani u međuspremnik na računalu na kojem se izvodi proces koji treba primiti poruku. Kad se poruka kopira u međuspremnik stvara se potvrda koja omogućuje pošiljaocu da nastavi s izvođenjem. Sinkrona prolazna komunikacija je zasnovana na primitku poruke (engl. *receipt-based synchronous communication*). U sinkronoj prolaznoj komunikaciji zasnovanoj na dostavi poruke (engl. *delivery-based synchronous communication*), koja je prikazana na slici e), potvrda se šalje tek kad je proces koji se izvodi preuzeo poruku. U zadnjem obliku, na slici f), koji se zasniva na odgovoru na poruku (engl. *response-based synchronous communication*) ne šalje se potvrda već se poruka obradi i šalje se odgovor. Pozivanje udaljenih procedura i pozivi metoda udaljenih objekata koriste ovu vrstu komunikacije.

2.2.4 Prolazna komunikacija zasnovana na porukama

Mnogi raspodijeljeni sustavi koriste usluge prijenosnog sloja za ostvarivanje svoje funkcionalnosti. Za to se najčešće koriste komunikacijske priključnice (engl. *socket*) koje predstavljaju krajnju točku komunikacije u koju aplikacija može pisati podatke koji će biti poslati preko mreže ili iz koje može čitati pristigle podatke. Tablica 1 prikazuje sučelje koje priključnice izlažu. Poslužitelj poziva prve četiri metode redoslijedom kojim su navedene u tablici. Korisnik prvo stvara priključnicu, a potom šalje ili prima podatke. Kad završi s komunikacijom korisnik zatvara priključnicu. Na kraju i poslužitelj zatvara priključnicu.

| Metoda | Opis |
|---------|---|
| Socket | Stvara novu priključnicu |
| Bind | Povezuje priključnicu sa stvarnom adresom |
| Listen | Najavljuje spremnost prihvaćanja spojeva |
| Accept | Čeka na zahtjev za spajanjem |
| Connect | Pokušava uspostaviti spoj |
| Send | Šalje podatke |
| Receive | Prima podatke |
| Close | Zatvara priključnicu |

Tablica 1: Sučelje za komunikacijske priključnice

Ipak, za sustave u kojima je brzina jedno od najbitnijih svojstava, korištenje komunikacijskih priključnica je neučinkovit način komunikacije. Zbog toga se pojavila potreba za standardiziranim sučeljem koje bi omogućilo veću prenosivost primjenskih programa. Tako je nastao standard za slanje poruka nazvan MPI (od engl. *message-passing interface*). MPI pretpostavlja da se komunikacija odvija unutar poznate grupe procesa. Svakoj se grupi dodijeli identifikator koji ju jedinstveno određuje. MPI podržava sve osnovne vrste prijenosa poruka. Tako velik broj načina komunikacije odabran je da se omogući što učinkovitije izvedbe prilagođene pojedinim platformama. Tablica 2 prikazuje neke osnovne funkcije za slanje i primanje poruka definirane MPI sučeljem.

| Naziv | Opis |
|--------------|---|
| MPI_bsend | Dodaj poruku u lokalni međuspremnik za slanje |
| MPI_send | Pošalji poruku i čekaj potvrdu (točan opis ovisi o ostvarenju) |
| MPI_ssend | Pošalji poruku i čekaj dok ne počne njena obrada |
| MPI_sendrecv | Pošalji poruku i čekaj odgovor |
| MPI_isend | Pošalji pokazivač na poruku, nastavi s izvođenjem (ovisi o ostvarenju) |
| MPI_issend | Pošalji pokazivač na poruku, nastavi s izvođenjem i čekaj primitak poruke |
| MPI_recv | Primi poruku; čekaj ako nema poruka |
| MPI irecv | Primi poruku; ne čekaj ako nema poruka |

Tablica 2: Primjeri funkcija definiranih standardom MPI

Osim navedenih funkcija, MPI omogućava i ostvarivanje sinkronizacije grupe procesa i višestruko razasiljanje poruka. Ukupno je definirano preko 120 funkcija iako se većina funkcionalnosti može postići s mnogo manjim skupom. Razlog tako velikog broja funkcija je optimizacija za različite tipove platformi.

2.2.5 Sustavi zasnovani na repovima poruka

Sustavi zasnovani na repovima poruka (engl. message queueing systems) ili posrednički sustavi za razmjenu poruka (engl. message-oriented middleware, MOM) omogućuju postojanu komunikaciju zasnovanu na razmjeni poruka. Primjenski programi komuniciraju umetanjem i čitanjem poruka iz odgovarajućih repova poruka. Svaka poruka može sadržavati proizvoljne podatke u formatu koji moraju razumjeti i pošiljalac i primatelj. Posredničkom sustavu zasnovanom na razmjeni poruka potrebna je samo adresa repa u koji poruka mora biti umetnuta. Zbog toga sučelje ovih posredničkih sustava može biti jednostavno. Tablica 3 prikazuje izgled osnovnog sučelja posredničkog sustava za razmjenu poruka. Stavi (engl. *put*) je neblokirajući poziv koji predaje poruku posredničkom sustavu koji nakon toga dostavlja poruku. Dohvati (engl. *get*) čita poruku iz repa ako on nije prazan. U suprotnom slučaju zaustavlja izvođenje dok se neka poruka ne ubaci u rep. Neki sustavi dodaju i mogućnost dohvaćanja poruke određenog prioriteta ili poruke koja zadovoljava određen uvjet. Povuci (engl. *poll*) dohvaća poruku, ali za razliku od Dohvati ne zaustavlja izvođenje programa ako je rep prazan. Dojavī omogućuje da proces odredi funkciju za dojavljivanje (engl. *callback function*) koja će se pozvati kad se u rep umetne poruka.

| Metoda | Opis |
|---------|---|
| Stavi | Stavlja poruku u rep |
| Dohvati | Dohvaća poruku iz repa; čeka ako je rep prazan |
| Povuci | Dohvaća poruku iz repa; ne čeka ako je rep prazan |
| Dojavī | Postavlja zahtjev za dojavom kad se nova poruka ubaci u rep |

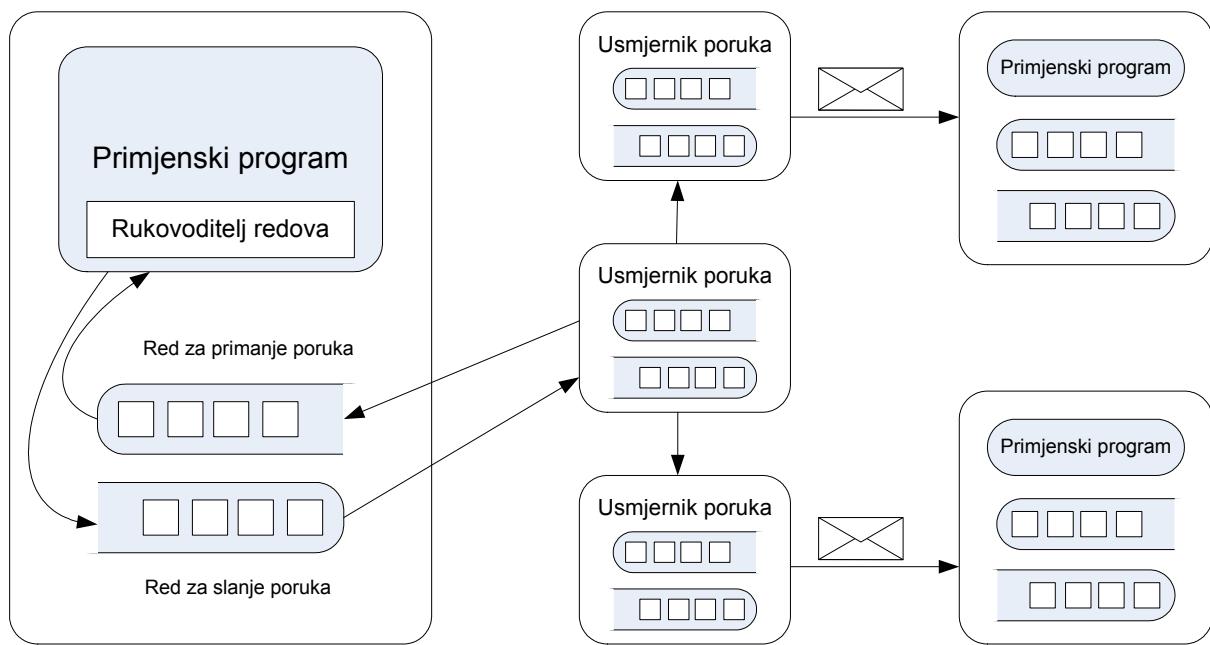
Tablica 3: Sučelje posredničkog sustava zasnovanog na razmjeni poruka

Komunikacija razmjenom poruka je pogodna za labavo-vezane sustave jer ne postoji potreba da se primatelj i pošiljalac izvode u isto vrijeme. Da bi se poruka poslala primatelj se ne mora izvoditi, a nakon što je poruka poslana, pošiljalac se više ne mora izvoditi da bi poruka bila pročitana. Posrednički sustav jamči dostavu poruke pa je komunikacija razmjenom poruka postojana. Korisniku se ne jamči da će poruka biti dostavljena unutar nekog vremenskog intervala ili čak da će biti pročitana jer to ovisi o samoj raspodijeljenoj aplikaciji.

Najčešće svaka aplikacija posjeduje svoj privatni rep u koji ostale aplikacije mogu stavljati poruke. Rep može čitati samo aplikacija kojoj je pridružen, ali moguće je i da više aplikacija dijeli jedan rep. Postupak slanja poruke počinje predajom poruke posredničkom

sustavu preko lokalnog repa (engl. *source queue*) koji onda mora iz adrese odredišnog repa (engl. *destination queue*) odrediti kamo treba poslati poruku. Zbog toga svaki rep treba imati popis imena svih repova u sustavu iz koje može odrediti mrežnu adresu na kojoj se nalazi odredišni rep.

Repovima upravljuju rukovodioci repova (engl. *queue managers*) koji mogu biti u izravnoj interakciji s raspodijeljenim programima koje koriste taj rep ili mogu imati ulogu usmjernika. Slika 2 prikazuje kako se poruka prenosi preko niza usmjernika od lokalnog repa do odredišnog repa. Lokalni rep mora znati samo adresu najbližeg usmjernika pa na taj način usmjernici tvore prekrivnu mrežu (engl. *overlay network*). Jedna od prednosti korištenja usmjernika je da ne mora postojati poseban sustav za upravljanje imenima. Svaki usmjernik zna podatke o imenima svih repova i topologiju dijela mreže. Zbog toga kod dodavanja i brisanja repova treba ispraviti odgovarajuće zapise u svakom usmjerniku. Kod većih sustava se ipak javlja potreba za dinamičkim usmjeravanjem.



Slika 2: Sustav za razmjenu poruka s usmjernicima

Kod većih sustava često se pojavljuju različiti primjenski programi koji ne mogu interpretirati sadržaj svih poruka od svih programa na ispravan način. Zadavanje formata poruka kao u prijenosnom mrežnom sloju ne rješava problem jer su ovdje poruke na višoj razini apstrakcije i prilagođene za točno određenu namjenu. Zbog toga su uvedeni posrednici

poruka (engl. *message brokers*) koji poruku prevode u format koji određeni primjenski program razumje. Često pretvorbu nije moguće obaviti bez gubitka nekog dijela informacija. Posrednici poruka često nude velik skup pravila kojima se poruka može prevoditi u drugi oblik, ali se pravila moraju ručno unositi u bazu.

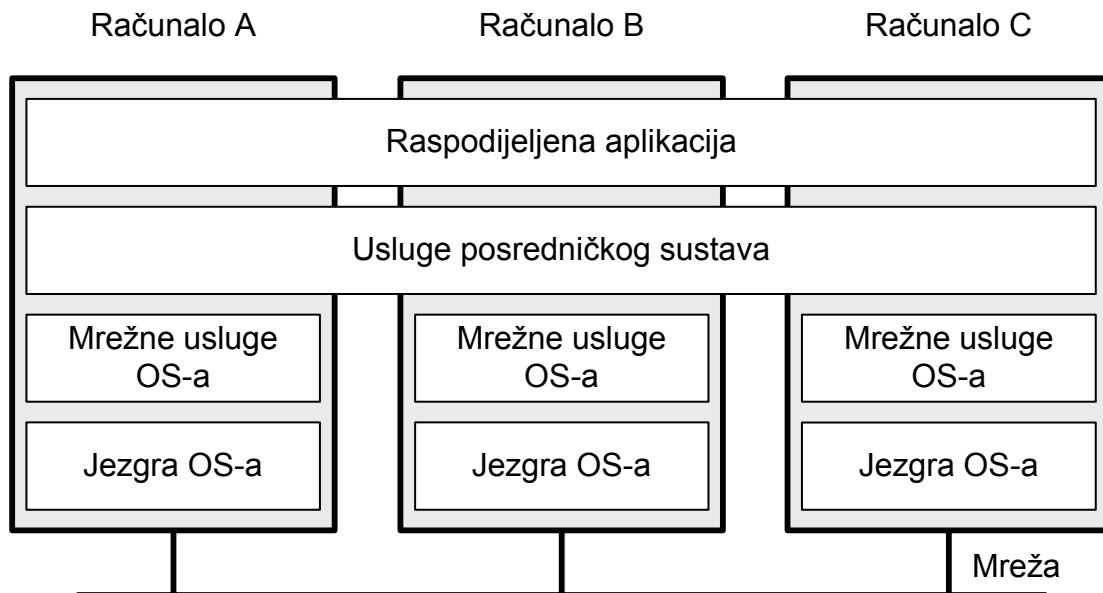
3 Računalni sustavi zasnovani na uslugama

Svojstva raspodijeljenih programa koji se koriste u sustavima za rad u stvarnom vremenu ili u znanstvenim istraživanjima se razlikuju od svojstava raspodijeljenih poslovnih sustava. Raspodijeljeni programi koji se koriste za znanstvena istraživanja ili u sustavima za rad u stvarnom vremenu izvode se najčešće na višeprocesorskim računalnim sustavima. Višeprocesorski sustavi su homogeni sustavi s dijeljenom memorijom koji se sastoje od međusobno jednakih procesora koji imaju pristup jednakoj količini privatne memorije i koji su svi međusobno povezani na jednak način. Homogeni sustavu su prilagođeni za izvođenje jednog raspodijeljenog programa koji se npr. koristi za izradu vremenskih prognoza. Druga se vrsta primjenskih programa izvodi u heterogenom sustavu najčešće sačinjenom od različitih računala povezanih različitim lokalnim mrežama ili Internetom i najčešće se koriste u poslovnim sustavima za povezivanje i suradnju poslovnih partnera. Primjenski programi koji se izvode u heterogenim sustavima zbog toga zahtijevaju visoku prilagodljivost stalnim promjenama zahtjeva i posebnu programske podršku koja od programera skriva raznorodnost sustava. Programska podrška koja zadovoljava navedena svojstva može se ostvariti programskom arhitekturom zasnovanom na uslugama (engl. *service-oriented architecture, SOA*). Računalni sustavi koji u izgradnji koriste naveden pristup nazivaju se sustavi zasnovani na uslugama.

3.1 Raspodijeljeni računalni sustavi

Raspodijeljeni računalni sustav čine nezavisna računala koja su međusobno povezana računalnom mrežom na taj način da se korisniku čine kao jedan cjelovit sustav [1]. U ovoj definiciji naglasak je stavljen na dvije činjenice. Prva je činjenica da su računala koja čine raspodijeljeni sustav nezavisna. Druga je vezana za programsku potporu, koja omogućuje da korisnici sustav koriste kao jedno računalo s virtualnim uniprocесором. Da bi raspodijeljeni sustavi podržavali raznorodna računala i računalne mreže uvedeni su posrednički sustavi (engl. *middleware*). Oni predstavljaju sloj iznad operacijskog sustava, nad kojim se gradi raspodijeljena aplikacija.

Slika 3 prikazuje organizaciju raspodijeljenog sustava pomoću posredničkog sustava. Kod izgradnje raspodijeljenih sustava bitna su četiri glavna cilja: povezivanje korisnika i računalnih sredstava, sakrivanje heterogenosti sustava, otvorenost i svojstvo razmjernog rasta.



Slika 3: Struktura raspodijeljenog sustava

Dijeljenje sredstava kao što su pisači ili sustavi za pohranu podataka je ekonomično pa je olakšavanje pristupa udaljenim sredstvima i njihovo dijeljenje s ostalim korisnicima jedan od glavnih ciljeva raspodijeljenog sustava. Ipak, s većom povezanošću rastu i sigurnosni problemi. Zbog toga se javlja potreba za autentikacijom korisnika i enkripcijom podataka. Zbog toga se takve operacije često ugrađuju u posredničke sustave.

Sakrivanjem heterogenosti sustava postiže se transparentnost. Tablica 4 prikazuje različite vrste transparentnosti sustava. Iako su svojstva navedena u tablici u većini slučajeva poželjna, nekad transparentnost nije poželjna. Tako je npr. komunikacija između dva procesa koja se izvode na različitim kontinentima barem nekoliko puta sporija od komunikacije dva procesa koji se izvode u susjednim sobama. Zbog toga je često potrebno odabrat kompromis između visokog stupnja transparentnosti i brzine izvođenja raspodijeljenih aplikacija.

| Vrsta transparentnosti | Opis |
|------------------------------|---|
| Transparentnost pristupa | Skriva razliku načinima na kojima se pristupa nekom sredstvu |
| Transparentnost lokacije | Skriva se lokacija na kojoj se sredstvo nalazi |
| Transparentnost preseljenja | Skriva se činjenica da sredstvo može mijenjati lokaciju |
| Transparentnost premještanja | Skriva se činjenica da sredstvo može mijenjati lokaciju za vrijeme korištenja |
| Transparentnost replikacije | Skrivanje činjenice da je sredstvo replicirano |
| Transparentnost istodobnosti | Skrivanje činjenice da više korisnika može istovremeno pristupati sredstvu |
| Transparentnost pogrešaka | Skrivaju se pogreške nastale u nekom sredstvu i oporavak od njih |
| Transparentnost postojanosti | Skriva da li se sredstvo nalazi u radnoj memoriji ili na disku |

Tablica 4: Različite vrste transparentnosti u raspodijeljenom sustavu (ISO 1995)

Otvorenost sustava određuje da li sustav nudi usluge na način koji je određen nekim skupom standarda i da li imaju dobro definirano sučelje. Ako sučelje nije u potpunosti definirano, onda će se različita ostvarenja istog sustava na kraju razlikovati u izvedbi nepotpunih dijelova sučelja. Prelazak s jednog na drugo ostvarenje je u takvim slučajevima otežan. U nekim slučajevima se može dogoditi i situacija u kojoj dva sustava ne mogu zajedno surađivati jer je nepotpuna definicija sustava dovela do različitog ostvarenja protokola, formata datoteka i sl.

Svojstvo razmernog rasta sustava može se mjeriti po tri dimenzije. Povećanje broja korisnika i sredstava kojima korisnici pristupaju moguće je u sustavima sa svojstvom razmernog rasta po veličini. Sustavi mogu imati svojstvo razmernog rasta po zemljopisnoj rasprostranjenosti, što znači da korisnici i sredstva mogu biti međusobno na velikim udaljenostima. Treća vrsta razmernog rasta je razmerni rast s obzirom na administrativne cjeline koje omogućuje da se sustavom može upravljati i sa porastom broja administrativnih cjelina. Povećanje razmernog rasta sustava po jednoj ili više dimenzija često uzrokuje smanjenje učinkovitosti aplikacija. Poboljšanje svojstva razmernog rasta može se postići na tri načina. Prvi je korištenje asinkrone komunikacije. Drugi način je veća raspodijeljenost aplikacije. Zadnji način je repliciranje podataka.

3.2 Raspodijeljeni sustavi zasnovani na uslugama

Za e-poslovanje (engl. *e-business*) i suradnju između poslovnih partnera (engl. *business-to-business, b2b*) potrebne su aplikacije čija brzina izvođenja, zauzeće memorije ili količina razmijenjenih podataka preko mreže nisu najvažnije karakteristike. Prilagodljivost, mogućnost prilagođavanja stalnim promjenama zahtjeva, brzina izgradnje i prilagodbe te

suradnja i integracija s drugim aplikacijama su uvjeti koje aplikacije moraju zadovoljiti. Tradicionalne programske paradigme kao što su objektno orijentirano programiranje ne mogu zadovoljiti navedene zahtjeve pa se pojavila potreba za novim načinom izgradnje aplikacija. Taj novi način je računarstvo zasnovano na uslugama (engl. *service oriented computing, SOC*). U SOC paradigmi naglasak se stavlja na arhitekturu aplikacije jer su svojstva ključnih programskih metoda koje se pojavljuju pri izgradnji komponenti aplikacije, kao što su npr. pristup bazama podataka i transakcije, većinom dobro proučena. Kod njihova korištenja za izgradnju složenijih sustava te integracije složenih sustava javljaju se problemi koje arhitektura zasnovana na uslugama pokušava riješiti.

Tablica 5 prikazuje osnovne razlike između standardne arhitekture postojećih raspodijeljenih sustava kao što su COBRA ili DCOM i arhitekture zasnovane na uslugama [7]. Usporedba pokazuje da je arhitektura zasnovana na uslugama primjerena za izgradnju aplikacija za e-poslovanje i za integriranje heterogenih aplikacija u prilagodljive i otvorene sustave.

| Standardne raspodijeljene arhitekture | SOA |
|--|---|
| Usmjereni prema funkcionalnosti | Usmjereni prema procesima |
| Dizajnirane da se ne mijenjaju | Dizajnirane s namjerom da se neprestano mijenjaju |
| Dug razvoj | Interaktivni i iterativni razvoj |
| Usmjereni prema minimalnom trošku | Usmjereni prema procesu |
| Jedna aplikacija | Orkestracija usluga |
| Čvrsto povezane | Labavo povezane |
| Homogenost tehnologije | Heterogenost tehnologije |
| Objektno orijentirano | Orijentirano na poruke |
| Poznato ostvarenje | Apstrakcija |

Tablica 5: Odnos standardne raspodijeljene arhitekture i arhitekture zasnovane na uslugama

Da bi omogućila izgradnju programske infrastrukture potrebne za poslovne sustave, arhitektura zasnovana na uslugama ima sljedeća svojstva [6]:

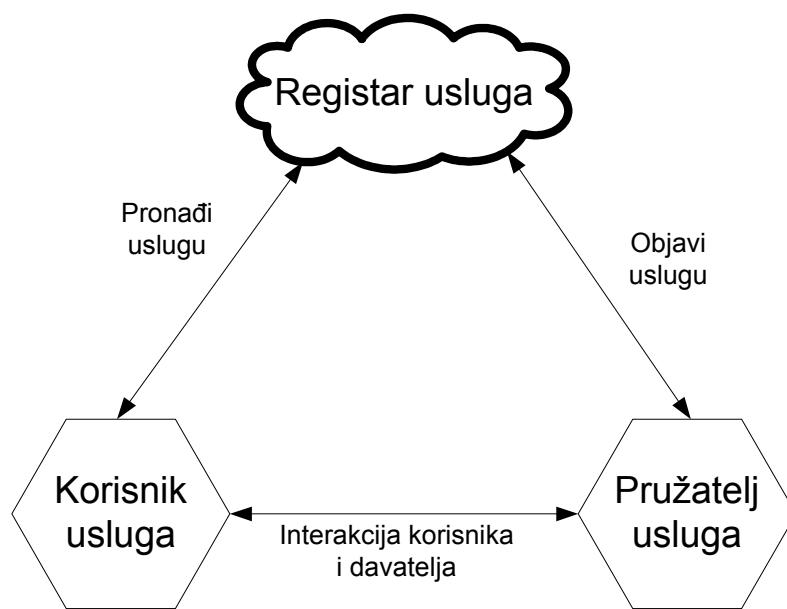
1. Poslovni procesi određuju usluge, a usluge određuju tehnologiju njihove izvedbe.
Na taj način usluge postaju apstraktički sloj između poslovnog procesa i tehnologije izvedbe. U praksi je često bilo potrebno mijenjati tehnologiju izvedbe nakon promjene zahtjeva. Razdvajanje tehnologije izvedbe od poslovnih procesa rješava ovaj problem.
2. Poslovni sustavi moraju biti agilni, a arhitektura zasnovana na uslugama im to treba omogućiti. Da bi se postigla agilnost aplikacije se trebaju moći brzo prilagođavati novim zahtjevima poslovanja.

3. Aplikacije koje koriste arhitekturu zasnovanu na uslugama neprestano se mijenjaju i nadograđuju. U tom smislu one nikad nisu gotove.

3.2.1 Načela računarstva zasnovanog na uslugama

Računarstvo zasnovano na uslugama temelji se na korištenju usluga za ostvarenje željene funkcionalnosti aplikacije. Takva arhitektura omogućuje prilagodljiv način korištenja već gotovih komponenti za pisanje novih ili proširivanje mogućnosti već postojećih aplikacija. Na slici 4 prikazan je osnovni model računarstva zasnovanog na uslugama. U njemu postoje tri vrste sudionika:

- **Pružatelj usluga:** ostvaruje uslugu i objavljuje njen opis u registar usluga
- **Registar usluga:** Sadrži opise objavljenih usluga i omogućuje korisnicima pronalaženje usluga i njihovih pružatelja
- **Korisnik usluge:** Pomoću registra usluga pronalazi pružatelje njemu potrebnih usluga i nakon toga koristi pronađene usluge.



Slika 4: Osnovna arhitektura računarstva zasnovanog na uslugama

Usluge su autonomni, cjeloviti i otvoreni dijelovi koda, neovisni o određenoj tehnologiji, platformi ili programskom jeziku. Usluge se mogu objavljivati, pretraživati i orkestrirati. Korisnici mogu koristiti usluge od različitih pružatelja usluga pa one predstavljaju osnovu za izgradnju raspodijeljenih poslovnih sustava koji se koriste unutar neke organizacije ili omogućuju suradnju nekoliko različitih organizacija. Zbog toga usluge moraju zadovoljavati tri osnovna uvjeta [8]:

1. **Neovisnost o određenoj tehnologiji:** za pozivanje usluga mora se koristiti skup prihvaćenih standarda koji se mogu koristiti na što većem broju različitih računalnih platformi.
2. **Labava povezanost:** za pozivanje neke usluge nije potrebno nikakvo znanje o njenoj unutarnjoj strukturi i tehnologiji kojom je izrađena.
3. **Podrška lokacijskoj transparentnosti:** opisi usluga trebaju se nalaziti u registru usluga i tako omogućiti određivanje lokacije usluge neposredno prije pozivanja.

Svaka usluga ima dva izdvojena dijela: dobro definirano sučelje i samo ostvarenje usluge. Sučelje je definirano korištenjem nekog općeprihvaćenog standarda kao što je WSDL, dok izvedba same usluge može koristiti proizvoljnu tehnologiju. Na taj način se zadovoljava prvi uvjet tj. neovisnost o određenoj tehnologiji izvedbe. Ipak, moraju postojati standardi za opisivanje sučelja usluge, razmjenu podataka i objavljivanje i pretraživanje registra usluga. Za razmjenu podataka danas je najprikladnija tehnologija XML, koji se šalje pomoću TCP, HTTP ili nekog drugog protokola. Danas se za komunikaciju usluga najčešće koriste protokoli SOAP zajedno s HTTP-om.

Kad je sučelje definirano na standardan način, npr. korištenjem WSDL-a i kad je definiran način razmjene podataka usluge su labavo povezane. One se mogu jednostavno povezati za vrijeme izvođenja aplikacije s uslugama koje pružaju različiti pružatelji usluga. Zbog toga je arhitektura zasnovana na uslugama prilagodljivija od čvrsto povezanih sustava koji najčešće zahtijevaju izmjenu i ponovno prevođenje kad se neka komponenta promijeni. Upotreba istog kôda je jednostavnija jer su komponente labavo vezane i neovisne jedna o drugoj. Kod ostalih programskih paradigmi je ponovno korištenje koda samo pogodnost koja ubrzava razvoj aplikacija i smanjuje broj grešaka. U računarstvu zasnovanom na uslugama kod raspodijeljenih aplikacija koje povezuju nekoliko poslovnih sustava najčešće nije moguće

zamijeniti usluge koje omogućuju suradnju s drugim poduzećima pa je ponovno korištenje istih usluga nužnost.

Korištenjem standarda za objavljivanje i pronalaženje usluga, kao što je UDDI, moguće je pronaći lokacije usluga koje aplikacija koristi. Na taj način je moguće pronaći više pružatelja usluga koji pružaju usluge s jednakom funkcionalnost pa se poduzeća mogu specijalizirati za razvoj usluga određene vrste. Tada je potrebno izabrati jednog od više mogućih pružatelja usluga. Na taj se način aplikacija može optimirati po bilo kojem parametru kvalitete usluge kao što su brzina izvođenja, pouzdanost, sigurnost i dostupnost usluge. U slučaju greške moguća je nedostupnost određene usluge i tada se može koristiti uslugu iste funkcionalnosti od drugog davatelja usluga. Zbog toga je postignuta veća otpornost na pogreške. Još jedna prednost je jednostavna zamjena s novijom verzijom usluge. Nova usluga se može odmah početi koristiti bez potrebe za ponovnom instalacijom svih aplikacija koje je koriste.

3.2.2 Podjela usluga prema složenosti

Usluge se međusobno razlikuju po stupnju složenosti i namjeni. Postoji pet vrsta usluga prema njihovoj složenosti [7]:

1. **Usluge za pristup podacima** (engl. *data acces*): omogućuju korisnicima da pristupaju, integriraju, prevode i transformiraju podatke iz različitih izvora informacija, one skrivaju pristup podacima, složenost tipova podataka i njihove transformacije u konačan oblik koji se predaje korisnicima. U arhitekturi zasnovanoj na uslugama su usluge za pristup podacima najčešće korištene usluge jer je razdvajanje logike aplikacije i načina pristupa podacima jedan od prvih koraka u izgradnji složenih programskih sustava.
2. **Usluge komponente** (engl. *component*): pružaju jednu od usluga raspodijeljene aplikacije, koje se smatraju dovoljno važnim da se pojedinačno objave. Primjer usluge komponente je usluga koja dodaje korisnika u neki sustav.
3. **Poslovne usluge** (engl. *business*): predstavljaju funkcionalnost dijelova poslovnih aplikacija. One izvode jednu ili više poslovnih operacija koje najčešće sadrže nekoliko transakcija. Mogu obavljati neki poslovni proces u cijelosti ili

mogu biti dio nekog procesa i zahtijevati suradnju s drugim uslugama za ispunjavanje njihove funkcionalnosti.

4. **Složene usluge** (engl. *Composite*): predstavljaju usluge ili aplikacije koje koriste funkcionalnost dvije ili više usluga iz prethodnih skupina. Omogućuju dodavanje logike postojećim uslugama i njihovo spajanje za izvođenje složenijih operacija.
5. **Dijeljene infrastrukturne usluge** (engl. *Shared or Enterprise Infrastructure Services*): predstavljaju usluge niže razine koje se koriste u većini novih aplikacija ili usluga i njihova upotreba ubrzava izgradnju aplikacija. Dijele se u četiri skupine:
 - Dijeljene aplikacijske usluge
 - Usluge za slanje poruka i posredničke usluge
 - Portali
 - Dijeljene poslovne usluge

Primjeri navedenih usluga su: registri usluga, sigurnosne usluge, usluga za stvaranje jedinstvenih imena ili identifikacijskih oznaka.

3.2.3 Zrnatost i suradnja usluga

Korištenje novih tehnologija često zahtijeva ponovno ostvarivanje velikih dijelova kôda. Korištenjem usluga moguće je izložiti sučelja postojećih aplikacija i omogućiti pristup funkcionalnosti koju su aplikacije pružale. Usluge koje izlažu funkcionalnost aplikacija mogu se koristiti u dalnjem unapređivanju i nadogradnji sustava. Na taj način se prijelaz na novu arhitekturu može ostvariti postupno. Pojedini se dijelovi naslijedenog kôda koji su dostupni kroz korištenje usluge nakon toga mogu mijenjati bez utjecaja na ostatak sustava.

Usluge se prema njihovoj funkcionalnosti i količini podataka koje šalju i primaju dijeli na *usluge sitne zrnatosti*, *usluge krupne zrnatosti* i *složene usluge*. Usluge sitne zrnatosti imaju minimalnu funkcionalnost te šalju i primaju male količine podataka. Usluge krupne zrnatosti ostvaruju složenije funkcije i razmjenjuju velike količine podataka. Složene usluge koriste usluge sitne i krupne zrnatosti za ostvarenje cjelovitih procesa.

Usluge sitne zrnatosti nisu pogodne za izravno korištenje u raspodijeljenim aplikacijama. Aplikacije sačinjene kao kompozicija usluga sitne zrnatosti moraju obavljati veliki broj poziva koji šalju samo male količine podataka pa su neučinkovite jer većinu poslanih podataka čine zaglavja protokola korištenih za komunikaciju, a ne aplikacijama stvarno korisne informacije. One su složene za održavanje jer su sačinjene od velikog broja usluga složenih međudjelovanja pa se greške u njima teže otkrivaju i ispravljaju. Zbog toga se usluge sitne zrnatosti najčešće koriste u izgradnji složenih usluga. Prednost korištenja usluga sitne zrnatosti je bolja kontrola pristupa i drugih sigurnosnih ograničenja pa su one najčešće dijeljene infrastrukturne usluge.

Za postizanje interoperabilnosti između aplikacija, arhitektura zasnovana na uslugama mora zadovoljiti sljedeće uvjete [5]:

- **Labava povezanost.** Labavo povezane usluge mogu se dinamički povezivati s drugim uslugama bez znanja o njihovom ostvarenju. U sustavima sačinjenim od labavo povezanih usluga ne vrijede transakcijska svojstva jednoznačnosti podataka i stanja tijekom međudjelovanja više usluga, ali su dopušteni ugovorni odnosi više razine koji određuju interakcije među pojedinim komponentama sustava da bi se postigla jednoznačnost na razini cijelog sustava.
- **Platformska neutralnost.** Sučelje svake usluge odvojeno je od ostvarenja usluge pa ostvarenje nije ovisno o određenoj platformi ili programskom jeziku. Ovisnost o nekom programskom jeziku onemogućuje suradnju aplikacija napisanih u različitim jezicima.
- **Prilagodljiva konfigurabilnost.** Sustavi zasnovani na uslugama se mogu prilagođavati uvjetima za vrijeme izvođenja pa se njihova konfiguracija može mijenjati dinamički bez gubitka korektnosti.
- **Postojanost.** Usluge ne moraju nužno imati dugačko vrijeme života, ali moraju biti spremne na obradu iznimaka. Usluge trebaju biti sposobne otkriti sve bitne iznimke, obraditi ih na odgovarajući način te poduzeti odgovarajuće akcije kad druge usluge obrade neku iznimku.

- **Krupna zrnatost.** Usluge bi trebale biti krupne zrnatosti ili složene usluge i obuhvaćati dijelove sustava na višoj razini apstrakcije. Korištenje usluga sitne zrnatosti stvara veću ovisnost među sustavima i povećava broj poruka koje se razmjenjuju.
- **Suradnja usluga.** Usluge trebaju biti ostvarene tako da mogu surađivati s drugim uslugama u rješavanju nekog problema. Usluge služe za izgradnju raspodijeljenih sustava pa centraliziranost nije njihova poželjna osobina.

3.2.4 Svojstva arhitekture zasnovane na uslugama

Korištenje arhitekture zasnovane na uslugama zahtijeva ispunjavanje određenih uvjeta, ali i donosi brojne prednosti. U nastavku su navedene neke osobine arhitekture zasnovane na uslugama i prednosti koje one donose:

Labava povezanost

- Povećava se organizacijska agilnost. Poduzeća mogu jednostavno koristiti usluge za stvaranje novih i modificiranje postojećih aplikacija.
- Povećanjem ponovne upotrebljivosti smanjuju se troškovi ostvarenja sustava jer se usluge mogu jednostavno koristiti u nekoliko aplikacija.
- Pružatelj i korisnik usluga neovisni su jedan o drugome jer su detalji ostvarenja usluga koje pružatelj nudi sakriveni od korisnika.

Korištenje općeprihvaćenih standarda

- Platformska neovisnost omogućuje poduzećima da koriste programsku podršku i računalno sklopovlje po svom izboru.
- Korisnici općeprihvaćenih standarda ne mogu postati ovisni o jednoj platformi i jednom dobavljaču programske podrške i računalnog sklopovlja.
- Olakšana je edukacija pa je veći broj stručnjaka koji poznaje te tehnologije.

Modularan pristup

- Omogućen je postupan razvoj i lakše održavanje. Spriječen je rizičan razvoj velikih monolitnih projekata.
- Pristup podijeli pa vladaj omogućuje brži i jednostavniji razvoj aplikacija.
- S vremenom se povećava broj ponovno iskoristivih komponenti.

Nenametljivost

- Omogućuje iskorištavanje postojeće programske podrške.
- Uklanja potrebu za prepravljanjem i testiranjem postojeće programske podrške

Arhitektura zasnovana na uslugama zahtjeva korištenje odgovarajuće infrastrukture i dodatne napore u dizajniranju i ostvarenju aplikacija. Zbog toga ona nije poželjna u sljedećim slučajevima [7]:

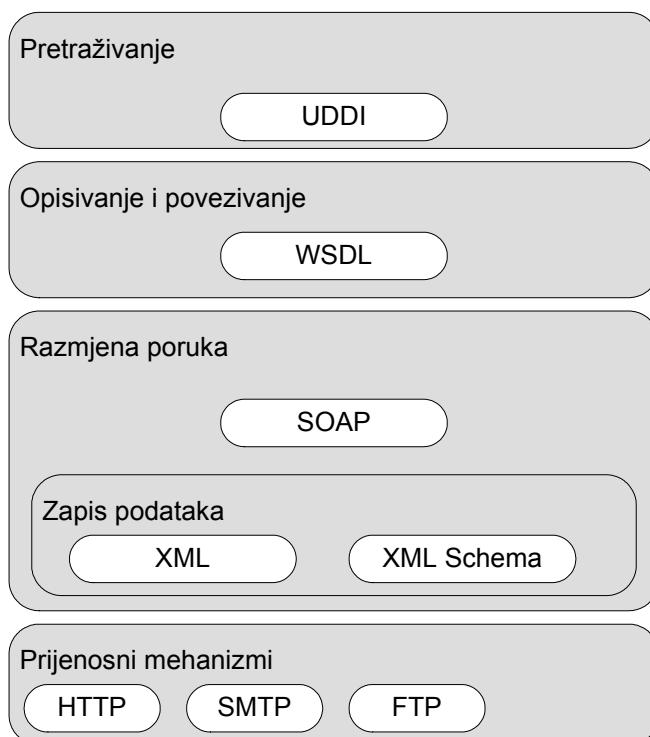
- Za izradu samostalnih i neraspodijeljenih aplikacija koje ne zahtijevaju integraciju s postojeći komponentama i aplikacijama. Primjer neraspodijeljene aplikacije je program za obradu teksta.
- Za prijelazne aplikacije koje se planiraju koristiti kratkoročno i koje se neće koristiti kao polazište za izradu budućih rješenja.
- Za aplikacije koje koriste jednosmjernu asinkronu komunikaciju za čiju je izvedbu labava povezanost nepotrebna ili nepoželjna.
- Za izradu aplikacija u homogenoj okolini u kojoj sva računala i postojeći programi koriste istu platformu.
- Aplikacije koje zahtijevaju grafički bogato sučelje prema korisniku.

3.2.5 Osnovna Web Services arhitektura

Korištenje arhitekture zasnovane na uslugama zahtjeva interoperabilnost usluga od različitih davatelja usluga pa su potrebni općeprihvaćeni standardi za komunikaciju,

objavljivanje i interakcije s mrežnim uslugama. Web Services stoga protokola je skup standarda koji omogućuje komunikaciju, objavljivanje i interakciju usluga i koji je prihvачen od vodećih poduzeća kao što su IBM, Microsoft, Oracle i drugih. Izgradnja mrežnih usluga korištenjem Web Services stoga protokola zasniva se na osnovnoj arhitekturi prikazanoj na slici 5.

Na dnu stoga se nalaze prijenosni mehanizmi. Oni su zaduženi za prijenos podataka između pružatelja, korisnika i registra usluga. Mrežne usluge koriste postojeće mehanizme prijenosa podataka preko mreže pa se odabir prijenosnog mehanizma zasniva na posebnostima aplikacije koja će ga koristiti. Zbog svoje sveprisutnosti, u praksi se najčešće koristi HTTP, ali mogu se koristiti i drugi protokoli kao što su SMTP, FTP. Osim otvorenih standarda komunikaciju je moguće ostvariti i nekim vlasničkim protokolom kao što su MQSeries ili CORBA, ali često se i tada koristi HTTP za povezivanje dijelova sustava koji komuniciraju različitim protokolima.



Slika 5: Stog protokola standarda Web Services

Sljedeći sloj predstavlja tehnologije koje se koriste za formatiranje podataka u poruke koje se razmjenjuju. Ovaj sloj koristi protokole za prijenos podataka iz prethodnog sloja i neovisan je o njima. XML se u zadnjih nekoliko godina potvrđio kao glavna i općeprihvaćena tehnologija za formatiranje podataka. Ipak, sam XML nije dovoljan za razmjenjivanje poruka

pa je danas u upotrebi protokol SOAP koji omogućuje razmjenu poruka odnosno pristup mrežnim uslugama.

Sloj za opis mrežnih usluga je sljedeći sloj i on korisniku usluge pruža sve informacije potrebne za njeno pozivanje. Ovaj sloj je najvažniji za ostvarivanje svojstva labave povezanosti i zajedno sa prethodnim slojevima odvaja ostvarenje usluge od njezinog korištenja. Postoje dvije vrste opisa. U jednom je cilj opisati jednu mrežnu uslugu, a u drugom odnose i međudjelovanja više mrežnih usluga. Sučelje usluge opisuje se jezikom WSDL.

Najviši sloj služi za pronalaženje usluga. Korisnici usluga pretražuju registar usluga koji sadrži pružatelje usluga, opise usluga i neke dodatne informacije. Za ostvarenje registra usluga i njegovo pretraživanje koristi se UDDI specifikacija.

3.2.6 XML

XML (engl. *eXtensible Mark-up Language*) je proširivi jezik zasnovan na oznakama. To je tekstualni i platformski neovisan format zapisivanja podataka. XML je jezik koji se koristi za stvaranje samoopisujućih i modularnih dokumenata, programa ili drugih jezika. XML dokumenti često se koriste za razmjenu podataka između različitih sustava i računalnih platformi. Za opisivanje podataka XML koristi oznake, omeđene simbolima < i >, koje opisuje podatke koje slijede. Kraj podataka koristi se oznakom koju prethodi znak / koja je također omeđena simbolima < i >. Podaci okruženi odgovarajućim parom oznaka čine XML element. XML je strukturirani jezik pa svaki element može sadržavati druge elemente ili nestrukturirane podatke. Svaka oznaka može sadržavati i svojstva koja dodatno opisuju element.

XML se temelji na korištenju skupa oznaka koje čine rječnik (engl. *vocabulary*). Za razliku od HTML-a koji ima fiksno definiran rječnik, XML-ov rječnik je ovisan o namjeni i proširiv je. Još jedna razlika u odnosu na HTML-om je svojstvo da XML opisuje podatke i ne definira njihov prikaz. Za prikazivanje sadržaja XML dokumenata se koriste druge tehnologije kao što su XSLT ili CSS.

Primjenom XML-a nastaje velik broj različitih rječnika oznaka pa se u njima mogu pojaviti oznake i svojstva jednakog imena i različitog značenja. Zbog toga je bio potreban način da se uredi korištenje imena i to je učinjeno prostorima imena (engl. *namespaces*). Prostor imena je jedinstveno određen pomoću URI-a (engl. uniform resource identifier). Svako ime je dodijeljeno nekom prostoru imena pa mora biti jedinstveno samo unutar njega.

Za određivanje ispravnosti sadržaja XML dokumenta koriste se svojstva dobre uređenosti (engl. *well-formed*) i valjanosti (engl. *validity*). Dokument je dobro uređen ako slijedi pravila odnosno sintaksu XML jezika koja definira način stvaranja i korištenja oznaka i svojstava. Svaka početna oznaka mora imati završnu oznaku, a dokument može imati samo jednu korijensku oznaku. Dobro definirana sintaksa XML jezika je bitna zbog prevođenja i obrade XML dokumenata te izgradnje XML jezičnih procesora neovisnih o semantici dokumenata koje obrađuju. Drugo bitno svojstvo XML dokumenta je valjanost. Dokument je valjan samo ako je uređen i ako ispunjava dodatna svojstva o tipu elemenata koji se pojavljuju u njemu, njihovu poretku te opsegu njihovih vrijednosti.

Svojstva za opisivanje valjanosti su specifična za svako područje primjene XML-a pa se javila potreba za definiranje tih svojstava. Jedan od načina na koji se svojstva valjanosti XML jezika mogu opisati je dodavanje dodatnog DTD dokumenta koji omogućuje definiranje pravila za određivanje redoslijeda, učestalosti i tipova elemenata koji se pojavljuju. DTD ima i nekoliko nedostataka jer je nastao prije XML-a pa ne koristi njegovu sintaksu. Osim toga ne podržava definiranje tipova i prostora imena, a XML dokument može koristiti samo jedan DTD dokument koji ga opisuje. Problem je riješen primjenom jezika XML Schema. On ima određen broj unaprijed definiranih podatkovnih tipova kao i mogućnost definiranja novih tipova, podržava prostore imena te omogućuje određivanje ograničenja nad vrijednostima nekog tipa.

3.2.7 SOAP

Kod integracije različitih programskih sustava nedostatak standardnog protokola predstavljao je najveći problem. SOAP je nastao kao protokol koji rješava taj problem i postao je općeprihvaćeni standard podržan od vodećih poduzeća kao što su Microsoft, IBM, BEA i drugi. Protokol se prvo primjenjivao za pozivanje udaljenih procedura pomoću XML-a (XML-RPC), ali se povećanjem popularnosti mrežnih usluga počeo koristiti za pozivanje

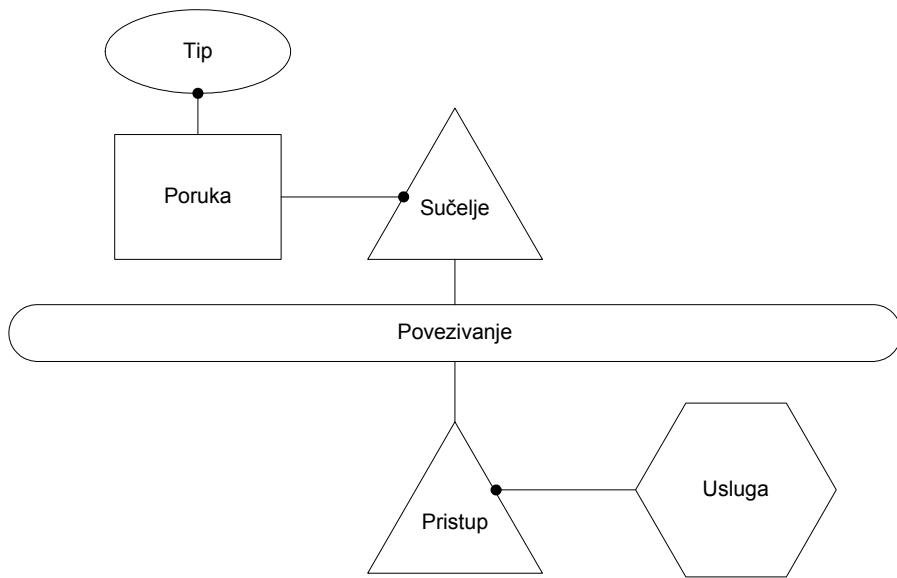
mrežnih usluga. Njegov naziv je u početku označavao da je to *jednostavan protokol za pristup objektima* (engl. *simple object access protocol, SOAP*), ali njegova se primjena ne ograničava na objektno-orientirane sustave pa je skraćenica neslužbeno dobila novi naziv *pristupni protokol zasnovan na uslugama* (engl. *service-oriented access protocol*).

SOAP je protokol neovisan o programskom jeziku i platformi, a za prijenos preko mreže može koristiti različite protokole kao što su HTTP, SMTP, FTP i druge. Zasnovan je na XML-u, a ne na nekom vlasničkom binarnom protokoli, kao što su DCOM i CORBA. SOAP povezuje sustave, ali ne omogućuje pronalaženje, pretraživanje i opisivanje usluga. Za to se koriste drugi protokoli kao što su WSDL i UDDI.

SOAP protokol definira omotnicu pomoću koje se izmjenjuju poruke i pozivaju udaljene procedure. Svaka omotnica sadrži tijelo, a može sadržavati i proizvoljan broj zaglavlja. Zaglavljia omogućuju da SOAP protokol bude proširiv bez narušavanja osnovne definicije protokola. Zaglavljje je prvi element omotnice i najčešće se koristi za usmjeravanje, autentifikaciju i slična proširenja funkcionalnosti.

3.2.8 WSDL

SOAP protokol omogućuje komunikaciju s uslugama, ali ne definira sučelje usluge ni format poruka koje se razmjenjuju. Za definiranje sučelja mrežnih usluga koristi se WSDL (engl. *web service description language*). WSDL dokument je dobro uređen XML dokument koji opisuje metode koje mrežna usluga nudi, njihove ulazne i izlazne parametre, tipove tih parametara, adresu usluge te protokole kojima se može pristupiti usluzi. Nadalje, WSDL definira ugovor između pružatelja i korisnika usluge te omogućuje odvajanje sučelja usluge od njenog ostvarenja.



Slika 6: Elementi WSDL dokumenta

Na slici 6 su prikazani osnovni elementi WSDL dokumenta i njihovi međuodnosi.

Postoji šest osnovnih elemenata WSDL:

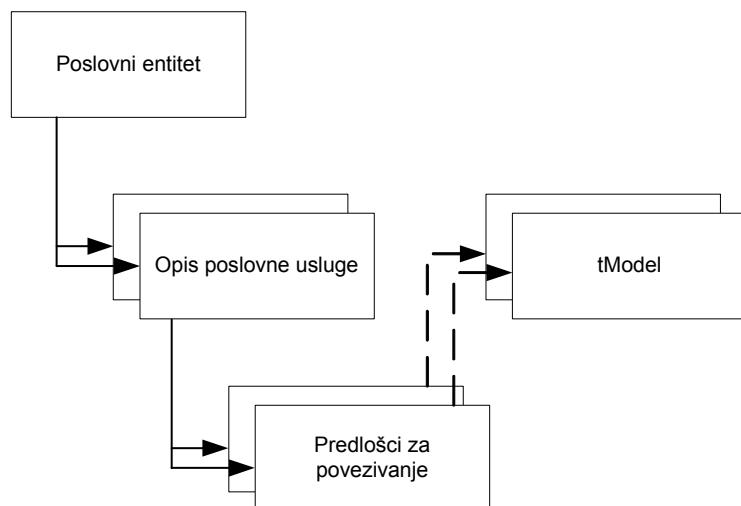
- Poruka (element Message) predstavlja jedan podatak koji se prenosi u komunikaciji između korisnika i mrežne usluge. Za svaku se poruku definira njeni ime i dijelovi od kojih svaki predstavlja jedan ulazni parametar ili jednu povratnu vrijednost. Ovaj element ne definira da li je poruka ulazna ili izlazna.
- Sučelje (element Port Type) opisuje jednu ili više operacija koje objekt javno izlaže. Sve operacije su opisane elementima tipa Operation. Svaka operacija odgovara deklaraciji metoda koje izlaže neki objekt u programskim jezicima kao što su C++, C# ili Java. Svaka operacija ima svoje ime i sadrži popis poruka. Za svaku se poruku definira da li je ulazna ili izlazna.
- Tip podatka (element Type) služi za definiranje tipova koji se koriste za oblikovanje poruka. Ovaj je element zapravo XML Schema dokument koji je umetnut u WSDL.
- Povezivanje (element binding) za svaki element sučelja određuje jedno njegovo ostvarenje pomoću nekog protokola, npr. SOAP protokola pa jedna usluga može imati više ostvarenja za različite protokole. Za svako od ostvarenja mora postojati odgovarajući zapis u elementu za povezivanje.

- Pristup usluzi (element Port) je krajnja točka na kojoj je dostupno ostvarenje sučelja usluge, a definirana je s određenim načinom povezivanja pomoću elementa binding i mrežnom adresom.
- Usluga (element Service) sastoji se od skupa pristupa usluzi.

WSDL dokument odvaja apstraktnu definiciju sučelja od stvarnog ostvarenja usluge. Elementi koji opisuju poruke, operacije, tipove i sučelje predstavljaju apstraktni dio dokumenta, dok elementi koji opisuju uslugu i pristup usluzi predstavljaju podatke o stvarnom ostvarenju. Element za povezivanje spaja ova dva dijela WSDL dokumenta.

3.2.9 UDDI

UDDI (engl. universal description, discovery and integration) specifikacija omogućuje opisivanje usluga koje pružatelji usluga nude korisnicima, te omogućuje njihovo pronalaženje i suradnju s drugim uslugama. Ariba, IBM i Microsoft su razvili prvu verziju specifikacije koja je ubrzo postala općeprihvaćena. UDDI se sastoji od same specifikacije i ostvarenja UDDI poslovnog registra usluga (engl. Business Registry) kojeg su 2001. g. pokrenuli Microsoft i IBM. Registr svatko može koristiti za objavljivanje svojih usluga i pronalaženje potrebnih usluga.



Slika 7: UDDI strukture podataka

Specifikacijom UDDI definiraju se informacije koje se čuvaju o pojedinoj usluzi u registru te programsko sučelje kojim se postavljaju upiti i ažuriraju informacije u registru. Slika 7 prikazuje četiri osnovne strukture podataka u UDDI specifikaciji:

- Poslovni entiteti (struktura business entity) sadrže informacije o poduzeću koje objavljuje usluge, kao što su ime poduzeća, opis usluga koje nudi i kontakt informacije.
- Opis poslovne usluge (struktura business service) nudi više informacija o pojedinoj usluzi iz registra. Jedno poduzeće može nuditi više usluga.
- Predlošci za povezivanje (struktura binding template) opisuju pristup usluzi koji npr. može biti: mailto, http, ftp i fax.
- Model (struktura tModel) opisuje protokole koje usluga koristi. Koristeći ovu informaciju korisnik može vidjeti koje usluge su mu dostupne.

Programsko sučelje za pristup registru ostvareno je pomoću protokola SOAP. Postojeće sučelje nudi upite za nekim od prethodno opisanih tipova podataka te postupke stvaranje, mijenjanja i brisanja zapisa iz registra. Primjer SOAP omotnice poziva za dohvaćanje predloška za povezivanje prikazan je na slici 8.

```
POST /get_bindingDetail HTTP/1.1
Host: http://www.someoperator.org
Content-type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: ""
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns=http://schemas.xmlsoap.org/soap/envelope/>
<Body>
    <get_bidingDetail generic="1.0" xmlns="urn:uddi-org:api">
</Body>
</Envelope>
```

Slika 8: Primjer UDDI poziva

Informacije u registru se mogu podijeliti u tri grupe:

- *Bijele stranice* (engl. *white pages*): sadrže ime i kontaktne detalje pružatelja usluge
- *Žute stranice* (engl. *yellow pages*): sadrže razredbu usluga prema vrsti poslovanja
- *Zelene stranice* (engl. *green pages*): sadrže tehničke informacije o uslugama

3.3 Načela programiranja na veliko i programiranja na malo

S razvojem sve složenijih i većih aplikacija postojeće metode, jezici i paradigme su se pokazale nedovoljnima. Počela se pojavljivati razlika između postupaka podjele aplikacije na module, opisivanja interakcije tih modula i njihova sučelja te samog ostvarenja pojedinih modula. Određivanja programskih modula i njihovih međudjelovanja te programiranje funkcionalnosti modula su postupci izgradnje programa različite razine apstrakcije. Programiranje na visokoj razini apstrakcije naziva se *programiranje na veliko* (engl. *programming in the large*), a programiranje na niskoj razini apstrakcije naziva se *programiranje na malo* (engl. *programming in the small*).

Programiranje na malo je pisanje kôda koje najčešće obavlja jedan programer ili manja grupa programera. Oni koriste programske jezike kao što su C++ ili C# da ostvare pojedine module i komponente aplikacije. Kôd ostvaren programiranjem na malo razumljiv je bez dodatne dokumentacije koja objašnjava kako kôd radi, za sve programere upoznate s namjenom kôda, odgovarajućim programskim jezikom i algoritmima upotrebljenim u kôdu. Vještina programiranja na malo obuhvaća pisanje što razumljivijeg, elegantnijeg i bržeg koda koji ispunjava svoju namjenu bez grešaka i kojeg je kasnije lako mijenjati i nadograđivati. Pogreške koje nastaju ovim načinom programiranja ograničene su na manji dio aplikacije i njihovo otkrivanje i ispravljanje najčešće ne predstavlja veći problem.

Programiranje na veliko je programiranje koje obavlja veća skupina programera ili manje grupa kroz dulje vremensko razdoblje. Kôd koji se dobije nije moguće u potpunosti razumjeti bez korištenja tehnike podijeli pa vladaj. Zbog toga je kod programiranja na veliko naglasak stavljen na podjelu posla na module i definiranje njihove interakcije. Određivanje programskih modula se značajno razlikuje od pisanja kôda za ostvarenje funkcionalnosti modula. Nakon podjele na module svaki je modul moguće ostvariti neovisno o ostalim

modulima i pritom je moguće koristiti programiranje na malo ili programiranje na veliko ovisno o složenosti pojedinog modula. Pogreške u podjeli posla na module ili u sučeljima pojedinih modula teško se ispravljaju kad je ostvarivanje tih modula već započeto jer izmjene najčešće nisu ograničene samo na jedan modul. Ponekad je zbog greške u definiranju jednog modula potrebno nekoliko modula ponovno dizajnirati i ostvariti. Osnovne vještine programiranja na veliko su mogućnost apstrahiranja, podjela sustava na module i definiranje interakcije između takvih modula.

Zbog velike razlike između postupaka programiranja na veliko i programiranja na malo isti alati i jezici nisu jednak pogodni za obje namjene. U arhitekturi zasnovanoj na uslugama ova činjenica ne predstavlja problem jer se pojedine usluge mogu ostvariti koristeći jedan programski jezik, a nakon toga se čitava aplikacija može ostvariti korištenjem usluga u nekom drugom jeziku. Postupak podjele aplikacije na usluge i podjela složenih usluga na module se obavlja programiranjem na veliko dok se ostvarenje jednostavnih usluga i modula obavlja programiranje na malo.

4 Okolina raspodijeljenog sustava zasnovanog na uslugama

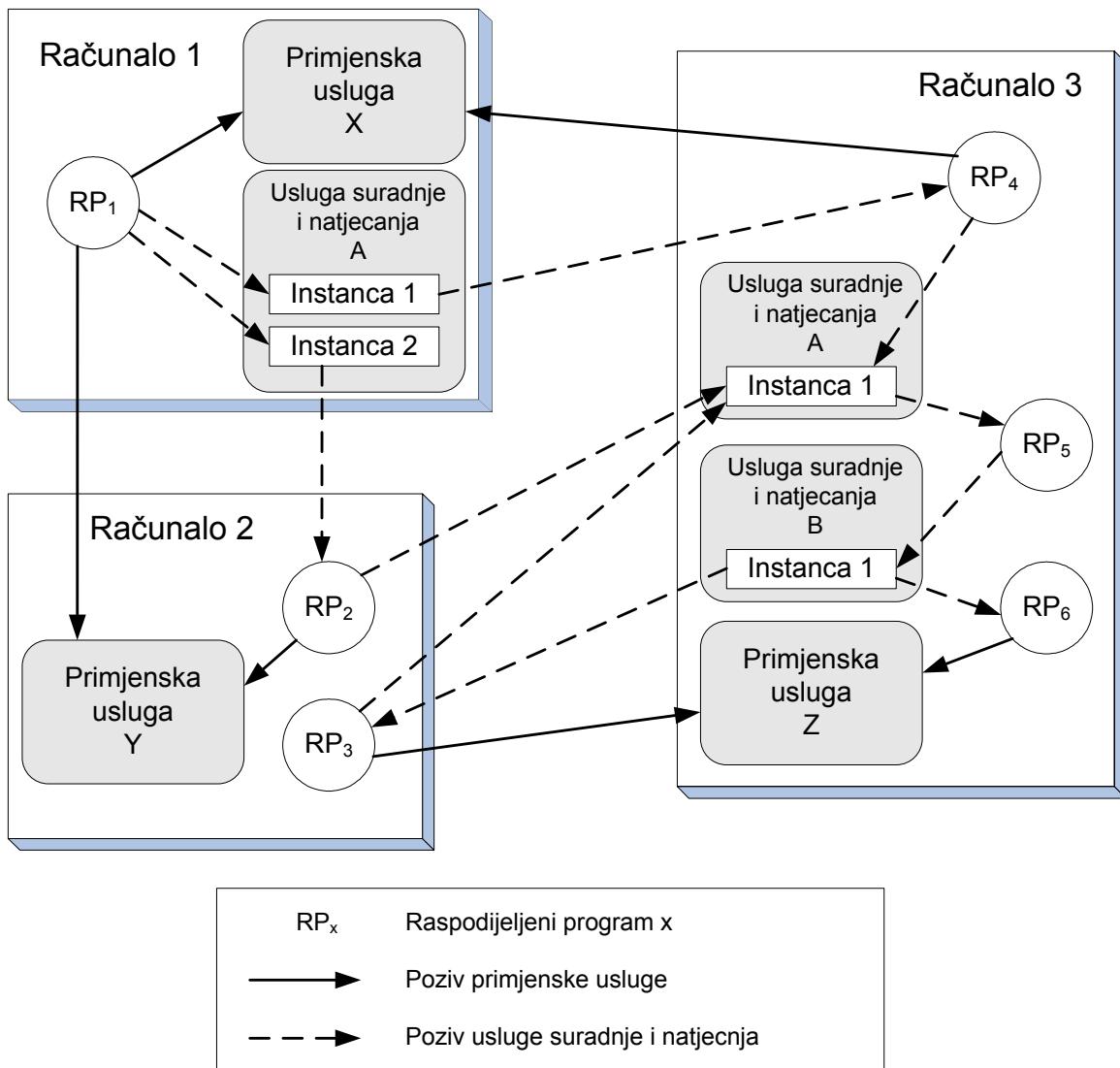
Računarstvo zasnovano na uslugama ima sve veću važnost u izgradnji raspodijeljenih sustava. Razlike u postupcima izgradnje aplikacije korištenjem usluga i izgradnje pojedinih usluga su uzrokovale pojavljivanje jezika koji su prilagođeni za definiranje kompozicija usluga i interakcija među uslugama. Primjer jezika za izgradnju aplikacija korištenjem usluga su WS-BPEL, WS-CDL i OWL-S. Najviše korišten jezik od navedenih jezika je WS-BPEL. On se koristi u brojnim sustavima za kompoziciju usluga koje su razvili Microsoft, IBM, Oracle i drugi. Ipak, svi ostvareni sustavi koriste centralizirane poslužitelje koji upravljaju izvođenjem raspodijeljene aplikacije. Korisnik-poslužitelj arhitektura u tim sustavima smanjuje njihovu pouzdanost i smanjuje mogućnost razmjernog rasta.

4.1 Raspodijeljeni sustav zasnovan na suradnji i natjecanju

Za izgradnju sustava s decentraliziranim upravljanjem, u sklopu projekta CRO-GRID Posrednički sustavi, razvijen je raspodijeljen sustav nazvan raspodijeljeni sustav zasnovan na suradnji i natjecanju (engl. *Coopetition-based distributed system*, CBDS). CBDS je otvoreni i proširiv sustav zasnovan na uslugama koji proširuje raspodijeljeni model zasnovan na koordinaciji. Raspodijeljeni sustav zasnovan na suradnji i natjecanju izvodi raspodijeljene programe koji pozivaju, koordiniraju i sinkroniziraju usluge te omogućuju njihovu suradnju i natjecanje. Podržan je i model zasnovan na obradi događaja i model zasnovan na obradi dokumenata.

Osnova CBDS-a je odvajanje aplikacijski specifične funkcionalnosti, postupaka koordinacije i mehanizama natjecanja i suradnje. Sve osnovne aplikacijski specifične funkcionalnosti su ostvarene kao odvojene i nezavisne usluge. Logika raspodijeljene aplikacije povezuje i sinkronizira izvođenje pojedinih raspodijeljenih programa koji su napisani u jeziku za opisivanje procesa izvedenom iz jezika WS-BPEL. Raspodijeljeni programi upravljaju aktivnostima u raspodijeljenom sustavu pozivajući usluge specifične za

pojedinu aplikaciju i koristeći osnovne infrastrukturne usluge koje omogućuju natjecanje, suradnju, komunikaciju i sinkronizaciju usluga.



Slika 9: Primjer raspodijeljenog sustava

Na slici 9 prikazan je primjer raspodijeljene aplikacije koja se izvodi u CBDS sustavu. Složena raspodijeljena aplikacija je podijeljena na tri primjenske usluge i šest raspodijeljenih programa. Primjenska usluga X instalirana je na računalu 1, usluga Y na računalu 2 i usluga Z na računalu 3. Na istim se računalima izvode i raspodijeljeni programi RP₁ do RP₆. Raspodijeljeni programi pozivaju primjenske usluge što je na slici prikazano punom linijom. Za sinkronizaciju, koordinaciju i natjecanje raspodijeljeni programi koriste infrastrukturne usluge suradnje i natjecanja A i B instaliranih na računala 1 i 3. Na računalu 1 su kreirane

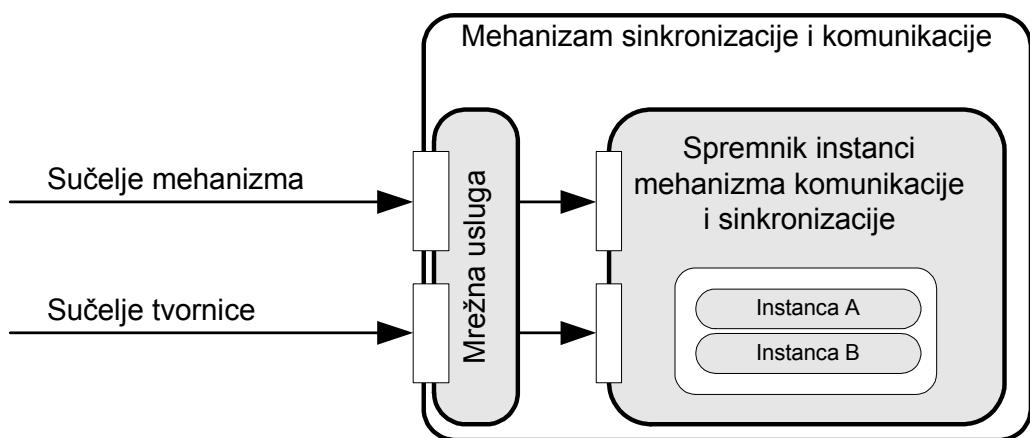
dvije instance mehanizma A. Raspodijeljeni programi pozivaju instance mehanizama za sinkronizaciju, koordinaciju i natjecanje. Primjenske usluge i mehanizmi suradnje i natjecanja su platformski neovisne usluge koje za svoje ostvarenje koriste standarde iz skupa WS-*.

5 Ostvareni komunikacijski i sinkronizacijski mehanizmi

U sklopu ovog diplomskog rada ostvarena su tri sinkronizacijska i komunikacijska mehanizma. Ostvareni mehanizmi su binarni i opći semafor te spremnik poruka. Oni su ostvareni kao samostalne usluge koristeći standarde HTTP, SOAP, WSDL, WS-Resource Framework i WS-Addressing u programskom jeziku C# koji je dio Microsoftove .Net platforme.

5.1 Osnovna arhitektura ostvarenih mehanizama

Ostvareni mehanizmi dijele zajedničku osnovnu arhitekturu koja je prikazana na slici 10. Svaka od instanci ostvarenih mehanizama za sinkronizaciju i komunikaciju je u nekom stanju i ono se mijenja samo pozivom odgovarajuće metode mrežne usluge s parametrom koji određuje instancu kojoj se pristupa. Između dva poziva stanje ostaje nepromijenjeno. Zbog toga se instance mehanizma za komunikaciju i sinkronizaciju čuvaju u spremniku za odgovarajući mehanizam. Mrežna usluga samo izlaže sučelje za pojedini mehanizam i nikad ne pristupa izravno instancama već sve pozive preusmjerava prema spremniku.



Slika 10: Arhitektura mehanizama sinkronizacije i komunikacije

Spremnik je posrednik između mrežne usluge i instanci mehanizama komunikacije i sinkronizacije. On služi i kao tvornica za stvaranje i uništavanje instanci. Zbog što veće dostupnosti usluga svaka se instance izvodi u vlastitoj upravljačkoj dretvi.

5.2 Binarni i opći semafor

Osnovni sinkronizacijski mehanizam u većini operacijskih sustava je binarni semafor. Korištenjem binarnog semafora i spremnika dijeljene memorije moguće je ostvariti većinu mehanizama sinkronizacije i komunikacije. Osnovna funkcija binarnog semafora je međusobno isključivanje procesa koji pristupaju zajedničkim računalnim sredstvima odnosno zaštita kritičnih odsječaka koda. Sve raspodijeljene aplikacije koje dijele zajednička sredstva osim trivijalno raspodijeljenih aplikacija moraju koristiti neki mehanizam za ostvarivanje kritičnih odsječaka. Zbog toga je pri ostvarenju CBDS-a binarni semafor odabran kao osnovni mehanizam sinkronizacije.

Često u raspodijeljenim aplikacijama postoji ograničen broj računalnih sredstava određene vrste. Tada je potrebno osigurati dijeljenje tih sredstava i osigurati da samo jedan proces pristupa sredstvu u jednom trenutku. Problem je moguće riješiti korištenjem jednog binarnog semafora za svako sredstvo koje se štiti ali to nije praktično rješenje. Zbog toga se za zaštitu pristupa računalnim sredstvima često koristi i opći semafor. On omogućuje zaštitu za korištenje ograničenog broja dijeljenih računalnih sredstava.

5.2.1 Stvaranje i uništavanje

Za stvaranje i uništavanje semafora koristi se uzorak tvornice (engl. *factory pattern*). Tablica 6 prikazuje sučelje tvornice binarnog semafora. Korisnik stvara novi binarni semafor pozivom metode *Stvori*. Ulazni parametri su identifikator instance binarnog semafora koji se stvara te krajnja točka korisnika koja jedinstveno određuje korisnika koji je stvorio binarni semafor. Povratne vrijednosti su uspješnost operacije stvaranja i identifikator stvorenog binarnog semafora koji ga jedinstveno određuje. Stvoreni binarni semafor je nakon stvaranja postavljen. Zbog sigurnosnih razloga dozvoljeno je uništavanje semafora samo korisniku koji ga je stvorio. Zbog toga je kao ulazni parametar metode *Uništi* potrebno, osim identifikatora

binarnog semafora koji se uništava, navesti i krajnju točku korisnika koji uništava semafor. Povratna vrijednost određuje da li je operacija uništavanja uspješno izvedena.

| Metoda | Parametri | |
|--------|----------------------|---|
| Stvor | Ulazni parametri | Identifikator instance binarnog semafora Krajnja točka korisnika |
| | Povratne vrijednosti | Identifikator kreirane instance binarnog semafora Uspješnost operacije |
| Uništi | Ulazni parametri | Identifikator binarnog semafora koji se uništava Krajnja točka korisnika |
| | Povratna vrijednost | Uspješnost operacije |

Tablica 6: Sučelje tvornice binarnog semafora

Sučelje tvornice općeg semafora prikazano je u Tablica 7. Postupak stvaranja općeg semafora je sličan stvaranju binarnog semafora. Jedina razlika je dodatni parametar koji određuje maksimalnu vrijednost brojača. To je i njegova početna vrijednost. Uništavanje obje vrste semafora je identično.

| Metoda | Parametri | |
|--------|----------------------|--|
| Stvor | Ulazni parametri | Identifikator instance općeg semafora Maksimalna vrijednost brojača općeg semafora Krajnja točka korisnika |
| | Povratne vrijednosti | Identifikator kreiranog općeg semafora Uspješnost operacije |
| Uništi | Ulazni parametri | Krajnja točka općeg semafora koji se briše Krajnja točka korisnika |
| | Povratna vrijednost | Uspješnost operacije |

Tablica 7: Sučelje tvornice općeg semafora

5.2.2 Sučelje binarnog i općeg semafora

Za primjenu binarnog i općeg semafora koristi se jednako sučelje. Ono je opisano u tablici 8 i sastoji se od dvije metode: *Ispitaj* i *Postavi*. Metoda *Ispitaj* provjerava da li je semafor zauzet i ako nije, onda ga zauzima. Prvi parametar metode je identifikator semafora koji se ispituje. Sljedeći parametar je *vrijeme čekanja* koje određuje kako će se postupiti u slučaju da je semafor već zauzet. Dozvoljene su samo dvije vrijednosti: nula i beskonačno (predstavljeno s vrijednošću -1). Ako je zadano *vrijeme čekanja* nula, onda povratna vrijednost određuje da li je semafor uspješno zauzet ili nije. Nakon neuspješnog ispitivanja semafora korisnik mora ponovno pozvati metodu *Ispitaj* da bi zauzeo semafor. Ako korisnik

mora čekati zauzeće semafora, onda postavlja parametar metode *Ispitaj vrijeme čekanja* na beskonačno. Povratna vrijednost označava da li je semafor uspješno zauzet ili je korisnik dodan u rep za čekanje te će biti obaviješten ako zauzme semafor. Za obavještavanje korisnika se koristi metoda mrežne usluge čiji opis je dan kao treći parametar metode *Ispitaj*. Četvrti parametar određuje krajnju točku korisnika koji želi zauzeti semafor. Povratna vrijednost određuje da li je korisnik uspio zauzeti semafor. Ako ona ima vrijednost *false*, tada zauzimanje semafora nije uspjelo i korisnik može očekivati obavijest o zauzimanju ako je kao *vrijeme čekanja* naveo beskonačno vrijeme.

Operacija postavljanja semafora obavlja se pozivom metode *Postavi*. Oslobađanje je dozvoljeno samo onim korisnicima koji su prethodno zauzeli binarni semafor pozivom metode *Ispitaj*. Metoda ima samo dva parametra. Prvi je identifikator semafora koji se postavlja. Drugi parametar je krajnja točka korisnika koji postavlja semafor. Povratna vrijednost obavještava korisnika o uspješnosti operacije.

| Metoda | Parametri | |
|---------|---------------------|--|
| Ispitaj | Ulazni parametri | Identifikator semafora koji se ispituje |
| | | Vrijeme čekanja |
| | | WSDL opis sučelja za dojavljivanje |
| | | Krajnja točka korisnika |
| Postavi | Povratna vrijednost | Uspješnost operacije |
| | Ulazni parametri | Identifikator semafora koji se postavlja |
| | | Krajnja točka korisnika |
| | Povratna vrijednost | Uspješnost operacije |

Tablica 8: Sučelje binarnog i općeg semafora

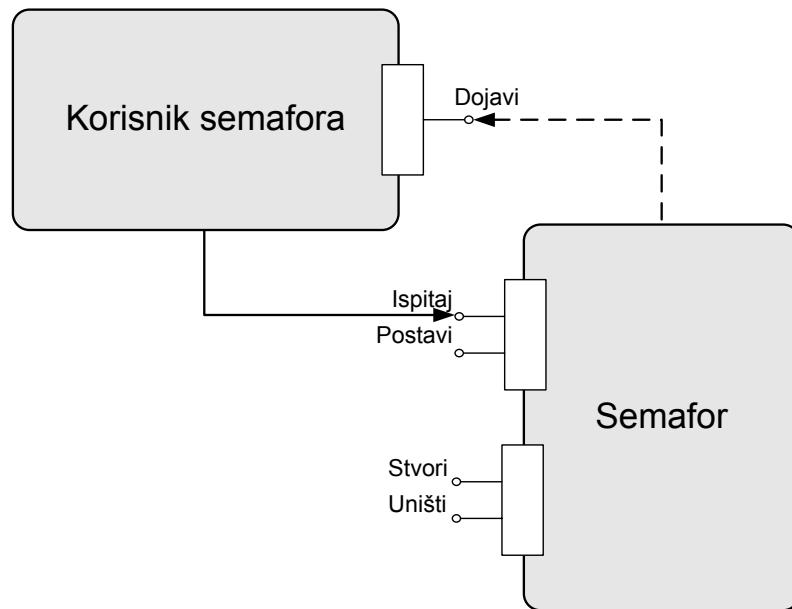
5.2.3 Način zauzimanja semafora

Semafori se mogu zauzimati na dva načina: prozivanjem (engl. *pool*) ili pozivom dojavne metode (engl. *callback method*). Kod zauzimanja semafora prozivanjem drugi parametar metode *Ispitaj, vrijeme čekanja*, ima vrijednost nula. Povratna vrijednost određuje da li je semafor uspješno zauzet. Ako proces nije uspješno zauzeo semafor, onda može ponovno pokušati nakon nekog vremena. Između dva pokušaja zauzimanja proces može odradivati neki drugi posao. Ako proces ne može nastaviti s izvođenjem dok ne zauzme semafor ili se želi izbjegći stvaranje dodatnog mrežnog prometa i trošenje računalnih sredstava kod prozivanja, onda zauzimanje prozivanjem nije prikladno. U takvim slučajevima je prikladniji način zauzimanja semafora pozivom dojavne metode. *Vrijeme čekanja* u zauzimanju dojavom postavlja se na beskonačno. Ako je semafor slobodan pri pozivu metode

Ispitaj, onda se semafor odmah zauzima kao i kod zauzimanja pozivanjem. Inače, povratna vrijednost se postavlja na *false* i semafor stavlja korisnika u rep za čekanje. Kad neki proces oslobodi semafor tada prvi korisnik u repu čekanja zauzima semafor, uklanja se iz repova te biva obaviješten pozivanjem njegove dojavne metode koja je ostvarena kao mrežna usluga.

5.2.4 Okolina binarnog i općeg semafora

Korisnici binarnog i općeg semafora su mrežne usluge. Na slici 11 prikazana je okolina semafora. Ako korisnik pozivom metode *Ispitaj* s vremenom čekanja postavljenim na beskonačno pokušava zauzeti semafor koji je već zauzet, onda će semafor zahtjev staviti u rep za čekanje. Kad se semafor oslobodi korisnik će o tome biti obaviješten. Korisnik može istovremeno čekati na nekoliko semafora pa je u dojavnoj metodi kao jedan parametar navedena i krajnja točka semafora. Drugi parametar dojavne funkcije je novo stanje semafora. Dojavna metoda nije u potpunosti definirana. Ona ima definirane parametre, ali naziv je proizvoljan. Zbog toga korisnik u zahtjevu navodi WSDL usluge za dojavu. Na taj način je moguće provjeriti da li su parametri usluge odgovarajući. Korisnik tako može koristiti različite mrežne usluge za primanje obavijesti o različitim semaforima.

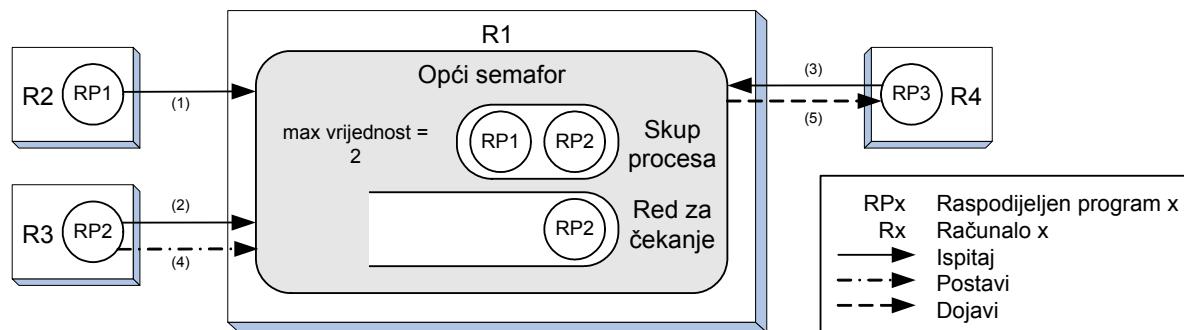


Slika 11: Okolina semafora

Sučelje za dojavu se koristi i u slučajevima kad se semafor uništava, a neki korisnici još čekaju na njegovo zauzimanje. Tada se svi korisnici obavijeste da se semafor uništava. To se postiže postavljanjem parametra novog stanja semafora na vrijednost DESTROYED. Na

ovaj način se smanjuje mogućnost nastanka potpunog zastoja prijevremenim uništavanjem semafora i olakšava se otkrivanje pogrešaka.

Slika 12 prikazuje primjer raspodijeljene aplikacije koja koristi opći semafor. Tri raspodijeljena programa koja se izvode na računalima R_2 , R_3 i R_4 koriste opći semafor koji je instaliran na računalu R_1 . Instanca općeg semafora ima početnu vrijednost 2. Zbog toga prva dva raspodijeljena programa uspijevaju zauzeti semafor (1), (2). Njihove krajnje točke se pohranjuju u skup procesa koji su zauzeli semafor. Kad treći raspodijeljeni program pokuša zauzeti semafor njegova se krajnja točka i WSDL opis dojavne metode pohranjuju u rep za čekanje (3). Kad drugi raspodijeljeni program pozove metodu *Postavi*, njegova se krajnja točka briše iz skupa procesa koji zauzimaju semafor i zamjenjuje se s krajnjom točkom trećeg raspodijeljenog programa. Nakon toga se treći raspodijeljeni program obavijesti pozivom dojavne metode.



Slika 12: Primjer raspodijeljene aplikacije koja koristi opći semafor

5.2.5 Svojstva binarnog i općeg semafora

Binarni i opći semafor prilagođeni su posebnostima raspodijeljenog sustava zasnovanog na suradnji i natjecanju. Zbog toga se razlikuju od ostvarenja koja su uobičajena u operacijskim sustavima. Osnovna razlika je platformska neovisnost pa se pomoću njih može izvršiti sinkronizacija aplikacija napisanih u različitim programskim jezicima koje se izvode na različitim operacijskim sustavima. U heterogenom okruženju u kojem se mogu izvoditi razne aplikacije koje međusobno surađuju, bile su potrebne posebne mjere sigurnosti pa je uvedeno ograničenje da semafor može uništiti samo proces koji ga je stvorio. Na taj način je uklonjena mogućnost da neki proces namjerno ili slučajno uništi semafor prerano, što bi moglo uzrokovati potpuni zastoj raspodijeljene aplikacije.

Binarni semafor je izgrađen zbog zaštite pristupa računalnim sredstvima pa je uvedeno ograničenje da semafor može postaviti samo proces koji ga je i zauzeo. Isto vrijedi i za opći semafor koji se zbog toga ne može iskoristiti za određivanje redoslijeda pristupanja dijeljenoj memoriji kod komunikacije porukama na način na koji je to opisano u poglavljju 2.1.3. Ipak, ovo ne predstavlja ograničenje jer raspodijeljeni sustav zasnovan na suradnji i natjecanju posjeduje posebne mehanizme za komuniciranje porukama. Još jedna bitna karakteristika je da proces ne može odmah i zauzeti semafor u postupku stvaranja. Zbog toga proces koji je stvorio semafor mora pozvati metodu *Ispitaj* kao i svi ostali procesi.

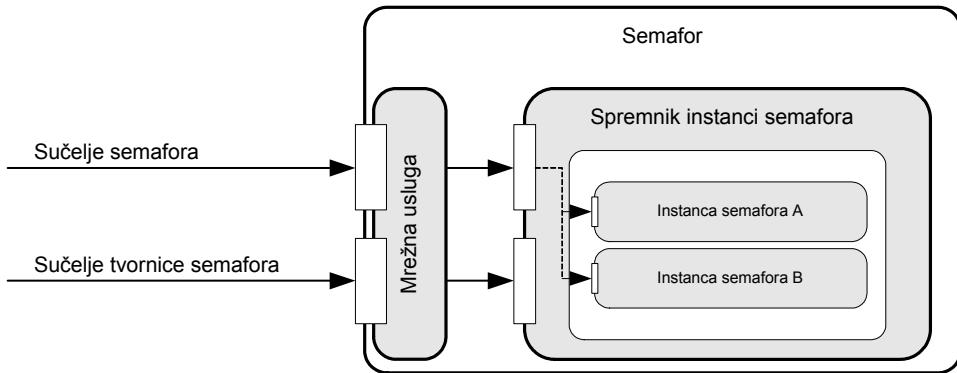
Često u ostvarenjima binarnog i općeg semafora kod njihova uništavanja nije definirano ponašanje procesa koji čekaju na semaforu. Uništavanje semafora koji je još u upotrebi tretira se kao pogreška i programeru je ostavljeno da ih otkriva samostalno. Pronalaženje pogrešaka u raspodijeljenim aplikacijama zasnovanim na uslugama je složeno pa je iskorišten već postojeći mehanizam dojavljivanja za obavještavanje svih korisnika koji čekaju na zauzeće semafora koji se uništava. Dojavljivanjem je olakšan postupak pronalaženje pogrešaka koje nastaju zbog pogrešnog uništavanja semafora i otklonjena opasnost od nastajanja potpunog zastoja.

5.3 Ostvarenje binarnog i općeg semafora

Binarni i opći semafor programski su ostvareni koristeći .Net platformu i ASP .NET Web Service tehnologiju. U razvoju su korišteni programski jezik C# i WS-Resources Framework standard. Zbog toga je svaka instanca semafora WS-Resource. Za mrežni poslužitelj kojim se pristupa uslugama, zbog prilagodenosti s ostalim tehnologijama tvrtke Microsoft, odabran je Microsoft IIS, verzija 5.1. On je omogućio jednostavno objavljivanje mrežnih usluga izgrađenih koristeći C# i .Net razvojnu platformu. Mrežna usluga ostvarene ovim tehnologijama sastoji se od nekoliko datoteka od kojih su najvažnije datoteke s nastavkom .cs, u kojoj se nalazi C# kôd usluge i .asmx, koja služi za pokretanje usluge. Da bi se moglo pristupiti mrežnoj usluzi potrebno je u operacijskom sustavu Windows postaviti prava pristupa svim datotekama. Usluge su smještene u virtualni direktorij CoopetitionServices/ImeUsluge/. Tako se usluga za binarni semafor smještena na računalo www.ris.fer.hr nalazi na adresi:

<http://www.ris.fer.hr/CoopetitionServices/BinarySemaphore/BinarySemaphore.asmx>

Na slici 13 prikazana je programska arhitektura semafora. U njoj su istaknuta tri C# razreda koji ostvaruju njegovu funkcionalnost.



Slika 13: Programska arhitektura općeg i binarnog semafora

Tablica 9 prikazuje nazive razreda koji ostvaruju module binarnog i općeg semafora koji su prikazani na slici 13.

| | Binarni semafor | Opći semafor |
|---------------|---------------------------|-----------------------------|
| Mrežna usluga | BinarySemaphoreWebService | CountingSemaphoreWebService |
| Spremnik | BinarySemaphoreContainer | CountingSemaphoreContainer |
| Instanca | BinarySemaphoreInstance | CountingSemaphoreInstance |

Tablica 9: Nazivi klasa binarnog i općeg semafora

5.3.1 Mrežne usluge i serijalizacija razreda u XML

Serijalizacija je postupak kojim se objekt pretvara u oblik koji se lako može prenositi, npr. preko mreže, ili zapisati u datoteku. Dvije osnovne vrste serijalizacije su binarna serijalizacija i serijalizacija u XML. Parametri mrežnih usluga prenose se u tijelu SOAP poruke ili u njenom zaglavljtu, a SOAP poruke se temelje na XML-u. Zbog toga se svi razredi koji se prenose kao parametri ili vraćaju kao rezultati kod poziva metoda mrežnih usluga moraju serijalizirati u XML.

.Net platforma omogućuje da se razredi, metode i varijable opišu dodatnim podacima koji detaljnije opisuju njihova svojstva. Dodatni podaci se dodaju upotrebom *svojstava* (engl. *attributes*). Svojstva su razredi izvedeni iz razreda *Attribute*. Njihovom uporabom moguće je dodati proizvoljne informacije pojedinim entitetima izvornog koda. Za neke namjene, kao što je serijalizacija u XML, koriste se predefinirana svojstva. Zbog toga je postupak serijalizacije u C# jeziku za programera u potpunosti transparentan. Svi se osnovni tipovi mogu serijalizirati u XML. Razredi koje korisnik napravi ne mogu se serijalizirati osim ako se to ne

dopusti korištenjem svojstva razreda *SerializableAttribute*. Programeru je omogućeno i da određuje kako će se pojedine varijable ili objekt u cijelosti serijalizirati. Moguće je mijenjati nazive pojedinih XML oznaka te određivati da li će se za neku varijablu dodati nova XML oznaka ili će se ona opisati pomoću svojstva. Korištenjem svojstava za opisivanja načina serijalizacije moguće je jednostavno izgraditi C# razrede koji se serijaliziraju u XML prema postojećim opisima u XML Schema dokumentima.

U SOAP zaglavlju se prema standardu *WS-Addressing* prenose identifikatori semafora ostvareni pomoću klase *ResourceID*. Drugi razred koji se predaje kao parametar je opisnik krajnje točke korisnika ostvaren razredom *EndpointReference*. On je opisan sa XML schema dokumentom *EndpointReferenceType.xsd* koji se nalazi na adresi <http://schemas.xmlsoap.org/ws/2004/03/addressing>. Da bi olakšao izgradnju C# razreda koji su opisani određenim XML schema dokumentom u .NET razvojnu okolinu uključen je alat XSD. Za stvaranje klase koja je opisana XML schema dokumentom *EndpointReferenceType.xsd* potrebno pokrenuti XSD program sa sljedećim parametrima:

```
XSD /c EndpointReferenceType.xsd
```

5.3.2 Sučelje mrežne usluge binarnog semafora

Razred *BinarySemaphoreWebService* izveden je iz razreda *WebService* koji je dio .Net programske potpore za razvoj mrežnih usluga. Na slici 14 je prikazan dio C# koda za deklariranje *BinarySemaphoreWebService* razreda. Korištenjem posebnog svojstva tipa *WebServiceAttribute* se označava razred kojim je ostvarena mrežna usluga. Tim svojstvom se definiraju osnovni podaci o mrežnoj usluzi kao što su ime te mrežne usluge, prostor imena u kojem se to ime nalazi te opis mrežne usluge.

```
[WebService( Name="BinarySemaphore",
Namespace="http://www.ris.fer.hr/CoopetitionServices/BinarySemaphore")]
public class BinarySemaphoreWebService : System.Web.Services.WebService
```

Slika 14: Deklaracija mrežne usluge u programskom jeziku C#

Metode mrežne usluge su obične C# metode sa svojstvom tipa *WebMethodAttribute*.

Primjer deklaracije metode *Obtain* mrežne usluge binarnog semafora je prikazan na slici 15.

Osim svojstva tipa *WebMethodAttribute* koristi se i svojstvo tipa *SoapHeaderAttributer* kojim se određuju parametri koji se prenose preko zaglavlja te da li su to ulazni parametri ili povratne vrijednosti.

```
[WebMethod]  
[SoapHeader("rID", Direction=SoapHeaderDirection.In)]  
public bool Obtain(int timePar, XmlElement CB, EndpointReferenceType rEPR)
```

Slika 15: Deklaracija metode mrežne usluge

Primjer SOAP poruke kojom se poziva metoda *Obtain* mrežne usluge binarnog semafora prikazan je na slici 16. U zaglavljtu SOAP poruke se, prema standardima WS-Resource Framework i WS-Addressing, adresira instanca semafora s identifikatorom A. Parametar CB je tipa *XmlElement* i pomoću njega se može definirati proizvoljan *XmlElement* u poruci. U metodi *Obtain* on se koristi za prijenos informacija potrebnih za ostvarenje mehanizma dojavljivanja.

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">  
    <soap:Header>  
        <ResourceID  
            xmlns="http://www.ris.fer.hr/CoopetitionServices/BinarySemaphore">  
                A  
            </ResourceID>  
    </soap:Header>  
    <soap:Body>  
        <Obtain  
            xmlns="http://www.ris.fer.hr/CoopetitionServices/BinarySemaphore">  
            <timePar>-1</timePar>  
            <CB>xml</CB>  
            <EndpointReference  
                xmlns="http://schemas.xmlsoap.org/ws/2004/03/addressing">  
                ...  
            </EndpointReference>  
        </Obtain>
```

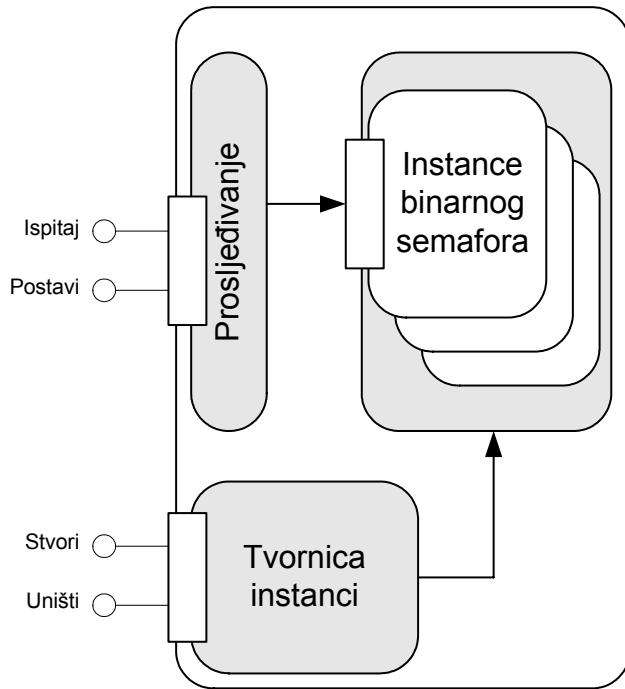
```
</soap:Body>  
</soap:Envelope>
```

Slika 16: SOAP omotnica za poziv metode Obtain

5.3.3 Spremnik instanci binarnih semafora

Razred *BinarySemaphoreContainer* ostvaruje funkcionalnost spremnika instanci binarnog semafora. On omogućuje stvaranje i uništavanje instanci binarnog semafora, pohranjuje ih i omogućuje mrežnoj usluzi pristup do pojedine instance. Instance u spremniku imaju jedinstvenu identifikacijsku oznaku sačinjenu od niza znakova. Zbog toga je za njihovo pohranjivanje potrebna struktura podataka koja se naziva rječnik. U .Net razvojnoj platformi struktura podataka rječnik ostvarena je razredom *Hashtable* pa su instance pohranjene u nju. Za svaki pristigli zahtjev posebna dretva pristupa spremniku pa su svi pristupi rječniku kritični odsječci. Za njihovo ostvarivanje koristi se ključna riječ *lock*.

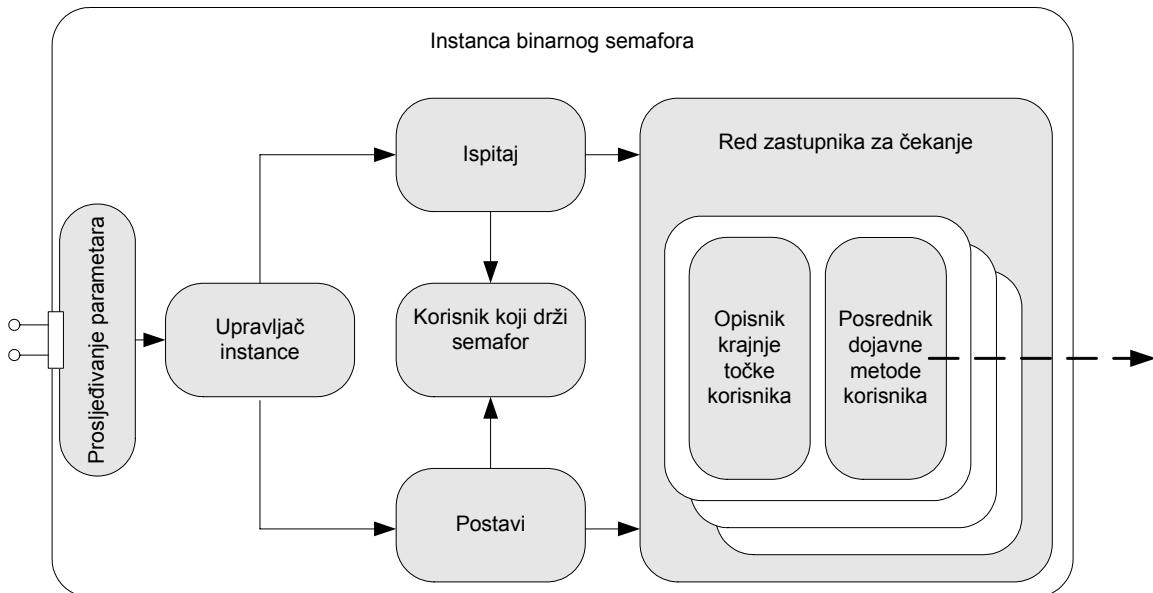
Spremnik omogućuje ostvarivanje sredstava s očuvanjem stanja. Zbog toga on mora postojati sve dok se zadnji binarni semafor ne uništi. Više različitih raspodijeljenih aplikacija može koristiti spremnik pa se on mora izvoditi u posebnom procesu neovisnom o aplikacijama koje ga koriste. Taj proces može biti poseban poslužiteljski proces koji komunicira s mrežnom uslugom koristeći se .Net Remoting mehanizmom. Drugi način je pohrana spremnika u *Application* objektu ASP .NET procesa. Ovaj način je jednostavan za ostvarenje jer tada mrežna usluga može izravno pristupiti spremniku bez korištenja nekog komunikacijskog mehanizma. Prvi način je otporniji na pogreške jer ne ovisi o ASP .NET podsustavu čiji prestanak rada uzrokuje gubitak sadržaja spremnika. On je i prilagodljiviji jer omogućuje proizvoljno ostvarenje spremnika, ali je složeniji zbog dodatne komunikacije. Isprobana su oba načina i na kraju je zbog svoje jednostavnosti odabran drugi način koji je prikazan na slici 17.



Slika 17: Arhitektura spremnika instanci binarnog semafora

5.3.4 Instanca binarnog semafora

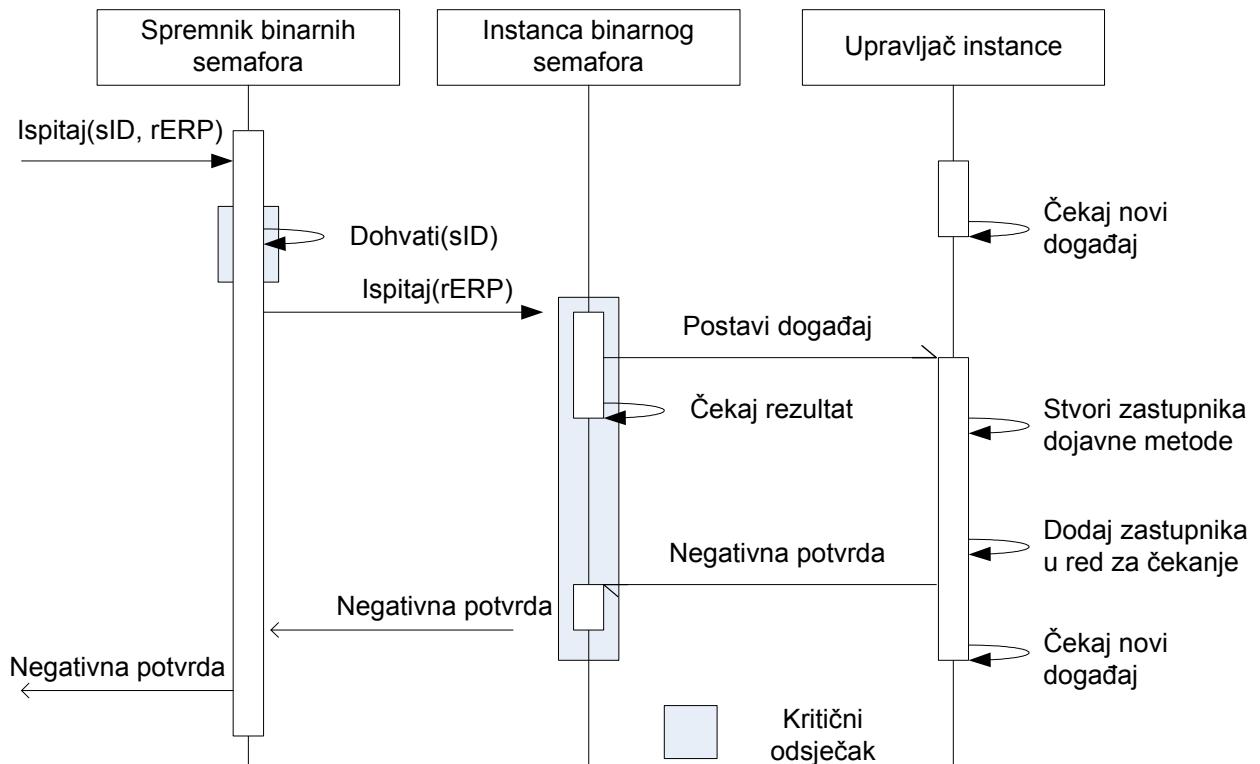
Instanca binarnog semafora je ostvarena razredom *BinarySemaphoreInstance*. Binarni semafor mora znati koja je krajnja točka korisnika koji je stvorio semafor, krajnju točku korisnika koji je zauzeo semafor te sve pristigle zahtjeve. Zahtjevi se čuvaju u repu u koji se za svaki zahtjev stavlja instanca razreda *FacilityProxyWrapper* koji pamti krajnju točku odredišta i WSDL koji opisuje uslugu za dojavljivanje. Slika 18 prikazuje programsku arhitekturu instance binarnog semafora.



Slika 18: Arhitektura instance binarnog semafora

Za svaki pristup mrežnoj usluzi, ASP .Net stvara novu dretvu kojom se pristupa binarnom semaforu. Operacije semafora su nedjeljive pa je pristup semaforu kritični odsječak. Ipak, za korisnika semafora bitno je jedino zauzimanje ili oslobođanje semafora, dok metoda *Release* još treba obavijestiti sljedećeg korisnika u repu. S gledišta korisnika taj dio je nebitan za uspješan ishod operacije pa je poželjno da vrijeme izvođenja dretve zahtjeva bude što kraće. Zbog toga svaka instance binarnog semafora posjeduje upravljačku dretvu koja obavlja sve operacije semafora. ASP .Net dretva predaje parametre upravljačkoj dretvi određene instance binarnog semafora. Ona tada počinje obavljati operaciju i vraća rezultat ASP .Net dretvi čim obavi operaciju koju je korisnik tražio, a nakon toga započne s postupkom dojavljivanja. Kad ASP .Net dretva izdiže iz kritičnog odsječka druga dretva može pristupiti semaforu. Za dva uzastopna pristupa istom semaforu se ipak mora pričekati da kontrolna dretva završi posao do kraja.

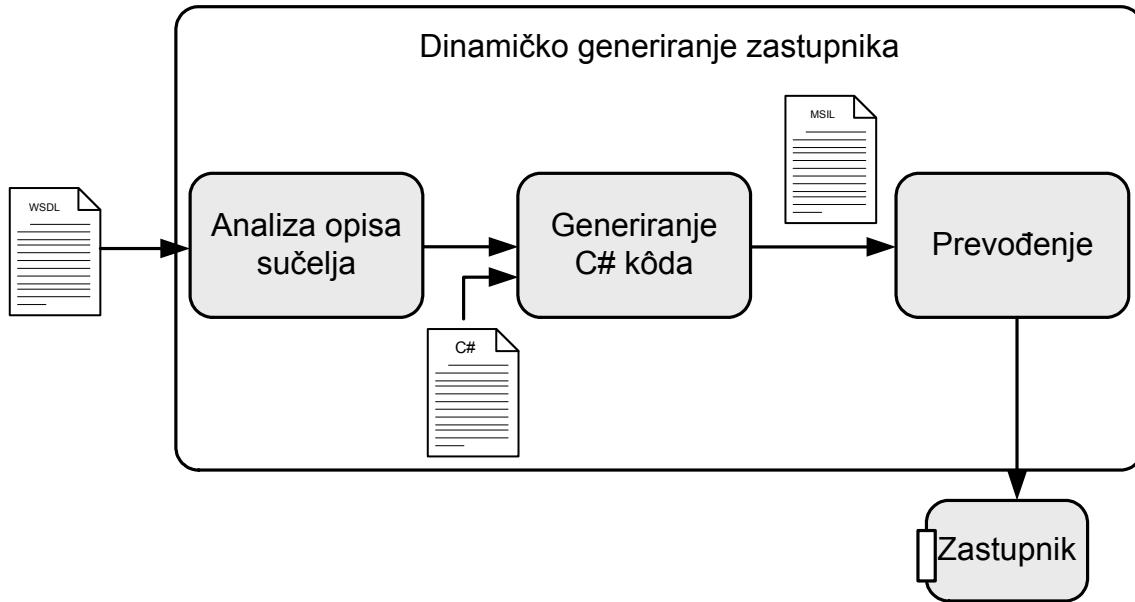
Slika 19 prikazuje vremenski dijagram postupka zauzimanja binarnog semafora koji je već zauzet. Slika prikazuje komunikaciju između ASP .Net dretve zahtjevatelja i upravljačke dretve instance.



Slika 19: Primjer ispitivanja semafora

5.3.5 Ostvarenje mehanizma dojave

Za ostvarenje mehanizma dojavljivanja koristi se razred *FacilityProxyWrapper*. On omogućuje pristupanje mrežnoj usluzi za dojavljivanje opisanoj WSDL-om koji je zahtjevatelj poslao kao parametar poziva funkcije *Obtain*. Postupak stvaranja zastupnika korisnika je prikazan na slici 20. Iz opisa dojavne usluge generira se kôd zastupnika mrežne usluge za dojavu u jeziku C#. Generirani kôd je razred sa tri metode. Prva služi za sinkrono pozivanje metode mrežne usluge, a druge dvije za asinkrono pozivanje. Iako dojavna metoda može imati proizvoljno ime, njeni parametri se ne mogu mijenjati pa se C# kôd stvara samo umetanjem odgovarajućeg imena mrežne usluge, imena dojavne metode i prostora imena u predložak koda koji je staticki generiran alatom WSDL. Tako generirani kôd nakon toga se prevodi u jezik MSIL (Microsoft intermediate language) pomoću .Net razreda *CSharpCodeProvider*. Rezultat te operacije je objekt razreda *Assembly* koji sadrži prevedeni kôd. Za pozivanje dojavne metode koristi se .Net reflection mehanizam.



Slika 20: Postupak dinamičkog stvaranja zastupnika

Kao parametri dojavne metode koriste se objekti razreda *EndPointreference* i pobrojani tip *sStatus*. Njihov kôd nije potrebno prevoditi kod svakog generiranja zastupnika klijenta za dojavljivanje već se mogu koristiti njihove prevedene verzije iz datoteka *EndPointReferenceLib.dll* i *SemaphoreUtils.dll*. Zbog sigurnosnih mehanizama ugrađenih u .Net platformu mrežne usluge ne mogu pristupati datotekama na računalu na kojem se izvode. Da bi se omogućilo pristup navedenim datotekama potrebno je u Windows postavkama za te dvije datoteke izravno omogućiti pristup ASP .NET procesu, a to se postiže dodjeljivanjem prava za čitanje tih datoteka ASPNET korisniku.

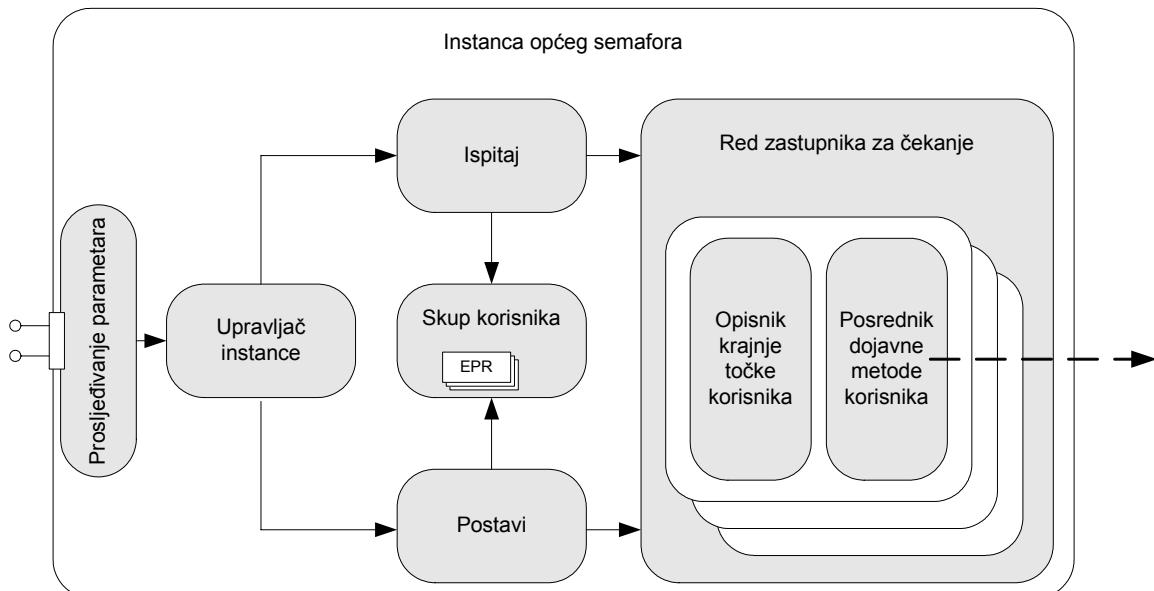
Mehanizam dojave koristi se za obavještavanja korisnika iz repa čekanja o zauzimanju semafora. Za dojavu se koristi sinkroni poziv metode. Drugi slučaj u kojem se koristi mehanizam dojavljivanja je kad vlasnik semafora odluči uništiti semafor čiji rep čekanja nije prazan. O tome se moraju obavijestiti svi korisnici u repu za čekanje. Sinkroni pozivi su u ovom slučaju neprimjereni jer je potrebno što prije obavijestiti sve korisnike. Zbog toga se za dojavljivanje koristi asinkroni poziv metode za dojavljivanje.

5.3.6 Razredi općeg semafora

Arhitektura izvedbe općeg semafora je potpuna jednaka arhitekturi binarnog semafora. Sastoje se od tri razreda: *CountingSemaphoreWebService*, *CountingSemaphoreContainer* i *CountingSemaphoreInstance*. Osnovna razlika u ostvarenju navedenih mehanizama

sinkronizacije je u njihovoј funkcionalnosti па су razlike najuočljivije u izvedbi upravljačke dretve klase *CountingSemaphoreInstance*, odnosno u dodatnom kodu kojim opći semafor pamti krajnje točke svih korisnika koji zauzmu semafor. Komunikacija mrežne usluge i spremnika, osnovna logika spremnika, komunikacija ASP .NET dretve s upravljačkom dretvom instance, pristup kritičnim odsjećcima i svi ostali aspekti izvedbe su isti. Slika 21 prikazuje arhitekturu izvedbe instance općeg semafora.

Kad neka aplikacija pozove metodu *Obtain*, ako je brojač pozitivan opći semafor pamti krajnju točku korisnika u skupu korisnika i smanjuje brojač. Kad brojač postane jednak nuli za sve pozive metode *Obtain* s vremenskim parametrom postavljenim na beskonačno opći semafor stvara zastupnika za dojavljivanje jednakim postupkom kao i binarni semafor i stavlja ga na kraj repa za čekanje.



Slika 21: Arhitektura instance općeg semafora

5.4 Spremnik poruka

Komunikacija porukama je poželjan način komunikacije za razvoj labavo povezanih sustava. Ipak, labavo povezane aplikacije najčešće trebaju mogućnosti asinkrone komunikacije pa usluge prijenosnog sloja mreže nisu dovoljne. Zbog toga je za potrebe razvoja raspodijeljenog sustava zasnovanog na suradnji i natjecanju ostvaren mehanizam komunikacije porukama. Raspodijeljene aplikacije mogu pomoću spremnika poruka

razmjenjivati XML poruke proizvoljnog formata. Spremnik je ostvaren kao mrežna usluga pa ga mogu koristiti sve aplikacije koje podržavaju WS-* skup standarda, neovisno o platformi na kojoj se izvode.

5.4.1 Stvaranje i uništavanje

Za stvaranje i uništavanje spremnika poruka koristi se uzorak tvornice. Sučelje tvornice je prikazano u tablici 10. Korisnik stvara novi spremnik poruka pozivom metode *Stvari*. Ulazni parametri su identifikator instance spremnika poruka koji se stvara te krajnja točka korisnika koja jedinstveno određuje korisnika koji je stvorio spremnik poruka. Povratna vrijednost je uspješnost operacije stvaranja i identifikator koji jedinstveno određuje stvoreni spremnik. Zbog sigurnosnih razloga dozvoljeno je uništavanje spremnika poruka samo korisniku koji ga je stvorio. Zbog toga je kao ulazni parametar metode *Uništi* potrebno, osim identifikatora spremnika poruka koji se uništava, navesti i krajnju točku korisnika koji je pozvao metodu. Povratna vrijednost određuje da li je operacija uništavanja uspješno izvedena.

| Metoda | Parametri | |
|--------|----------------------|--|
| Stvari | Ulazni parametri | Identifikator instance spremnika poruka Krajnja točka korisnika |
| | Povratne vrijednosti | Identifikator kreirane instance spremnika poruka Uspješnost operacije |
| Uništi | Ulazni parametri | Identifikator spremnika poruka koji se uništava Krajnja točka korisnika |
| | Povratna vrijednost | Uspješnost operacije |

Tablica 10: Sučelje tvornice spremnika poruka

5.4.2 Sučelje spremnika poruka

Tablica 11 prikazuje sučelje spremnika poruka. Spremnik poruka je nakon stvaranja prazan. Da bi se u njega pohranila neka poruka koristi se metoda *StaviPoruku* koja prima 3 parametra. Prvi parametar je identifikator spremnika poruka koji određuje u koji će se spremnik poruka pohraniti. Sljedeći parametar je sama poruka. Zadnji parametar je krajnja točka korisnika koji pohranjuje poruku u spremnik. Povratna vrijednost uspješno obavljene operacije stavljanja poruke je *true*.

Dohvaćanje poruke iz spremnika poruka obavlja se pozivom metode *DohvatiPoruku*. Metoda ima četiri parametra. Prvi parametar je identifikator spremnika poruka iz kojeg se

dohvaća poruka. Sjedeći parametar je *vrijeme čekanja* koje određuje da li korisnik očekuje rezultat odmah nakon poziva ili je spremnik u vrijeme poziva bio prazan. Ovaj parametar ima slično značenje kao i *vrijeme čekanja* kod binarnog i općeg semafora. Ako je poruka odmah dostupna, onda se ona odmah vrati korisniku bez obzira na vrijednost parametra *vrijeme čekanja*, a ako poruke nema, tada se vraća posebna poruka koja označava da je spremnik prazan. Kad korisnik postavi vrijeme čekanja na beskonačno i spremnik je prazan, zahtjev se pohranjuje u rep čekanja, dok se u spremnik ne pohrani neka poruka. Tada se mehanizmom dojavljivanja, koji je ostvaren na jednaki način kao i kod binarnog i općeg semafora, poruka dostavi prvom korisniku u repu čekanja.

| Metoda | Parametri | |
|---------------|---------------------|---|
| StaviPoruku | Ulazni parametri | Identifikator instance spremnika poruka |
| | | XML poruka |
| DohvatiPoruku | Ulazni parametri | Krajnja točka korisnika |
| | | Uspješnost operacije |
| DohvatiPoruku | Povratna vrijednost | Identifikator instance spremnika poruka |
| | | Vrijeme čekanja |
| DohvatiPoruku | Ulazni parametri | WSDL opis sučelja za dojavljivanje |
| | | Krajnja točka korisnika |
| DohvatiPoruku | Povratna vrijednost | XML poruka |

Tablica 11: Sučelje spremnika poruka

5.4.3 Načini dohvaćanja poruka

Poruke se mogu dohvaćati na dva načina: prozivanjem ili pozivom dojavne metode. Parametar *vrijeme čekanja* određuje koju metodu korisnik želi primijeniti. Ako se *vrijeme čekanja* postavi na nula, tada se koristi prva metoda koja je pogodna za slučajeve kad se odmah treba izvršiti neka akcija bez obzira da li ima pristiglih poruka. Ako neki proces treba poruku za nastavak posla, tada je pogodnija druga metoda u kojoj je *vrijeme čekanja* postavljeno na beskonačno. Za podjelu zadataka radnim procesima koristi drugi način pozivanja. Kad neki proces završi sa poslom on mora čekati sljedeći zadatak koji će pristići na spremnik poruka.

5.4.4 Vrste komunikacije porukama

Komunikacija pomoću spremnika poruka je postojana i asinkrona. Spremnik poruka pohranjuje sve uspješno pristigle poruke. Kad raspodijeljeni program jednom pošalje poruku on se ne mora izvoditi da bi poruka bila isporučena. Korisnik koji želi pročitati poruku iz spremnika ne mora se izvoditi u trenutku kad je poruka poslana. Zbog toga su dva procesa koja komuniciraju pomoću spremnika poruka vremenski nepovezana. Korisnik koji šalje poruku mora znati adresu i identifikator spremnika, ali on nije referencijski povezan s programom koji će tu poruku pročitati jer ne mora znati njegovu adresu.

Pošiljaocu poruke ne jamči se da će poruka biti isporučena jer korisnik nikad ne mora pozvati metodu *DohvatiPoruku* nakon što je poruka stavljena u spremnik. Poruka će ostati u spremniku dok je korisnik ne pročita ili dok vlasnik ne odluči uništiti spremnik poruka i tako uzrokuje gubitak svih poruka.

Svaki proces može imati svoj privatni spremnik poruka koji će omogućiti ostalim raspodijeljenim procesima da mu šalju poruke. Moguće kombinacije su prikazane u tablici 12. Kod komunikacije s jednim primateljem i jednim pošiljaocem radi se o privatnoj komunikaciji. Druga mogućnost je da jedan pošiljalac šalje poruke koje prima više raspodijeljenih programa. Ovakva komunikacija je tipična za dodjelu zadataka radnim procesima. Treći oblik komunikacije u kojoj više pošiljaoca šalje informacije jednom raspodijeljenom programu se najčešće koristi da se prikupe rezultati obrade nekih podataka ili općenito podaci s različitih izvora. Zadnji oblik komunikacije omogućuje centralno mjesto za prikupljanje informacija. U njemu se poruke skupljaju s više izvora i dijele na obradu različitim radnim procesima.

| | Jedan pošiljalac | Više pošiljaoca |
|-----------------|-----------------------------|----------------------------|
| Jedan primatelj | Jedan na jedan komunikacija | Više na jedan komunikacija |
| Više primatelja | Jedan na više komunikacija | Više na više komunikacija |

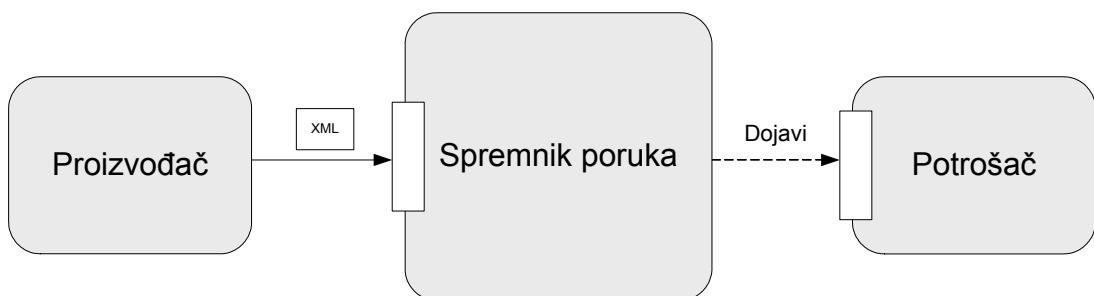
Tablica 12: Tipovi komunikacije s obzirom na broj primatelja i pošiljaoca

Kod svih tipova komunikacije osim privatne, prisutno je natjecanje između pošiljaoca i natjecanje između primatelja. Poruke su poredane u spremniku onim redoslijedom kojim su u njega stavljene. Zbog toga će se poruka koja je prije stavljena u spremnik prije i pročitati.

Kad više raspodijeljenih programa čita poruke iz istog spremnika, poruku će pročitati samo program koji je prvi pročitao poruku jer se poruka nakon čitanja briše iz spremnika.

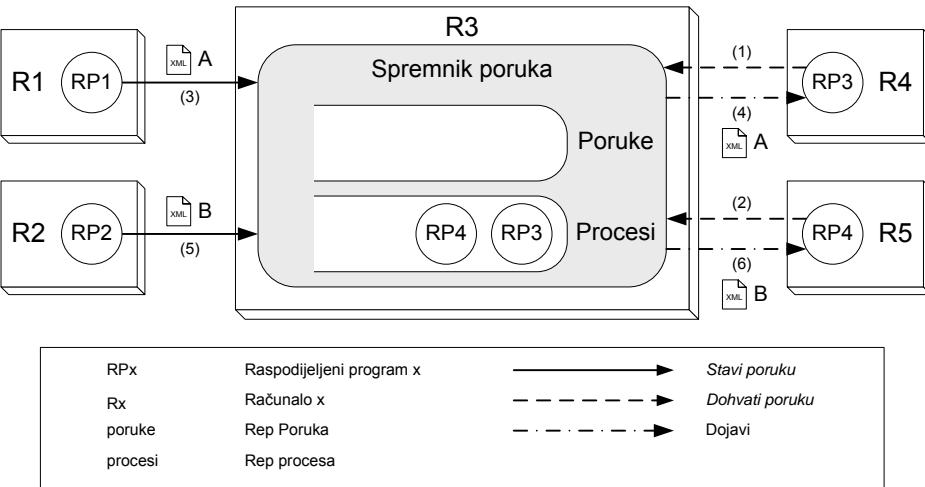
5.4.5 Okolina spremnika poruka

Na slici 22 je prikazana okolina spremnika poruka. Korisnici spremnika poruka mogu biti mrežne usluge ili raspodijeljene aplikacije i dijele se na proizvođače poruka i potrošače poruka. Proizvođači poruka stavljaju poruke u spremnik koje potrošači dohvaćaju iz spremnika. Potrošači poruka moraju imati ostvareno sučelje za dojavljivanje ako žele biti obaviješteni o dolasku poruka.



Slika 22: Okolina spremnika poruka

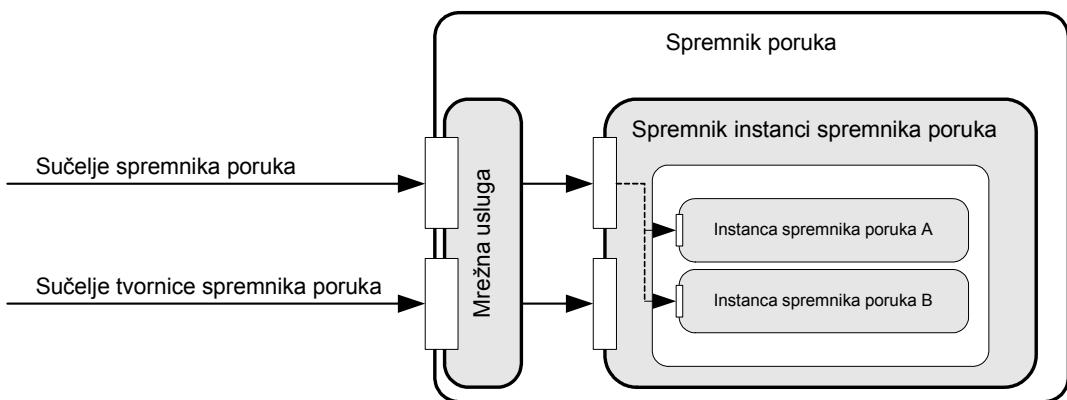
Na slici 23 je prikazan primjer upotrebe spremnika poruka u raspodijeljenoj aplikaciji. Spremnik je instaliran na računalo R_3 . Na računala R_1 i R_2 nalaze se raspodijeljeni programi RP_1 i RP_2 koji su proizvođači poruka, a na računalima R_4 i R_5 su instalirani programi RP_3 i RP_4 koji su potrošači. Korištenjem spremnika poruka za komunikaciju ostvarena je vremenski i referencijski nepovezana komunikacija programa RP_1 i RP_2 s programima RP_3 i RP_4 . Programi RP_3 i RP_4 se međusobno natječu jer čitaju poruke iz istog spremnika. Zbog toga se prije izvođenja ne zna koliko će poruka primiti koji spremnik i koje će to poruke biti. U primjeru je program RP_3 prvi zatražio poruku od spremnika (1), a nakon njega to je isto učinio i program RP_4 (2). Spremnik je prazan pa su oba zahtjeva pohranjena u repu za čekanje. Nakon što mu program RP_1 pošalje poruku (3), spremnik će obavijestiti program RP_3 i izbrisati ga iz repa za čekanje. Kad program RP_2 proizvede poruku (5), spremnik će o tome obavijestiti program RP_4 (6) te ga obrisati iz repa za čekanje.



Slika 23: Primjer upotrebe spremnika poruka

5.5 Ostvarenje spremnika poruka

Za ostvarenje spremnika poruka korišteni su isti programski alati i tehnologije kao i za ostvarenje binarnog i općeg semafora. Na slici 24 prikazana je programska arhitektura spremnika poruka. U njoj su istaknuta tri C# razreda koja ostvaruju njegovu funkcionalnost. Mrežna usluga pruža osnovno sučelje spremnika poruka i postupke za njegovo stvaranje i uništavanje. Ostvarena je razredom *MailBoxWebService*. Instance spremnika poruka su razreda *MailBoxInstance* te se spremaju u spremnik razreda *MailBoxContainer*.



Slika 24: Programska arhitektura spremnika poruka

5.5.1 Sučelje mrežne usluge spremnika poruka

Razred *MailBoxWebService* predstavlja mrežnu uslugu pa je izведен iz klase *WebService*. Kao kod ostvarenja binarnog i općeg semafora i za ostvarenje spremnika poruka korišteni su atributi tipa *WebServiceAttribute*, *WebMethodAttribute* i *SoapHeaderAttribute* za definiranje mrežne usluge, njezinih metoda i parametara od tih metoda.

Primjer SOAP poruke kojom se poziva metoda *PutMessage* mrežne usluge spremnika poruka prikazan je na slici 25. U zaglavlju SOAP poruke je, kao i kod metoda binarnog i općeg semafora, adresa instance spremnika poruka, koja je ostvarena u jeziku C# pomoću razreda *ResourceID*. Ostali parametri prenose se preko tijela SOAP poruke.

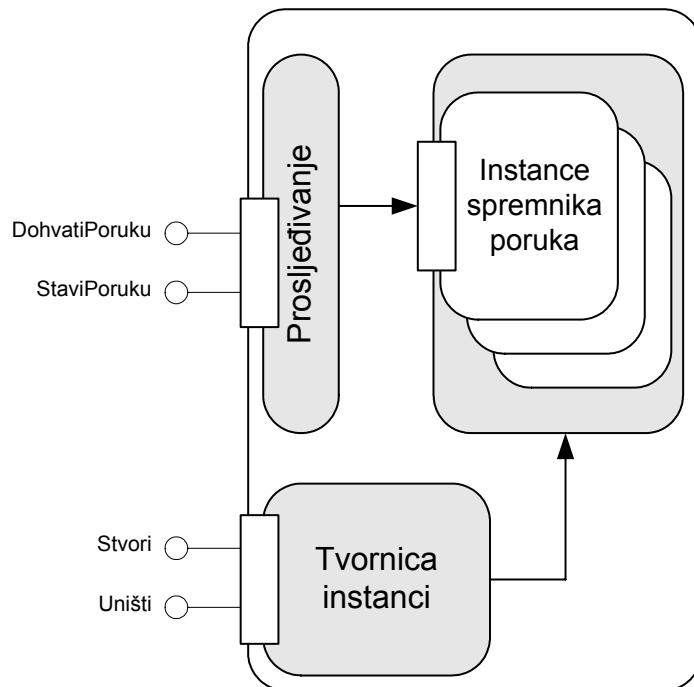
```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <ResourceID xmlns="www.ris.fer.hr/CoopetitionServices/MailBox"/>
    </soap:Header>
    <soap:Body>
        <PutMessage xmlns=" www.ris.fer.hr/ CoopetitionServices/MailBox">
            <mbMessage type="string" controlParam="string">xml</mbMessage>
            <EndpointReference
                xmlns="http://schemas.xmlsoap.org/ws/2004/03/addressing">
                <Address />
                <ReferenceProperties>xml</ReferenceProperties>
                <PortType />
                <ServiceName PortName="NCName" />
                xml
            </EndpointReference>
        </PutMessage>
    </soap:Body>
</soap:Envelope>
```

Slika 25: SOAP omotnica za poziv metode PutMessage

5.5.2 Spremnik instanci spremnika poruka

Razred *MailBoxContainer* ostvaruje funkcionalnost spremnika instanci spremnika poruka. *MailBoxContainer* kao i spremnik instanci binarnog ili općeg semafora omogućuje

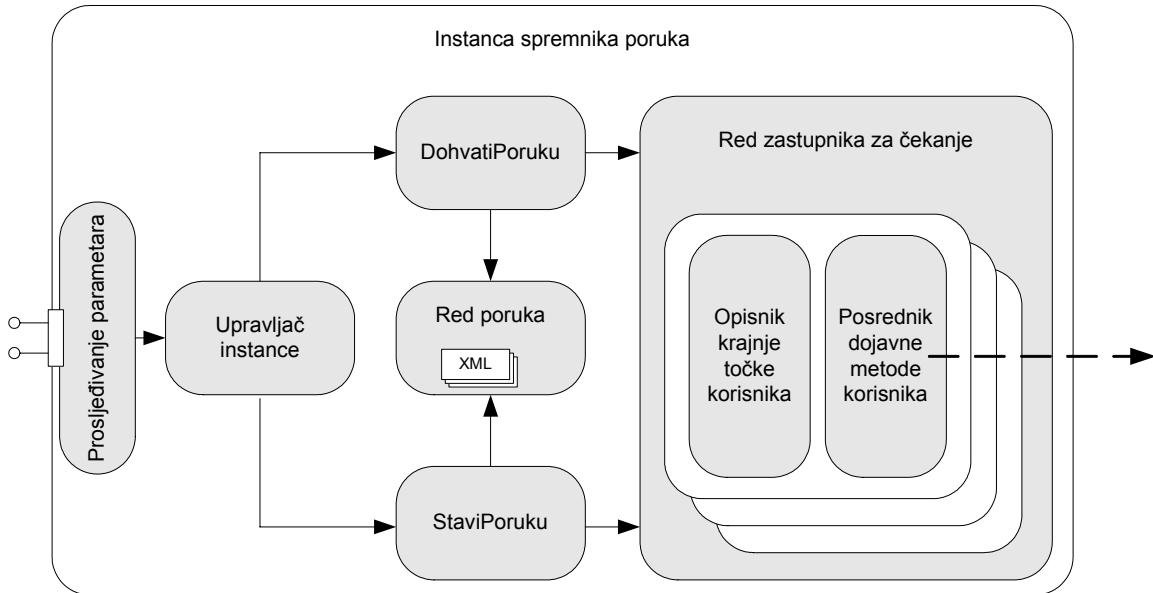
stvaranje, uništavanje te korištenje instanci spremnika poruka. Instance spremnika poruka su spremljene u strukturu podataka rječnik koja je ostvarena razredom *Hashtable*. Slika 26 prikazuje programsku arhitekturu spremnika instanci. Spremnik instanci je pohranjen u *Application* objekt ASP .NET procesa.



Slika 26: Programska arhitektura spremnika instanci spremnika poruka

5.5.3 Instanca spremnika poruka

Instanca spremnika poruka ostvarena je razredom *MailBoxInstance*. Slika 27 prikazuje arhitekturu instance spremnika poruka. Pozivom neke metode instance spremnika poruka parametri se zapisuju u pomoćne varijable te se signalizira upravljač instance. Upravljač instance se izvršava u vlastitoj dretvi koja omogućuje da se program koji je pozvao neku metodu spremnika poruka što prije nastavi izvoditi, dok upravljač instance može dalje obavljati operaciju kao što je dojavljivanje korisnika iz repa za čekanje. Upravljač instance obavlja sve operacije spremnika poruka osim stvaranja i uništavanja.



Slika 27: Programska arhitektura instance spremnika poruka

Pristigle se poruke pohranjuju u repu poruka. Poruke su ostvarene razredom *MailBoxMessage*. Kad korisnik pokuša pročitati poruku iz spremnika, upravljač instance prvo pokuša dohvatiti poruku iz repa. Ako je rep prazan, onda se stvara posebna poruka koja obavještava korisnika da je spremnik poruka prazan. Ako je vrijeme čekanja postavljeno na beskonačno, tada se u rep za čekanje dodaje objekt tipa *MailBoxProxyWrapper* koji pamti krajnju točku korisnika i WSDL njegova sučelja za dojavljivanje.

5.5.4 Ostvarenje mehanizma dojave

Za ostvarenje mehanizma dojavljivanja koristi se razred *MailBoxProxyWrapper*. On omogućuje pristupanje mrežnoj usluzi za dojavljivanje opisanoj WSDL-om koji je zahtjevatelj posao kao parametar poziva funkcije *GetMessage*. Iz opisa dojavne usluge stvara se C# kôd. Stvoreni kôd je razred s tri metode. Prva služi za sinkrono pozivanje metode mrežne usluge, a druge dvije za asinkrono pozivanje. Iako dojavna metoda može imati proizvoljno ime, njeni se parametri ne mogu mijenjati pa se C# kôd stvara samo umetanjem odgovarajućeg imena mrežne usluge, imena dojavne metode i prostora imena u predložak kôda koji je statički stvoren alatom WSDL. Postupak je jednak kao i kod mehanizma dojavljivanja za binarni i opći semafor. Tako stvoreni kôd se nakon toga prevodi u jezik MSIL (Microsoft Intermediate Language) pomoću .Net razreda *CSharpCodeProvider*. Rezultat prevođenja je objekt razreda *Assembly* koji sadrži prevedeni kôd. .Net reflection mehanizam omogućuje pozivanje metode za dojavljivanje.

Mehanizam dojave koristi se za dostavu poruke korisnicima koji su u repu za čekanje. Dojavna metoda ima dva parametra. Prvi parametar je krajnja točka spremnika poruka. Ona je potrebna jer korisnik može koristiti istu dojavnu metodu za primanje obavijesti od više korisnika. Drugi parametar je sama poruka. Metoda za dojavu poziva se sinkrono. Kad se spremnik poruka uništava, svi korisnici koji su u repu za čekanje obavještavaju da će spremnik biti uništen. Svim se korisnicima iz repa za čekanje pošalje kontrolna poruka koja ih obavještava o uništenju spremnika poruka. Pri tome se koriste asinkroni pozivi mrežne usluge za dojavu zbog brzine obavještavanja.

5.5.5 Format poruke

Poruke koje se razmjenjuju korištenjem spremnika poruka te poruke koje sam spremnik stvara da bi obavijestio korisnika o stanju u kojem se nalazi ostvarene su pomoću razreda *MailBoxMessage*. Korištenjem .NET svojstava za upravljanje serijalizacijom u XML postignut je format poruke prikazan na slici 28. Tip poruke je opisan svojstvima oznake `<mbMessage>`. Svojstvo *type* određuje kojeg je tipa poruka. Poruka može biti podatkovna ako je *type = data* ili kontrolna ako je *type = control*.

```
<mbMessage type="tip poruke" controlParam="tip kontrolne poruke" >  
Korisnički XML  
</mbMessage>
```

Slika 28: XML poruke spremnika poruka

U tablici 13 su prikazani tipovi poruka i njihovi kontrolni parametri. Ako je tip poruke *data*, tada se unutar oznaka `<mbMessage>` i `</mbMessage>` nalazi proizvoljna XML poruka. Za kontrolne poruke koristi se svojstvo *controlParam* oznake `<mbMessage>` koje može imati vrijednosti *noMessage*, ako je spremnik poruka prazan kad korisnik pokuša dohvatiti poruku, *mbDestroyed*, ako se obavještava korisnik koji čeka poruku o uništenju spremnika poruka i *mbError* ako je došlo do greške u radu spremnika poruka. Kontrolne poruke ne sadrže dodatne informacije osim poruke tipa *mbError* koja sadrži opis C# iznimke koji se koristi kod ispravljanja grešaka spremnika poruka i aplikacija koje ga koriste.

| Tip poruke | controlParam | Sadržaj poruke |
|-------------------|---------------------|-----------------------|
| Data | Ne koristi se | Korisnička XML poruka |
| Control | noMessage | Ne koristi se |
| | mbDestroyed | Ne koristi se |
| | mbError | Opis C# iznimke |

Tablica 13: Tipovi poruka

6 Zaključak

Uporaba arhitekture zasnovane na uslugama olakšava izgradnju raspodijeljenih aplikacija. Međutim, postojeći sustavi za modeliranje poslovnih procesa korištenjem mrežnih usluga koriste centralizirano upravljanje izvođenjem raspodijeljenih aplikacija. Za izgradnju sustava s raspodijeljenim upravljanjem potrebni su mehanizmi komunikacije i sinkronizacije. U sklopu ovog diplomskog rada proučeni su mehanizmi sinkronizacije i komunikaciju u operacijskim sustavima i postojećim raspodijeljenim posredničkim sustavima. Odabrana su i ostvarena dva sinkronizacijska i jedan komunikacijski mehanizam. Njihova namjena je potpora izgradnji raspodijeljenih aplikacija za CBDS.

Za testiranje svakog od tih mehanizama ostvarena je ispitna aplikacija koja je omogućila stvaranje, uništavanje i simulaciju primjene pojedinih mehanizama. Stanje instanci i sve akcije su prikazane korištenjem grafičkog sučelja. Sva tri mehanizma pokazala su točnost i pouzdanost pri radu. Pri njihovoj upotrebi u izgradnji nekoliko demonstracijskih aplikacija u sklopu projekta Cro-Grid izgrađeni mehanizmi su bili dovoljni za ostvarenje željenih sinkronizacijskih i komunikacijskih mehanizama te za ostvarivanje suradnje i natjecanja raspodijeljenih programa u CBDS-u.

Iako je tehnologija korištena za izgradnju ostvarenih mehanizama razvila tvrtka Microsoft, korištenje općeprihvaćenih standarda osigurava njihovu primjenu i za izgradnju raspodijeljenih sustava čije su komponente ostvarene pomoću drugih alata za razvoj mrežnih usluga i drugih programskih jezika. Jedini uvjet za korištenje je pridržavanje WS-* skupa standarda. Jedan od glavnih razloga za odabir tehnologija za ostvarenje mehanizama komunikacije i sinkronizacije je jednostavnost. Nedostatak odabranih tehnologija su lošije performanse kod prvog korištenja neke usluge uzrokovane izvođenjem programa u virtualnom stroju, ali brzina izvođenja nije bila glavni kriterij kod ostvarenja CBDS-a.

Ostvarene usluge su dovoljno jednostavne za korištenje, a pri tome su dovoljno općenite da se mogu iskoristiti i za ostvarenje aplikacija s složenijim modelima sinkronizacije i komunikacije. Ponovna iskoristivost ostvarenih mehanizama za sinkronizaciju i

komunikaciju u raspodijeljenim aplikacijama je velika i njihova upotreba nije ograničena na aplikacije izgrađene za CBDS.

Literatura

- [1] Tanenbaum, Andrew S.: „**Distributed Systems Principles and Paradigms**“, Prentice Hall, New Jersey, 2002.
- [2] Tanenbaum, Andrew S.: „**Modern Operating Systems**“, Second Edition, Prentice Hall, New Jersey, 2002.
- [3] Tanenbaum, Andrew S.: „**Computer Networks**“, Fourth Edition, Prentice Hall, New Jersey, 2003.
- [4] Andro Milanović, Siniša Srbljić, Daniel Skrobo, Davor Čapalija, Saša Rešković: „**Coopetition Mechanisms for Service-Oriented Distributed System**”, The 3rd International Conference on Computing, Communications and Control Technologies (CCCT '05), Austin, SAD, srpanj, 2005.
- [5] Singh, Munindar P., Michael N. Huhns: „**Service-Oriented Computing: Key Concepts and Principles**”, IEEE Internet Computing, siječanj/veljača 2005.
- [6] Jason Bloomberg: „**Principles of SOA**“, Application Development Trends, ožujak 2003.
- [7] BEA: „**Service-Oriented Architecture: Solution Accelerator**“, Version 1.0 revision 2
- [8] Mike P. Papazoglu: "Service-Oriented Computing: Concepts, Characteristics and Directions"
- [9] Budin, L.: „**Predavanja iz predmeta: Operacijski sustavi**“, Fakultet elektrotehnike i računarstva, Zagreb, 2003.
- [10] Glavinić, V.: „**Predavanja iz predmeta: Mreže računala**“, Fakultet elektrotehnike i računarstva, Zagreb, 2002.
- [11] M. P. Papazoglou and D. Georgakopoulos: „**Service-Oriented Computing**“, Communication of the ACM, October 2003/Vol. 46 No. 10

[12] W3C Working Draft 14 November 2002, Web Services Architecture,

<http://www.w3.org/TR/2002/WD-ws-arch-20021114/>