

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Daniel Skrobo

**RASPODIJELJENO USPOREDNO
INTERPRETIRANJE PROGRAMA U
ARHITEKTURAMA ZASNOVANIM NA
USLUGAMA**

MAGISTARSKI RAD

Zagreb, 2006.

Magistarski rad izrađen je na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sisteme Fakulteta elektrotehnike i računarstva

Mentor: Prof. dr. sc. Siniša Srbljić

Magistarski rad ima 180 stranica.

Rad br.: _____

Povjerenstvo za ocjenu u sastavu:

1. Prof. dr. sc. Nikola Bogunović – predsjednik
2. Prof. dr. sc. Siniša Srbljić – mentor
3. Doc. dr. sc. Saša Dešić – Ericsson Nikola Tesla Zagreb

Povjerenstvo za obranu u sastavu:

1. Prof. dr. sc. Nikola Bogunović – predsjednik
2. Prof. dr. sc. Siniša Srbljić – mentor
3. Doc. dr. sc. Saša Dešić – Ericsson Nikola Tesla Zagreb

Datum obrane: 25. siječnja 2006.

Zahvaljujem se...

prof. dr. sc. Siniši Srbljiću na prenešenom iskustvu, vrijednim savjetima i pruženoj prilici za rad u dinamičnoj i poticajnoj okolini.

mr. sc. Andri Milanoviću na pruženoj pomoći i praktičnim savjetima koji su omogućili izradu ovog rada.

mr. sc. Ivanu Bencu i mr. sc. Ivanu Skuliberu za korektnu i uspješnu suradnju na projektu, te upoznavanje s "corporate svijetom" naše struke.

Ivanu Gavranu dipl. ing., Matiji Podravcu dipl. ing., Miroslavu Popoviću dipl. ing., Dejanu Škvorcu dipl. ing., za svu pomoć koju su mi pružili tijekom zajedničkog rada na CroGrid projektu.

i na kraju, obitelji i prijateljima koji su bili uz mene sve ove godine.



Sadržaj

1	Uvod	1
2	Računarstvo zasnovano na uslugama.....	4
2.1	Modeli za razvoj programskih sustava	6
2.1.1	Pregled modela	7
2.1.2	Usporedba modela	10
2.2	Programske arhitekture zasnovane na uslugama	12
2.2.1	Osnovne značajke usluga.....	13
2.2.2	Osnovna arhitektura zasnovana na uslugama	15
2.2.3	Proširena arhitektura zasnovana na uslugama	16
2.3	Primjena računarstva zasnovanog na uslugama	17
2.3.1	Sustavi za upravljanje poslovnim procesima.....	18
2.3.2	Sustavi za izgradnju spletova računala	20
2.3.3	Uslužni računalni sustavi.....	21
2.3.4	Sustavi ravnopravnih sudionika.....	22
2.4	Tehnologije za ostvarivanje sustava zasnovanih na uslugama	22
2.4.1	Tehnologija Web Services	23
2.4.2	Electronic Business using eXtensible Markup Language.....	28
3	Kompozicija usluga	31
3.1	Zahtjevi kompozicije usluga.....	32
3.2	Metode za kompoziciju usluga	34
3.2.1	Kompozicija usluga zasnovana na opisima procesa	36
3.2.2	Uslužne komponente.....	45
3.2.3	Petrijeve mreže	49
3.2.4	Procesne algebre	53
3.2.5	Konačni automati.....	58
3.2.6	Semantičke usluge	63
4	Sustavi za izgradnju raspodijeljenih aplikacija kompozicijom usluga	71
4.1	BizTalk Server	71
4.2	BPEL Process Manager	75
4.3	ActiveBPEL	77
4.4	Self-Serv	79
5	Raspodijeljeni interpretator programa	83
5.1	Oblikovanje raspodijeljenih aplikacija zasnovanih na uslugama	84
5.1.1	Aplikacijske usluge.....	85
5.1.2	Raspodijeljeni programi.....	85
5.1.3	Koordinacijski mehanizmi	85
5.1.4	Obrasci za oblikovanje raspodijeljenih aplikacija	90

5.2	Programski jezik za suradnju i natjecanje	99
5.2.1	Struktura raspodijeljenog programa napisanog u programskom jeziku za suradnju i natjecanje	101
5.2.2	Naredbe deklaracije varijabli i simboličkih imena usluga.....	104
5.2.3	Naredbe komunikacije	106
5.2.4	Naredbe upravljanja tijekom izvođenja	109
5.2.5	Naredbe upravljanja podacima	112
5.3	Višestupanjsko prevođenje programa.....	113
5.4	Elementi raspodijeljenog interpretatora programa	114
5.5	Arhitektura rasporedišvača programa	116
5.6	Arhitektura interpretatora programa	117
5.6.1	Prevoditelj programa.....	119
5.6.2	Spremnik programa.....	121
5.6.3	Raspodijeljeni program.....	121
6	Programsko ostvarenje raspodijeljenog interpretatora programa ...	125
6.1	Programska arhitektura raspodijeljenog interpretatora.....	126
6.1.1	Interpretator programa	130
6.1.2	Raspodijeljeni program.....	148
6.2	Postavljanje i korištenje raspodijeljenog interpretatora programa ..	151
7	Mjerenje svojstava interpretatora programa	156
7.1	Okolina za provođenje mjerena	156
7.2	Rezultati mjerena	158
8	Zaključak.....	162
9	Literatura	164
10	Životopis	173
11	Sažetak.....	174
12	Summary	175
13	Ključne riječi.....	176
A	Definicija programskog jezika za suradnju i natjecanje	177
B	Primjer raspodijeljenog programa napisanog u jeziku za suradnju i natjecanje	178

1. poglavlje

Uvod

Globalna mreža Internet povezuje raznorodne računalne mreže. Brzi tehnološki razvoj omogućio je primjenu globalne mreže Internet u svrhu razmjene, objavljivanja i pretraživanja informacija vezanih uz različite ljudske djelatnosti. Osnovne funkcionalnosti globalne mreže Internet proširene su izgradnjom različitih usluga, kao što su usluge za elektroničko poslovanje (engl. *e-business*), elektroničku trgovinu (engl. *e-commerce*) i elektroničko obrazovanje (engl. *e-learning*). Danas globalna mreža Internet čini osnovicu za izgradnju nove vrste raspodijeljenih programskih sustava koji se grade primjenom načela računarstva zasnovanog na uslugama (engl. *service oriented computing*). Usluge su nezavisne programske jedinice koje omogućavaju pristup skupu funkcionalnosti putem standardnih sučelja i protokola. Raspodijeljeni programski sustavi grade se međusobnim povezivanjem različitih usluga dostupnih na globalnoj mreži Internet.

Kompozicija usluga jedan je od osnovnih mehanizama koji se koristi tijekom izgradnje raspodijeljenih programskih sustava zasnovanih na uslugama. Kompozicija usluga omogućava izgradnju programskih sustava koji ostvaruju složene funkcionalnosti međusobnim povezivanjem postojećih usluga dostupnih u globalnoj mreži Internet. Osnovni zahtjevi za kompoziciju usluga su ostvarivanje tražene kvalitete, ispravnost, sigurnost i svojstvo razmernog rasta složenih usluga. Područje kompozicije usluga trenutno se intenzivno istražuje i ne postoji sveobuhvatna metoda za kompoziciju usluga koja ispunjava sve navedene zahtjeve. Najšire prihvaćena i u praksi najčešće korištena metoda za kompoziciju usluga je kompozicija usluga zasnovana na opisima procesa. Opis procesa složene usluge sadrži definicije obrazaca za razmjenu poruka i pretvorbu sadržaja poruka koje usluge razmjenjuju kako bi ostvarile složenu uslugu. Opisi procesa mogu biti ostvareni primjenom metoda koreografije usluga ili orkestracije usluge. Orkestracija usluga ostvaruje se središnjim upravljanjem tijekom izvođenja složene usluge iz gledišta jednog procesa. Koreografija usluga ostvaruje se izgradnjom opisa obrazaca razmjene poruka za svaku uslugu koja čini složenu uslugu. Opisi procesa složene usluge izvode se primjenom sustava za izvođenje složenih usluga zasnovanih na opisima procesa.

Na tržištu je dostupno nekoliko komercijalnih i javno dostupnih sustava za izvođenje složenih usluga zasnovanih na opisima procesa. Većina dostupnih sustava za izvođenje

složenih usluga ostvareni su kao središnji poslužitelji za izvođenje složenih usluga. Središnji poslužitelji imaju složenu građu, koriste središnji model za upravljanje tijekom izvođenja složenih usluga, predstavljaju točku nepouzdanosti (engl. *single point of failure*) složenih usluga i nemaju mogućnost razmjernog rasta (engl. *scalability*). Nadalje, poslužitelji složenih usluga također zahtijevaju korištenje složenih jezika za opis procesa.

U magistarskom radu opisan je hibridni model za izgradnju raspodijeljenih aplikacija zasnovanih na uslugama koji je ostvaren kombinacijom metoda koreografije i orkestracije usluga. Hibridni model omogućava oblikovanje i izgradnju aplikacija zasnovanih na uslugama s raspodijeljenim upravljanjem tijekom izvođenja. U magistarskom radu opisana je arhitektura i programsko ostvarenje raspodijeljenog interpretatora programa. Raspodijeljeni interpretator programa omogućava postavljanje i izvođenje izgrađenih raspodijeljenih aplikacija zasnovanih na uslugama. Raspodijeljeni interpretator programa ostvaren je kao skup jednostavnih i slabo povezanih interpretatora programa postavljenih na skupu radnih računala. Hibridni model za izgradnju raspodijeljenih aplikacija i raspodijeljeni interpretator programa omogućavaju oblikovanje, izgradnju, postavljanje i izvođenje raspodijeljenih aplikacija zasnovanih na uslugama sa svojstvima razmjernog rasta i otpornosti na pogreške.

Drugo poglavlje rada sadrži opis osnovnih načela i postupaka računarstva zasnovanog na uslugama koji se primjenjuju tijekom izgradnje raspodijeljenih računalnih sustava zasnovanih na uslugama. U trećem poglavlju opisana je razredba metoda za kompoziciju usluga na statičke i dinamičke. Za svaku od opisanih metoda kompozicije usluga navedeni su prednosti i nedostaci. U četvrtom poglavlju opisani su trenutno najpoznatiji na tržištu dostupni sustavi za izgradnju, postavljanje i izvođenje raspodijeljenih aplikacija zasnovanih na uslugama. Opisana je arhitektura sustava *BizTalk Server*, *BPEL Process Manager*, *ActiveBPEL* i *Self-Serv*.

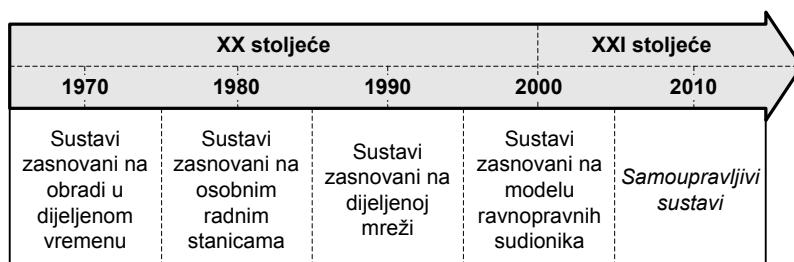
U petom poglavlju opisan je hibridni model izgradnje raspodijeljenih aplikacija zasnovanih na uslugama. Opis modela uključuje definiciju jezika *CL* (*Coopetition Language*) koji omogućava izgradnju raspodijeljenih aplikacija kompozicijom usluga u globalnoj mreži Internet. Osim hibridnog modela izgradnje raspodijeljenih aplikacija, opisana je arhitektura raspodijeljenog interpretatora programa koji omogućava izvođenje raspodijeljenih aplikacija zasnovanih na uslugama. U šestom poglavlju opisani su elementi programskog ostvarenja raspodijeljenog intereptatora programa. U sedmom poglavlju opisani su rezultati mjerjenja učinkovitosti rada raspodijeljenog interpretatora. Zaključak

magistarskog rada i budući rad u području raspodijeljenog interpretiranja programa navedeni su u osmom poglavlju.

2. poglavlje

Računarstvo zasnovano na uslugama

Drugu polovicu 20. stoljeća obilježio je ubrzani razvoj računalne industrije potaknut uspješnom praktičnom primjenom rezultata istraživačkog rada u području računalnih znanosti. Ubrzani razvoj računalnih znanosti i industrije omogućio je primjenu računalnih sustava u sve većem broju ljudskih djelatnosti. Raznovrsni računalni sustavi danas se primjenjuju u mnogim granama ljudskih djelatnosti kao što su poslovanje, znanost, obrazovanje, zabava, državna uprava i liječnička skrb.



Slika 1: Pregled razvoja računalnih sustava

Slika 1 prikazuje pregled razvoja računalnih sustava tijekom druge polovice 20. stoljeća i početkom 21. stoljeća [1] [2]. Značajno smanjenje veličine i troškova izrade sklopova zasnovanih na tranzistoru omogućili su početkom 60-tih godina 20. stoljeća sve širu komercijalnu primjenu računala [3]. Povijesni razvoj računalnih sustava podijeljen je u četiri razdoblja koja su obilježena primjenom računalnih sustava zasnovanih na različitim arhitekturama: sustavi zasnovani na obradi u dijeljenom vremenu (engl. *time-shared systems*), sustavi zasnovani na radnim stanicama (engl. *workstations*), sustavi zasnovani na dijeljenoj mreži (engl. *shared network systems*) i sustavi zasnovani na modelu ravnopravnih sudionika (engl. *peer-to-peer systems*).

Sustavi zasnovani na obradi u dijeljenom vremenu omogućavaju istovremeno i nezavisno izvođenje zadataka dvaju ili više korisnika. Nadalje, sustavi zasnovani na obradi u dijeljenom vremenu ostvaruju se korištenjem središnjeg poslužiteljskog računala (engl. *mainframe*) i skupa međusobno nezavisnih terminala (engl. *terminals*). Poslužiteljsko računalo omogućava spremanje i izvođenje različitih aplikacija. Terminali omogućavaju korisnicima pristup poslužiteljskom računalu, upravljanje izvođenjem aplikacija i prikazivanje rezultata. Za razliku od poslužiteljskog računala, terminali nemaju mogućnost

lokalnog izvođenja aplikacija i spremanja rezultata izvođenja. Razvoj integriranih sklopova i smanjenje troškova njihove izrade omogućili su izgradnju sustava zasnovanih na osobnim radnim stanicama. Za razliku od terminala, radne stanice sadrže jedinice za lokalno spremanje i izvođenje korisničkih aplikacija. Međusobnim povezivanjem skupa radnih stanica primjenom računalnih mreža ostvareni su sustavi zasnovani na dijeljenoj mreži. Prvi sustavi zasnovani na dijeljenoj mreži ostvareni su uporabom lokalnih računalnih mreža. Nadalje, razvoj globalne mreže Internet omogućio je povezivanje radnih stanic diljem svijeta. Najčešće korišten model za izgradnju sustava zasnovanih na dijeljenoj mreži je model korisnik-poslužitelj (engl. *client-server model*) [4]. Sustavi zasnovani na modelu korisnik-poslužitelj uključuju središnji poslužitelj (engl. *server*) i skup nezavisnih korisničkih radnih stаница (engl. *client*). Poslužitelj omogućava spremanje i izvođenje korisničkih aplikacija opće namjene, dok korisnička računala omogućavaju pristup i uporabu aplikacija dostupnih na poslužitelju. Korisnička računala imaju mogućnost lokalne obrade podataka, spremanja međurezultata i prikaza rezultata obrade. Proširivanjem svojstava modela korisnik-poslužitelj omogućen je razvoj sustava zasnovanih na modelu ravnopravnih sudionika [5]. Sustavi zasnovani na modelu ravnopravnih sudionika sadrže skup ravnopravnih računala koja međusobno dijele računalna sredstva kao što su diskovni prostor, računalno vrijeme, komunikacijske kanale ili podatke. U sustavima zasnovanim na modelu ravnopravnih sudionika ne postoji središnje upravljačko mjesto sustava i ne postoji podjela uloga na poslužiteljska i korisnička računala. Sva računala unutar sustava ostvaruju dijeljeni skup funkcionalnosti i istovremeno su korisnici i poslužitelji za ostala računala u sustavu.

Najnoviju generaciju računalnih sustava čine *samoupravlјivi sustavi* (engl. *Autonomic systems*) [6] koji se zasnivaju na aktivnim (engl. *active*), samostalnim (engl. *autonomous*), slabo povezanim (engl. *loosely coupled*), te raznovrsnim komponentama. *Samoupravlјivi sustavi* zasnivaju se na načelu samostalnog upravljanja (engl. *self-management*) koje omogućava sustavima da samostalno nadgledaju, upravljaju i prilagođavaju vlastito ponašanje prema uvjetima u okolini. Osnovna svojstva *samoupravlјivih sustava* su *samostalno određivanje postavki* (engl. *self-configuration*), *samostalno optimiranje* (engl. *self-optimization*), *samostalan oporavak* (engl. *self-healing*) i *samostalna zaštita* (engl. *self-protection*). Svojstvo *samostalnog određivanja postavki* omogućava sustavima automatsko upravljanje vlastitim postavkama tijekom izvođenja u svrhu prilagođavanja uvjetima u okolini. Svojstvo *samostalnog optimiranja* omogućava sustavima nadgledanje i prilagođavanje rada vlastitih komponenata s ciljem ostvarivanja učinkovitog izvođenja sustava. Svojstvo *samostalnog oporavka* omogućava komponentama sustava da samostalno otkrivaju i uklanjaju pogreške tijekom izvođenja sustava. Svojstvo

samostalne zaštite omogućava sustavima samostalno otkrivanje i zaštitu od zlonamjernih napada i djelovanja koja ugrožavaju cjelovitost sustava.

2.1 Modeli za razvoj programskih sustava

Postupci za oblikovanje (engl. *design*), ostvarivanje (engl. *implementation*), ispitivanje (engl. *testing*) i održavanje (engl. *maintenance*) programskih sustava iznimno su složeni. Funkcijske i nefunkcijske značajke programskih sustava u pravilu su određene raznovrsnim i složenim, a ponekad i proturječnim zahtjevima. Tijekom izgradnje većine programskih sustava najčešće je potrebno ispuniti sljedeće zahtjeve: učinkovitost (engl. *performance*), vremenska ograničenja (engl. *real-time constraints*), otpornost na pogreške (engl. *fault tolerance*), sigurnost (engl. *security*), prilagodljivost (engl. *adaptability*), proširivost (engl. *extensibility*), uskladivost (engl. *interoperability*), ponovna iskoristivost (engl. *reusability*), vrijeme isporuke (engl. *time to market*) i ukupna cijena izgradnje (engl. *total development cost*) [1].

Zahtjev učinkovitosti nalaže neprekidan i pravilan rad računalnog sustava uslijed značajnog povećanja broja korisnika i zahtjeva koje sustav mora poslužiti. Zahtjevi koji uključuju vremenska ograničenja određuju vremenske odnose u radu komponenata sustava koji moraju biti ispunjeni tijekom rada računalnog sustava. Zahtjev za otpornošću na pogreške nalaže pravilan rad sustava unatoč ispadima i neispravnom radu pojedinih komponenti sustava. Zahtjev za sigurnošću nalaže očuvanje tajnosti i privatnosti podataka spremljenih u računalnom sustavu, te sprečavanje neovlaštenog pristupa i uporabe usluga sustava. Zahtjev za prilagodljivošću nalaže mogućnost primjene računalnog sustava u različitim okolinama i uvjetima rada. Zahtjev proširivosti nalaže mogućnost proširivanja postojećih funkcionalnosti sustava uvođenjem novih komponenata. Zahtjev uskladivosti nalaže mogućnost povezivanja i suradnje sustava s raznovrsnim računalnim sustavima u svrhu ostvarivanja novih složenih funkcionalnosti. Zahtjev za ponovnom iskoristivošću nalaže mogućnost primjene komponenata postojećeg sustava u svrhu brže i učinkovitije izgradnje novih računalnih sustava. Zahtjev za ispunjavanjem vremena isporuke određuje vremenski rok unutar kojeg razvoj računalnog sustava mora biti završen, a izgrađeni sustav spreman za plasman na tržište. Zahtjev za ukupnom cijenom izgradnje određuje ukupni iznos sredstava koja je moguće utrošiti tijekom izgradnje programskog sustava.

Tijekom proteklih desetljeća istraživačkog rada, predloženi su različiti modeli za razvoj programskih sustava koji olakšavaju postupke oblikovanja, ostvarivanja, ispitivanja i

održavanja računalnih sustava. Najšire prihvaćeni modeli za razvoj programskih sustava su: model razvoja zasnovan na proceduralnoj paradigmri (engl. *procedural development*), model razvoja zasnovan na objektima (engl. *object-oriented development*), model razvoja zasnovan na komponentama (engl. *component-based development*) i najnoviji model razvoja zasnovan na uslugama (engl. *service-oriented development*).

2.1.1 Pregled modela

Model za razvoj programskih sustava zasnovan na proceduralnoj paradigmri jedan je od najstarijih modela. Nadalje, model se zasniva na primjeni jezika opće namjene koji omogućavaju strukturiranje logike programskog sustava u procedure. Procedure su zatvorene programske cjeline s definiranim ulazima i izlazima koje omogućavaju izgradnju programskih sustava primjenom metode *funkcijske dekompozicije* (engl. *functional decomposition*). Metoda *funkcijske dekompozicije* zasniva se na provođenju postupaka analize i dekompozicije složenih funkcionalnosti programskog sustava u skup jednostavnijih funkcionalnosti. Za svaku od prepoznatih jednostavnijih funkcionalnosti sustava ponovno se zasebno provodi analiza i dekompozicija. Postupak analize i dekompozicije provodi se rekurzivno sve dok se ne dosegnu osnovne funkcije programskog sustava koje je moguće izravno ostvariti procedurama odabranog jezika zasnovanog na proceduralnoj paradigmri. Jezici zasnovani na proceduralnoj paradigmri omogućavaju primjenu jedinstvenog skupa naredbi za opisivanje algoritma i struktura podataka. Programe je moguće pisati bez poznavanja pojedinosti sklopovske i programske arhitekture ciljnog računala. Nadalje, ostvareni programi su prenosivi i mogu se izvoditi na računalima različitih arhitektura primjenom odgovarajućih jezičnih procesora [7].

Razvojem jezika zasnovanih na proceduralnoj paradigmri, sredinom 80-tih godina prošlog stoljeća uveden je novi model razvoja programskih sustava zasnovan na objektima [8]. Osnovne značajke modela zasnovanog na objektima su primjena načela *umatanja* (engl. *encapsulation*), *prikrivanja podataka* (engl. *data hiding*), *nasljedivanja* (engl. *inheritance*) i *višeobličja* (engl. *polymorphism*). Načelo *umatanja* zasniva se na grupiranju procedura i struktura podataka koje procedure koriste tijekom izvođenja u jedinstvenu logičku cjelinu nazvanu *razred objekta* (engl. *object class*). Jedan *razred objekta* opisuje strukture podataka koje spremaju stanje objekta i metode koje ostali objekti mogu pozivati kako bi promijenili stanje objekta. Načelo *prikrivanja podataka* zasniva se na definiranju privatnih i javnih struktura podataka koje su dostupne pozivom metoda *razreda objekta*. Privatne strukture podataka i metode koristi isključivo objekt vlasnik tijekom izvođenja i nisu javno dostupne.

Javne strukture podataka i metode mogu koristiti svi objekti. Načelo *nasljeđivanja* omogućava izgradnju novih *razreda objekata* nasljeđivanjem metoda i struktura podatka postojećih *razreda objekata*. Primjena načela *nasljeđivanja* omogućava učinkovito ponovno iskorištavanje prethodno ostvarenih funkcionalnosti čime se značajno skraćuje ukupno vrijeme i trošak razvoja programskog sustava. Načelo *višeobličja* omogućava istovremeno definiranje više zamjenskih ostvarenja logike metoda *razreda objekta*. Koja logika će biti izvršena tijekom poziva metode ovisi o *razredu objekta* koji je korišten za ostvarivanje poziva metode. Primjena modela razvoja zasnovanog na objektima olakšava izgradnju programskih sustava zbog toga što omogućava strukturiranje funkcionalnosti programskog sustava, izgradnju programskog sustava ponovnim iskorištavanjem postojećih funkcionalnosti i izgradnju programskih sustava s prilagodljivim funkcionalnostima. Osim navedenih prednosti, primjena modela razvoja zasnovanog na objektima ima i nekoliko nedostataka. Mehanizme programskog jezika zasnovanog na objektima moguće je koristiti samo unutar granica odabranog jezika i nije ih moguće koristiti za izgradnju programskih sustava zasnovanih na raznovrsnim računalnim platformama i operacijskim sustavima. Tijekom oblikovanja stvarnih programskih sustava potrebno je ostvariti veliki broj *razreda objekata* sa složenim strukturama podataka i čvrstim međusobnim vezama. Zbog sitne razine zrnatosti objekata i njihove velike količine, vrlo je složeno učinkovito upravljati njihovim vezama i međuzavisnostima. Nemogućnost učinkovitog upravljanja vezama i međuzavisnostima objekata dodatno otežava postupak održavanja i unapređivanja funkcionalnosti programskog sustava.

Kako bi se povećala zrnatost elemenata za izgradnju programskih sustava i omogućilo uporabu raznovrsnih programskih jezika, operacijskih sustava i računalnih platformi, početkom 90-tih godina prošlog stoljeća uveden je model razvoja zasnovan na komponentama [9]. Komponenta je zatvorena programska cjelina koja ostvaruje skup funkcionalnosti dostupnih putem pristupnog sučelja. Primjena komponenti omogućava povezivanje raznovrsnih programskih jedinica u jedinstveni raspodijeljeni programski sustav. Uporabom komponenti moguće je ostvariti ponovnu iskoristivost dijelova funkcionalnosti ili funkcionalnosti cjelovitih programskih sustava. Zbog krupnije zrnatosti u odnosu na objekte, korištenje komponenata umanjuju složenost i vrijeme potrebno za izgradnju programskih sustava. Komponente se postavljaju, izvode i međusobno komuniciraju unutar zatvorenih i jednoobraznih (engl. *uniform*) komponentnih okolina. Obzirom da se koriste unutar jednoobrazne okoline, komponente imaju svojstvo pokretljivosti i primjenjuju se za izgradnju raspodijeljenih programskih sustava s značajkama otpornosti na pogreške i razmjernog rasta. Osim navedenih prednosti, primjena modela

razvoja zasnovanog na komponentama ima i nedostatke vezane uz zrnatost, uskladivost, prenosivost i održavanje komponenata [10]. Tijekom oblikovanja programskog sustava zasnovanog na komponentama, nužno je odrediti odgovarajuću zrnatost komponenata kako bi se ostvarilo učinkovito izvođenje programskog sustava. Korištenje jednoobrazne komponentne okoline umanjuje uskladivost sustava s novim komponentama. Svaku novu komponentu koja se ugrađuje u programski sustav potrebno je prethodno uskladiti s odabranom komponentnom okolinom. Nadalje, nove komponente potrebno je također prije ugradnje u sustav detaljno ispitati i uskladiti s postojećim komponentama kako bi se sačuvala cjelovitost sustava. Obzirom da su komponente zatvorene programske cjeline, u slučaju pogreške u radu određene komponente potrebno je stupiti u kontakt s proizvođačem komponente ili osigurati odgovarajuću zamjensku komponentu.

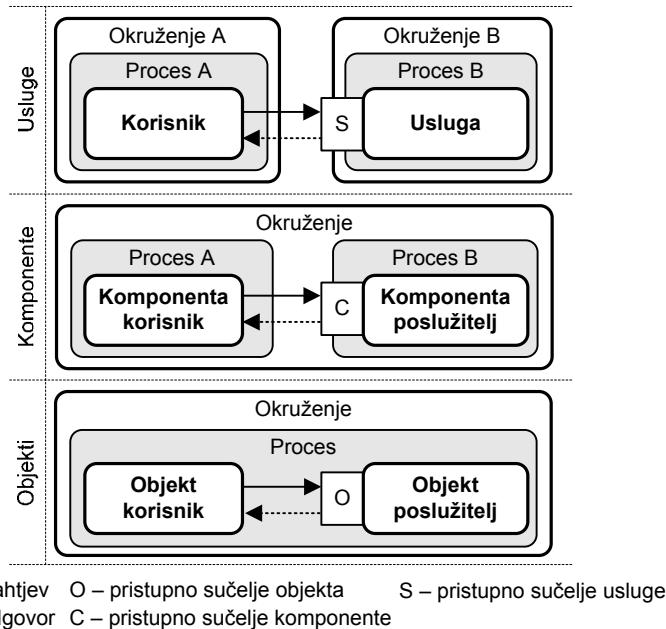
Najnovija istraživanja u području raspodijeljenog računarstva rezultirala su uvođenjem novog modela za razvoj računalnih sustava koji je zasnovan na uslugama [11] [12]. Usluge su samostalne, raspodijeljene i međusobno slabo povezane programske jedinice koje ostvaruju pristup različitim funkcionalnostima putem standardnih sučelja. Za razliku od programskih sustava zasnovanih na objektima i komponentama koji se oblikuju, grade i isporučuju kao proizvod, model razvoja zasnovan na uslugama nalaže iznajmljivanje funkcionalnosti programskog sustava [13]. Funkcionalnosti programskog sustava iznajmljuju se korisnicima na zahtjev (engl. *on demand*), naplaćuju se prema korištenju (engl. *pay per usage*) i isporučenoj kvaliteti usluge (engl. *Quality of Service*). Poslužitelji usluga na osnovi zahtjeva korisnika određuju odgovarajući skup usluga, dogovaraju uvjete korištenja usluga i isporučuju usluge korisniku. Nakon uspješnog isporučivanja funkcionalnosti usluga korisniku, poslužitelji usluga oslobođaju zauzete usluge i sredstva [13]. Primjena modela razvoja zasnovanog na uslugama ima nekoliko značajnih prednosti za korisnike i razvijatelje programskih sustava. Korisnici imaju mogućnost primjene modela poslovanja zasnovanog na vanjskom razvoju (engl. *outsourcing*) [14]. Primjena modela poslovanja zasnovanog na vanjskom razvoju omogućava korisnicima značajno smanjenje troškova kupovine i održavanje računalne i programske potpore. Nadalje, mogućnost iznajmljivanja programskih funkcionalnosti povećava poslovnu prilagodljivost korisnika promjenjivim uvjetima poslovanja na tržištu. Iznajmljivanjem programskih rješenja putem globalne mreže Internet, korisnici ne ovise o naslijedenim programskim sustavima koji nakon promjena uvjeta u poslovanju ograničavaju njihove mogućnosti. Nadalje, korisnici ne moraju plaćati cijenu izgradnje cjelokupnog programskog sustava posebne namjene, već samo iznajmljene funkcionalnosti prema ugovorenoj cijeni usluge. Razvijatelji koji poslužuju funkcionalnosti vlastitih programskih sustava kao usluge zadržavaju nadzor nad funkcionalnostima

programskog sustava. Obzirom da razvijatelji ne isporučuju programski sustav korisniku kao gotovi proizvod već kao uslugu, razvijatelji imaju mogućnost lakšeg održavanja i unapređivanja postojećih funkcionalnosti programskog sustava. Nadalje, razvijatelji programskih sustava imaju mogućnost korištenja usluga ostalih poslužitelja usluga na tržištu u svrhu brže izgradnje novih programske sustava.

2.1.2 Usporedba modela

Za izgradnju programskih sustava preporučljivo je istovremeno koristiti modele za razvoj zasnovane na objektima, komponentama i uslugama. Pravilan odabir koji od elemenata sustava će biti ostvareni objektima, komponentama i uslugama ima značajan utjecaj na ukupnu učinkovitost rada programskog sustava. Odgovarajuće odluke moguće je donositi samo uz poznavanje osnovnih razlika između objekata, komponenata i usluga.

Objekte, komponente i usluge moguće je usporediti prema značajkama položaja, okruženja, učinkovitosti, autorstva i brojnosti [15]. Svojstvo položaja određuje kontekst izvođenja logike objekta, komponente ili usluge. Primjeri mogućih položaja su dretva, proces ili računalo. Svojstvo okruženja određuje vrstu razvojne okoline koja je korištena u svrhu izgradnje i izvođenja objekta, komponente ili usluge. Primjeri okruženja su .NET [16] i J2EE [17]. Svojstvo učinkovitosti određuje učinkovitost komunikacije između objekata, komponenata ili usluga tijekom izvođenja. Svojstvo autorstva određuje pravnu ili fizičku osobu u čijoj je nadležnosti razvoj i održavanje objekta, komponente ili usluge. Svojstvo brojnosti određuje količinu objekata, komponenata ili usluga korištenih tijekom izgradnje programske sustava.



Slika 2: Usporedba objekata, komponenata i usluga prema značajkama lokacije i okruženja

Slika 2 prikazuje usporedbu objekata, komponenata i usluga prema značajkama položaja i okruženja. Poslužiteljski i korisnički objekti uvijek se izvode u istom procesu. Nadalje, poslužiteljske i korisničke komponente u pravilu se izvode u različitim procesima, dok se poslužiteljske i korisničke usluge izvode u različitim procesima koji se izvode u različitim okruženjima. Usporedbom objekata, komponenata i usluga prema značajci okruženja moguće je zaključiti da se poslužiteljski i korisnički objekti i komponente uvijek izvode u istom okruženju. Nadalje, poslužiteljske i korisničke usluge u pravilu se izvode u različitim okruženjima.

Usporedbom objekata, komponenata i usluga prema značajci učinkovitosti moguće je zaključiti da objekti međusobno ostvaruju najučinkovitiju komunikaciju. Komunikacija između objekata najučinkovitija je obzirom da se ostvaruje unutar procesa razmjenom vrijednosti putem sustavskog stoga. Komunikacija između komponenata ostvaruje se primjenom mehanizama za međuprocesnu komunikaciju (engl. *inter-process communication mechanisms*). Iako su mehanizmi za međuprocesnu komunikaciju optimirani za učinkovitu komunikaciju unutar jednoobraznog komponentnog okruženja, oni su manje učinkoviti od komunikacije između objekata. Komunikacija između usluga najmanje je učinkovita obzirom da zahtijeva komunikaciju između raznovrsnih okruženja. Tijekom komunikacije ostvaruje se pretvorba zapisa struktura podataka koji se koriste u izvorišnom okruženju u standardom definirani zapis struktura podataka koji ne ovisi o okruženju. Nadalje, u odredišnom okruženju ostvaruje se pretvorba iz standardom definiranog zapisa podataka u

zapis podataka koji se koristi u odredišnom okruženju. Opisani postupak pretvorbe zapisa struktura podataka iznimno je složen što čini komunikaciju između usluga najmanje učinkovitom.

Uspored bom objekata, komponenata i usluga prema značajci autorstva moguće je ostvariti sljedeće zaključke. Objekti se koriste za ostvarivanje čvrsto povezanih cjelina koje ostvaruju zatvorene odsječke funkcionalnosti programskog sustava. U većini slučajeva osoba koja je ostvarila poslužiteljski objekt ostvaruje i odgovarajući korisnički objekt. Komponente se u pravilu koriste u svrhu učinkovitog raspodjeljivanja tijeka izvođenja funkcionalnosti programskog sustava na skup računala. Nadalje, u većini slučajeva ista razvojna grupa razvija poslužiteljske i korisničke komponente. Obzirom da se usluge koriste s ciljem povezivanja raznovrsnih programskih sustava, poslužiteljske i korisničke usluge u pravilu razvijaju različite tvrtke ili razvojne grupe.

Tijekom izgradnje programskih sustava u najvećoj količini koriste se objekti. Objekti se koriste u najvećoj količini obzirom da se rabe za ostvarivanje aplikacijske logike programskog sustava. Komponente se koriste u manjoj količini od objekata, obzirom da se koriste za grupiranje objekata u zasebne programske jedinice koje je moguće učinkovito raspodijeljeno izvoditi na skupu računala. Usluge se koriste u najmanjoj količini jer se u većini slučajeva rabe za grupiranje komponenata s ciljem omogućavanja javnog pristupa određenim funkcionalnostima programskog sustava.

2.2 Programske arhitekture zasnovane na uslugama

Programska arhitektura računalnog sustava opisuje funkcionalnosti i međuzavisnosti programskih elemenata sustava, te postupke njihova međusobnog grupiranja i povezivanja [18]. Programske arhitekture za sustave različitih područja primjene kao što su sustavi za prikupljanje podatka (engl. *data acquisition systems*), sustavi za spremanje podataka (engl. *database systems*) i sustavi za upravljanje (engl. *control systems*) istražuju se dugi niz godina i dobro su poznate. Sljedeći važan korak u razvoju programskog inženjerstva (engl. *software engineering*) je definiranje vršne arhitekture koja omogućava objedinjavanje postojećih raznovrsnih programskih sustava u funkcionalne cjeline koje ostvaruju nove složenije funkcionalnosti. Kako bi se ispunio navedeni zahtjev, uvedena je arhitektura zasnovana na uslugama (engl. *service-oriented architecture*) [19]. Arhitektura zasnovana na uslugama definira pravila za oblikovanje i izgradnju raspodijeljenih aplikacija zasnovanih na uslugama koje koriste krajnji korisnici ili udaljene usluge dostupne u globalnoj mreži Internet.

2.2.1 Osnovne značajke usluga

Osnovne značajke usluga su slaba povezanost (engl. *loosely coupled services*), opisi usluga (engl. *service descriptions*), krupna zrnatost (engl. *coarse grained services*), čuvanje stanja (engl. *state preservation*), kompozicija usluga (engl. *service composition*), prilagodljivost (engl. *flexibility*) i otvoreni standardi (engl. *open standards*) [20] [21].

Usluge su samostalne i međusobno ravnopravne programske jedinice koje suraduju kako bi ostvarile funkcionalnosti određene raspodijeljene aplikacije. Tijekom međusobne suradnje, usluge ostvaruju komunikaciju primjenom različitih obrazaca za razmjenu poruka (engl. *message exchange patterns*). Obrasci za razmjenu poruka definirani su primjenom ugovora između usluga (engl. *service level agreement*) [22] koji vrijede tijekom suradnje usluga. Primjena ugovora osigurava isporuku usluga prema ugovorenim uvjetima korištenja i kvaliteti usluge. Nakon što usluge završe međusobnu suradnju, ugovor postaje nevažećim i usluge oslobađaju zauzeta računalna sredstva. Usluge nastoje odabratи najbolje ugovore koji im jamče najpovoljnije uvjete korištenja udaljenih usluga. Primjena ugovora omogućava ostvarivanje slabe povezanosti između usluga vezama koje se dinamički mijenjaju za vrijeme izvođenja usluga. Značajka slabe povezanosti usluga omogućava izgradnju prilagodljivih raspodijeljenih aplikacija zasnovanih na uslugama sa svojstvima razmernog rasta i otpornosti na pogreške.

Opis usluge određuje značajke usluge bez detalja njezina programskog ostvarenja. Primjena opisa usluga ključna je za uspješno provođenje postupaka otkrivanja (engl. *discovery*), odabira (engl. *selection*), pristupa (engl. *invocation*) i kompozicije (engl. *composition*) usluga. Opisi usluga grade se primjenom formalnih jezika koji omogućavaju definiranje konzistentnih pravila za opisivanje mogućnosti usluga (engl. *service capabilities*), pristupnih sučelja usluga (engl. *service access interfaces*), ponašajnih svojstava usluga (engl. *service behavior*) i kvalitete usluga (engl. *Quality of Service*). Opisi mogućnosti usluga određuju namjenu i rezultate korištenja usluge. Opisi pristupnih sučelja definiraju strukturu ulaznih i izlaznih poruka usluge, te strukturu poruka za dojavu pogrešaka tijekom rada usluge. Opisi pristupnih sučelja opisuju operacije koje usluga izlaže putem pristupnog sučelja na korištenje udaljenim korisnicima. Primjenom opisa pristupnih sučelja usluge korisnici imaju mogućnost komunicirati s uslugom bez poznavanja detalja njezina programskog ostvarenja. Opisi ponašajnih svojstava usluge određuju akcije koje usluga provodi tijekom izvođenja. Opis kvalitete usluge određuje različite nefunkcijske značajke (engl. *non-functional attributes*) usluge kao što su cijena (engl. *cost*), učinkovitost (engl.

efficiency), dostupnost (engl. *availability*), propusnost (engl. *throughput*), vrijeme odziva (engl. *response time*) i sigurnosne postavke (engl. *security attributes*) [23].

Obzirom na način njihova ostvarenja, usluge se dijele u dvije skupine usluga: osnovne usluge (engl. *basic services*) i složene usluge (engl. *composite services*). Osnovne usluge čine usluge koje tijekom izvođenja ne koriste udaljene usluge. Složene usluge čine usluge koje su ostvarene povezivanjem dvije ili više usluga s ciljem ostvarivanja novih usluga sa složenim funkcionalnostima. Svaka usluga pomoću koje je ostvarena složena usluga ostvara segment ukupne funkcionalnosti složene usluge.

Usluge imaju krupnu zrnatost i koriste se u svrhu objedinjavanja raznovrsnih naslijedenih sustava (engl. *legacy systems*) u jedinstvene sustave koji ostvaruju nove složene funkcionalnosti. Povezivanje usluga krupne zrnatosti i upravljanje njihovim međuzavisnostima ostvara se primjenom načela programiranja na veliko (engl. *programming in the large*) [24].

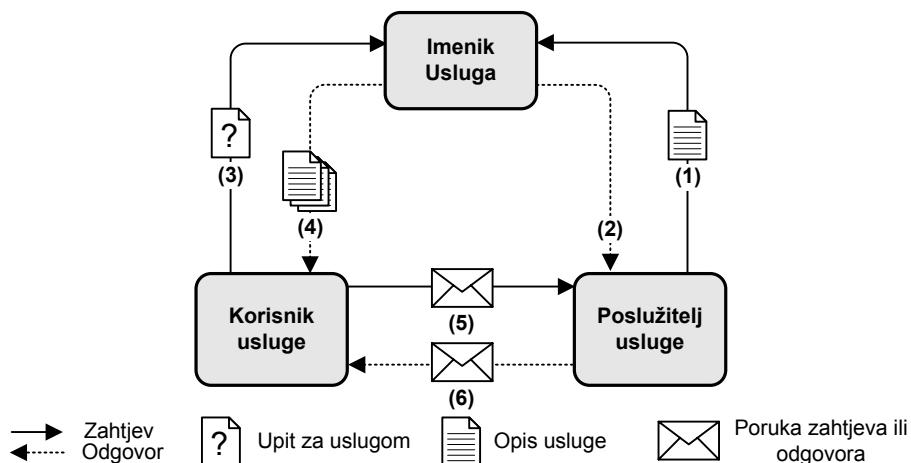
Usپoredбом prema značajci čuvanja stanja usluge se dijele na usluge bez čuvanja stanja (engl. *stateless services*) i usluge s čuvanjem stanja (engl. *stateful services*). Usluge bez čuvanja stanja ostvaruju obradu svake pristigle poruke zahtjeva neovisno o prethodno primljenim i obrađenim porukama zahtjeva. Usluge s čuvanjem stanja uključuju potporu za korištenje standardnih mehanizma za stvaranje, opisivanje, naslovljavanje, upravljanje i uništavanje sredstava koje usluge spremaju. Rezultat obrade svake poruke zahtjeva ovisi o sadržaju primljene poruke zahtjeva, redoslijedu prispijeća i sadržaju prethodno obrađenih poruka zahtjeva, te informacijama koje usluga dohvaća iz sustava za trajno spremanje podataka [20].

Usluge imaju mogućnost izmjene vlastitih radnih postavki tijekom izvođenja kako bi poboljšale učinkovitost izvođenja. Usluge mogu imati različita pristupna sučelja koja omogućavaju pristup istim funkcionalnostima, ali različitom kvalitetom usluge. Prilagodljivost usluga omogućava jednostavnije održavanje i unapređivanje raspodijeljenih aplikacija zasnovanih na uslugama.

Usluge su zasnovane na otvorenom skupu standarda koji se primjenjuju za opisivanje strukture podataka, komunikacijskih protokola, sučelja usluga, složenih usluga i ugovora između usluga. Otvoreni standardi osiguravaju neovisnost usluga o računalnim platformama, operacijskim sustavima i razvojnim okruženjima. Nadalje, primjena otvorenih standarda omogućava objedinjavanje raznovrsnih sustava u jedinstveni računalni sustav.

2.2.2 Osnovna arhitektura zasnovana na uslugama

Osnovna arhitektura zasnovana na uslugama (engl. *basic service-oriented architecture*) definira osnovne elemente programskih sustava zasnovanih na uslugama i mehanizme koji se primjenjuju tijekom njihove izgradnje i izvođenja [11]. Elementi osnovne arhitekture zasnovane na uslugama su: poslužitelj usluge (engl. *service provider*), korisnik usluge (engl. *service client*) i imenik usluga (engl. *service registry*). Poslužitelj usluge ostvaruje i nadzire pristup funkcionalnostima određene usluge. Korisnik usluge je krajnji korisnik ili udaljena usluga koja zahtijeva određenu vrstu funkcionalnosti. Imenik usluga ostvara javno dostupni imenik koji sadrži informacije o uslugama koje su dostupne u globalnoj mreži Internet.



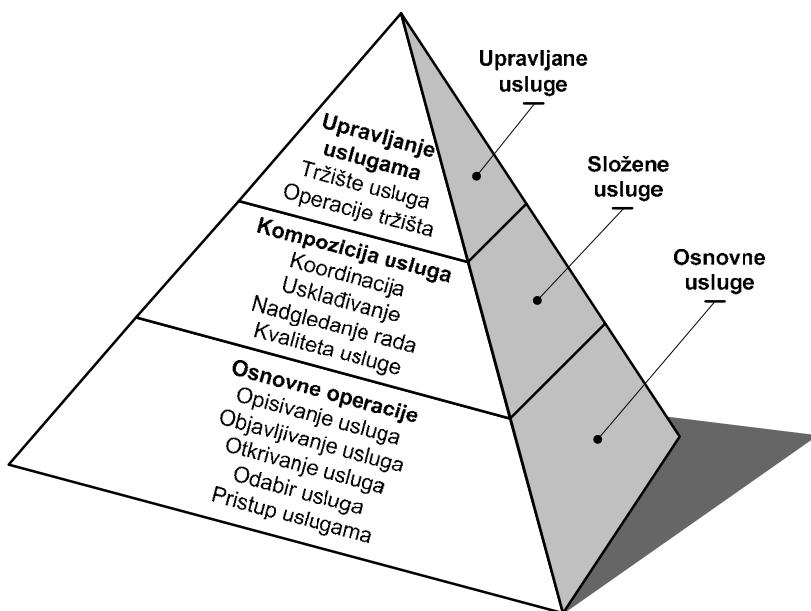
Slika 3: Osnovna arhitektura zasnovana na uslugama

Slika 3 prikazuje elemente osnovne arhitekture zasnovane na uslugama i njihove međusobne zavisnosti. Poslužitelj usluge, korisnik usluge i imenik usluge ostvaruju međudjelovanje provođenjem postupka *objavljivanje-pronalažak-pristupanje* (engl. *publish-find-bind*). Postupak *objavljivanje-pronalažak-pristupanje* započinje nakon što poslužitelj usluge ostvari funkcionalnosti i postavi uslugu. Poslužitelj usluge gradi dokument koji opisuje postavljenu uslugu i objavljuje (engl. *publish*) izgrađeni dokument u imeniku usluga (1). Imenik usluga prihvata dokument s opisom nove usluge i potvrđuje primitak (2) poslužitelju usluge. Kako bi pronašao (engl. *find*) uslugu s traženim funkcionalnostima, korisnik usluge usmjerava upit imeniku usluga (3). Upit sadrži opis funkcijskih i nefunkcijskih značajki usluge koju zahtijeva korisnik usluge. Po primitku upita, imenik usluga pretražuje skup dokumenata s opisima trenutno objavljenih usluga i vraća dokumente s opisima usluga koje ispunjavaju značajke definirane u primljenom upitu (4). Korisnik usluge odabire najprikladniju uslugu u primljenom skupu dokumenata s opisima usluga. Kako bi ostvario pristup (engl. *bind*) usluzi, korisnik usluge šalje poruku zahtjeva (5) poslužitelju usluge. Poslužitelj usluge odgovara (6) na upit korisnika usluge.

odabranoj usluzi. Odabranu udaljenu uslugu prima poruku zahtjeva, ostvaruje obradu zahtjeva i vraća poruku odgovora koji sadrži rezultate obrade (6). Postupak *objava-pronalažak-povezivanje* završava nakon što korisnik usluge primi poruku odgovora s rezultatima izvođenja usluge.

2.2.3 Proširena arhitektura zasnovana na uslugama

Proširena arhitektura zasnovana na uslugama (engl. *extended service-oriented architecture*) zasniva se na primjeni naprednih metoda računarstva zasnovanog na uslugama (engl. *service-oriented computing*), kao što su kompozicija usluga (engl. *service composition*), upravljanje uslugama (engl. *service management*), upravljanje transakcijama (engl. *service transaction management*) i sigurnost usluga (engl. *service security*) [11].



Slika 4: Proširena arhitektura zasnovana na uslugama

Slika 4 prikazuje proširenu arhitekturu zasnovanu na uslugama. Proširena arhitektura zasnovana na uslugama je slojevita arhitektura koja sadrži sloj osnovnih usluga (engl. *basic services layer*), sloj složenih usluga (engl. *composite services layer*) i sloj upravljanih usluga (engl. *managed services layer*). Sloj osnovnih usluga uključuje elemente i postupke koje definira osnovna arhitektura zasnovana na uslugama opisana u odjeljku 2.2.2.

Sloj složenih usluga uključuje postupke koji se koriste u svrhu izgradnje i izvođenja složenih usluga. Postupci koji omogućavaju izgradnju i izvođenje složenih usluga su koordinacija usluga (engl. *service coordination*), usklađivanje usluga (engl. *service*

conformance), nadgledanje usluga (engl. *service monitoring*) i kvaliteta usluge (engl. *quality of service*). Postupak koordinacije usluga primjenjuje se za definiranje pravilnog redoslijeda i vremenske usklađenosti razmjene poruka između osnovnih usluga koje čine složenu uslugu. Postupak usklađivanja usluga omogućava pretvorbu sadržaja poruka koje osnovne usluge međusobno izmjenjuju tijekom rada. Postupak nadgledanja omogućava praćenje i upravljanje tijekom izvođenja usluga s ciljem utvrđivanja nepravilnosti i neučinkovitosti u radu složenih usluga. Kvaliteta usluge nalaže postavljanje i određivanje kvalitete složene usluge upravljanjem značajkama usluge kao što su učinkovitost, dostupnost, propusnost, vrijeme odziva i sigurnosne postavke [23]. Opisane postupke primjenjuju sastavljači usluga (engl. *service aggregators*) tijekom izgradnje složenih usluga. Nakon što izgrade određenu složenu uslugu, sastavljači usluga objavljaju opis izgrađene složene usluge u imeniku usluga i postaju njezini poslužitelji. Izgrađene i objavljene složene usluge moguće je ponovno koristiti ravnopravno s osnovnim uslugama tijekom izgradnje novih složenih usluga.

Sloj upravljanja usluga uključuje postupke koji omogućavaju upravljanje uslugama u svrhu izgradnje otvorenih tržišta usluga (engl. *open service marketplaces*). Svrha otvorenih tržišta usluga je ostvarivanje preduvjeta za povezivanje korisnika i poslužitelja usluga s ciljem njihove međusobne elektroničke poslovne suradnje. Tržišta usluga grade se za pojedina područja poslovnih djelatnosti u kojima je moguće uspostaviti lanac ponude i potražnje (engl. *supply demand chain*) između korisnika i poslužitelja usluga. Preduvjet za izgradnju tržišta usluga je definiranje jedinstvene poslovne terminologije, korištenje ugovora između usluga, vrednovanje i nadziranje kvalitete usluga. Za izgradnju i upravljanje otvorenim tržištem zadužene su pravne osobne nazvane graditelji tržišta (engl. *market builders*). Graditelje tržišta čine udruženja organizacija koja ujedinjuju različite poslužitelje usluga u određenom poslovnom području. Osim graditelja tržišta, važnu ulogu imaju i operateri usluga (engl. *service operators*). Zadaća operatera usluga je održavanje, nadziranje i upravljanje okolinom za postavljanje i izvođenje usluga u svrhu postizanja učinkovitog posluživanja složenih usluga i raspodijeljenih aplikacija zasnovanih na uslugama.

2.3 Primjena računarstva zasnovanog na uslugama

Načela računarstva zasnovanog na uslugama i arhitekture zasnovane na uslugama moguće je primjeniti za izgradnju raznovrsnih raspodijeljenih računalnih sustava. Arhitekture zasnovane na uslugama uspješno se primjenjuju za oblikovanje i izgradnju sustava za upravljanje poslovnim procesima (engl. *business process management*), sustava

za izgradnju spletova računala (engl. *grid systems*), uslužnih računalnih sustava (engl. *utility computing systems*) i sustava ravnopravnih sudionika (engl. *peer-to-peer systems*).

2.3.1 Sustavi za upravljanje poslovnim procesima

Primjena informacijskih sustava omogućava poslovnim organizacijama učinkovito provođenje svakodnevnih poslovnih aktivnosti. Poslovne organizacije u pravilu koriste različite poslovne aplikacije koje su ostvarene primjenom raznovrsnih računalnih platformi, operacijskih sustava, razvojnih okolina i tehnologija. Učinkovito postavljanje, povezivanje, izvođenje, održavanje i upravljanje raznovrsnim poslovnim aplikacijama unutar jedne organizacije ostvaruje se primjenom postupaka za objedinjavanje poslovnih aplikacija (engl. *Enterprise Application Integration, EAI*) [25]. Izgradnjom globalne mreže Internet, ostvarila se mogućnost povezivanja informacijskih sustava različitih poslovnih organizacija s ciljem ostvarivanja njihove elektroničke poslovne suradnje (engl. *Business to Business Integration*) [26]. Elektronička poslovna suradnja ostvaruje se usklađivanjem i povezivanjem raznovrsnih poslovnih aplikacija različitih poslovnih organizacija putem globalne mreže Internet.

Postavljanje, izvođenje, održavanje i upravljanje poslovnim aplikacijama ostvaruje se primjenom načela upravljanja poslovnim procesima (engl. *Business Process Management, BPM*). Poslovni procesi omogućavaju povezivanje raznovrsnih poslovnih aplikacija unutar jedinstvene organizacijske domene i između različitih organizacijskih domena. Poslovni procesi ostvaruju obrasce za razmjenu i obradu poruka koje se razmjenjuju između poslovnih aplikacija u svrhu provođenja elektroničke poslovne suradnje. Postavljanje, izvođenje, upravljanje i održavanje poslovnih processa provodi se primjenom sustava za upravljanje poslovnim procesima (engl. *Business Process Management Systems*).



Razvoj strategija za objedinjavanje poslovnih aplikacija [27] prikazan je na slici 5. Slika 5a) prikazuje strategiju izravnog objedinjavanja (engl. *point-to-point*) koja se zasniva

na izravnom povezivanju svih poslovnih aplikacija putem odgovarajućih sučelja. Kako bi se ostvarilo objedinjavanje N aplikacija, potrebno je svaku aplikaciju proširiti s $N-1$ sučelja koja omogućavaju komunikaciju s preostalih $N-1$ aplikacija. Osnovni nedostatak ove strategije objedinjavanja je veliki broj sučelja i veza koje je potrebno izgraditi i održavati. U najgorem slučaju, kada se izravno objedinjuje N aplikacija, potrebno je izgraditi i održavati ukupno $N \times (N-1)$ sučelja i $N \times (N-1)/2$ veza. Dodatni nedostatak strategije izravnog objedinjavanja je potreba za proširivanjem poslovne logike svake aplikacije s logikom za međusobnu koordinaciju poslovnih aplikacija.

Slika 5b) prikazuje strategiju objedinjavanja zasnovanu na primjeni posrednika poruka (engl. *message broker*). Posrednik poruka omogućava pretvorbu sadržaja poruka i usmjeravanje poruka koje poslovne aplikacije međusobno razmjenjuju. Svaku aplikaciju potrebno je proširiti samo jednim sučeljem čime se smanjuje ukupni broj sučelja i veza koje je potrebno izgraditi i održavati. Osnovni nedostatak strategije objedinjavanja zasnovane na posredniku poruka je potreba za proširivanjem logike poslovnih aplikacija s logikom za njihovu međusobnu koordinaciju.

Slika 5c) prikazuje strategiju objedinjavanja zasnovanu na primjeni upravitelja procesa (engl. *process manager*) koji je ostvaren unapredivanjem funkcionalnosti posrednika poruka. Zadaća upravitelja procesa je izvođenje poslovnih procesa koji ostvaruju logiku za objedinjavanje aplikacija. Poslovni procesi ostvaruju obrasce za usmjeravanje poruka i pretvorbu njihova sadržaja čime se ostvaruje koordinacijska logika poslovnih aplikacija. Primjenom poslovnih procesa nije potrebno mijenjati poslovnu logiku objedinjenih aplikacija, što olakšava upravljanje objedinjenim poslovnim aplikacijama i njihovo održavanje. Postupci oblikovanja, postavljanja, analize, izvođenja i nadgledanja poslovnih procesa jedinstvenim se imenom nazivaju upravljanje poslovnim procesima.

Primjena načela računarstva zasnovanog na uslugama olakšava provođenje postupaka za objedinjavanje poslovnih aplikacija. Usluge omogućavaju izlaganje poslovne logike aplikacija putem standardnih pristupnih sučelja. Primjena standardnih pristupnih sučelja, komunikacijskih protokola i notacija za predstavljanje podataka omogućava uspostavljanje komunikacije između raznovrsnih aplikacija, te naslijedenih računalnih sustava. Objedinjavanje poslovnih aplikacija moguće je ostvariti unutar jedne organizacijske domene ili između više različitih organizacijskih domena. Logika za koordinaciju poslovnih aplikacija nije dio poslovnih aplikacija, već je izdvojena i opisana korištenjem standardnih jezika za opis poslovnih procesa. Opise poslovnih procesa ostvarene standardnim jezicima

moguće je prenositi između različitih organizacijskih domena i izvoditi ih primjenom raznovrsnih sustava za upravljanje poslovnim procesima. Objedinjavanje poslovnih aplikacija ostvareno primjenom usluga olakšava postupke postavljanja, izvođenja, nadgledanja i upravljanja poslovnim aplikacijama. Nadalje, primjena usluga osigurava prilagodljivost poslovnih organizacija s obzirom na promjenjive uvjete tržišta, te smanjuje ukupne troškove izgradnje i održavanja poslovnih informacijskih sustava [21].

2.3.2 Sustavi za izgradnju spletova računala

Dugogodišnji razvoj i istraživanja superračunala nalaze se na prekretnici. Izgradnja i održavanje superračunala tehnički je vrlo složen postupak koji zahtjeva ulaganje velikih novčanih sredstava. Nadalje, vrijeme uporabe i razdoblje isplativosti izgrađenih superračunala značajno se smanjuje. Trenutni razvoj i istraživanja usmjereni su prema zamjenskim rješenjima za izvođenje aplikacija koje zahtijevaju veliku količinu računalnih sredstava [28]. Globalna mreža Internet ostvaruje komunikacijsku infrastrukturu koja omogućava izgradnju spleta računalnih sustava (engl. *Grid*). Splet računalnih sustava je dinamičan i prilagodljiv raspodijeljeni sustav koji objedinjuje raznovrsne računalne sustave putem Interneta u svrhu izvođenja aplikacija koje zahtijevaju veliku količinu računalnih sredstava. Sustavi zasnovani na spletu računala omogućavaju sigurno, usklađeno i pouzdano dijeljenje računalnih sredstava između raznovrsnih krajnjih korisnika, organizacija i računalnih sustava [29]. Primjenom sustava zasnovanih na spletu računala moguće je izgraditi i izvoditi složene raspodijeljene znanstvene i poslovne aplikacije. Znanstvene aplikacije omogućavaju povezivanje raznovrsnih osjetila i aktuatora, provođenje složenih računskih postupaka primjenom grozdova računala (engl. *clusters*), analizu dobivenih skupova podataka i prikazivanje rezultata provedenih analiza. Poslovne aplikacije omogućavaju povezivanje, usklađivanje, nadgledanje i upravljanje sredstvima i poslovnim procesima u računalnim sustavima poslovnih organizacija u svrhu provođenja učinkovitog poslovanja [30] [31].

Primjena načela računarstva zasnovanog na uslugama omogućava uspješno ostvarivanje sustava zasnovanih na spletu računala. Usluge omogućavaju izgradnju raspodijeljenih prividnih sredstava (engl. *virtual resources*) [32] koji omogućavaju upravljanje i korištenje raznovrsnih sustava kao što su osjetila, aktuatori, sustavi za obradu i spremanje podataka. Grupiranjem i povezivanjem prividnih sredstava moguće je ostvariti složene zemljopisno raspodijeljene aplikacije zasnovane na spletu računala. Slaba povezanost usluga u sustavu spletu računala osigurava razmjeran rast sustava s obzirom na

brojnost sredstava i korisnika. Standardni i jednoobrazni (engl. *uniform*) mehanizmi za ostvarivanje sigurnosti, pouzdanosti i kvalitete usluga omogućavaju učinkovito korištenje, nadgledanje i upravljanje sustavom zasnovanim na spletu računala.

Izgradnju sustava zasnovanih na spletu računala omogućavaju različita razvojna okruženja (engl. *frameworks*). Razvojna okruženja za izgradnju sustava zasnovanih na spletu računala pružaju potporu tijekom oblikovanja, izgradnje i izvođenja aplikacija zasnovanih na spletu računala. Organizacija *Globus Alliance* [33] predvodi razvoj najpoznatijeg razvojnog okruženja za izgradnju sustava zasnovanih na spletu računala pod imenom *GT* (*Globus Toolkit*) [34]. Razvojno okruženje *GT* uključuje javno dostupne programske knjižnice s mehanizmima za ostvarivanje nadgledanja, otkrivanja, upravljanja, usklađivanja, zaštite, suradnje i komunikacije sredstava. Programski sustavi zasnovani na spletu računala grade se proširivanjem i povezivanjem komponenata *GT* razvojnog okruženja. Programske komponente *GT* razvojnog okruženja ostvarene su prema načelima *OGSA* (*Open Services Grid Architecture*) [35] [36] arhitekture. *OGSA* arhitektura zasniva se na primjeni načela arhitektura zasnovanih na uslugama i tehnologiji *Web Services* [37]. Najnovija inačica *OGSA* arhitekture uključuje usluge sa svojstvom čuvanja stanja, koje definira *WS-ResourceFramework* skup specifikacija [38].

2.3.3 Uslužni računalni sustavi

Uslužno računarstvo (engl. *utility computing*) zasniva se na načelu iznajmljivanja računalnih sredstava krajnjem korisniku prema ugovorenim uvjetima korištenja i kvaliteti usluge [39]. Najčešće se iznajmljuju računalna sredstva, kao što su aplikacije, računalno vrijeme, podaci, prostor za spremanje podataka i mrežni komunikacijski kanali. Primjena načela uslužnog računarstva omogućava organizacijama ulaganje sredstava samo u svrhu specijalizacije u određenoj poslovnoj domeni. Za sve ostale aktivnosti koje nisu dio užeg područja djelatnosti organizacije primjenjuje se načelo vanjskog razvoja [14], koje nalaže iznajmljivanje usluga dostupnih na tržištu usluga. Primjenom modela vanjskog razvoja smanjuju se ili u potpunosti uklanjaju troškovi kupovine i održavanja računalne infrastrukture i aplikacija. Zbog osnovnih načela na kojima se zasniva računarstvo zasnovano na uslugama, načela računarstva zasnovanog na uslugama podupiru i olakšavaju izgradnju sustava zasnovanih na načelu uslužnog računarstva [22].

2.3.4 Sustavi ravnopravnih sudionika

Sustavi ravnopravnih sudionika (engl. *peer-to-peer systems*) [5] omogućavaju dijeljenje računalnih sredstava kao što su računalno vrijeme, podaci, mrežni komunikacijski kanali i prostor za spremanje podataka između udaljenih računalnih sustava. Dijeljenje računalnih sredstava između dva ili više sudionika (engl. *peers*) ostvaruje se njihovom izravnom komunikacijom, bez primjene središnjeg upravljačkog sustava. Svaki sudionik istovremeno može posluživati vlastita sredstva i koristiti sredstva ostalih sudionika. Sudionici su međusobno slabo povezani i mogu samostalno napuštati ili pristupati sustavu ravnopravnih sudionika. Osnovne značajke sustava ravnopravnih sudionika su dinamičnost, otpornost na pogreške i svojstvo razmjernog rasta s obzirom na brojnost sredstava i korisnika sustava.

Primjena načela računarstva zasnovanog na uslugama omogućava izgradnju sustava ravnopravnih sudionika [40]. Sudionike sustava čine samostalne i međusobno slabo povezane usluge koje istovremeno poslužuju vlastita sredstva i koriste sredstva udaljenih sudionika dostupnih kao usluge u mreži ravnopravnih sudionika. Primjena standardnih pristupnih sučelja i komunikacijskih protokola omogućava izgradnju sustava ravnopravnih sudionika koji su ostvareni povezivanjem raznovrsnih računalnih platformi, operacijskih sustava i programskih okruženja. Svojstvo razmjernog rasta sustava ravnopravnih sudionika zasnovanog na uslugama ostvareno je izgradnjom raspodijeljenog imenika usluga. Zapisи imenika usluga raspodijeljeni su unutar sustava ravnopravnih sudionika. Svaki sudionik, osim aplikacijskih usluga koje poslužuje, sprema i dio ukupnih zapisa imenika usluga. Kako bi pronašli odgovarajuća sredstva u mreži ravnopravnih sudionika, sudionici primjenjuju različite raspodijeljene algoritme za pretraživanje imenika sredstava.

2.4 Tehnologije za ostvarivanje sustava zasnovanih na uslugama

Računalne sisteme s arhitekturama zasnovanim na uslugama moguće je graditi primjenom dvije vrste tehnologija: *Web Services* [37] i *ebXML (Electronic Business using eXtensible Markup Language)* [41]. Tehnologija *Web Services* definira skup jednostavnih i proširivih specifikacija koje čine radni okvir za izgradnju raspodijeljenih računalnih sustava zasnovanih na uslugama za proizvoljno područje primjene. Tehnologija *ebXML* standardizira minimalni skup mehanizama koji su potrebni za ostvarivanje učinkovite elektroničke poslovne suradnje putem globalne mreže Internet između dvije ili više organizacija [42].

2.4.1 Tehnologija Web Services

Tehnologija *Web Services* omogućava izgradnju raspodijeljenih aplikacija i računalnih sustava zasnovanih na uslugama u globalnoj mreži Internet [37]. Zasniva se na skupu otvorenih specifikacija *WS-** [43] koje propisuju standardne elemente i mehanizme za izgradnju sustava zasnovanih na uslugama. Osnova svih specifikacija u skupu *WS-** su jezici *XML* [44] (engl. *eXtensible Mark-up Language*) i *XML Schema* [45].

Jezik *XML* pripada širokoj obitelji jezika s oznakama (engl. *mark-up languages*) [46] i omogućava opisivanje sadržaja i strukture podataka u tekstualnom obliku koji nije ovisan o računalnoj platformi, operacijskom sustavu ili razvojnoj okolini. Svaki *XML* dokument sadrži skup elemenata s proizvoljnim atributima. Atributi pridruženi elementu služe za opisivanje sadržaja koji se nalazi u tijelu elementa. Svaki element u svojem tijelu sadrži konačan broj novih elemenata. Elementi i atributi u *XML* dokumentu jedinstveno su određeni imenom i prostorom imena (engl. *namespace*) kojem pripadaju. Obzirom da se *XML* primjenjuje u širokom području primjene, velika je vjerojatnost da se u dva različita *XML* dokumenta koriste dva ista imena. Kako bi se izbjegao sukob imena (engl. *name collision*), svakom dokumentu pridružuje se jedinstveni prostor imena.

Jezik *XML Schema* koristi se u kombinaciji s jezikom *XML* i omogućava definiranje strukture i sadržaja *XML* dokumenata. Nadalje, jezik *XML Schema* uključuje i standardni skup jednostavnih podatkovnih tipova kao što su cjelobrojni tip, broj s pomičnim zarezom i tekstualni niz koji se koriste tijekom definiranja sadržaja *XML* elemenata. Na osnovi opisa u jeziku *XML Schema* moguće je ostvariti provjeru strukture i sadržaja *XML* dokumenata. Provjera se ostvaruje primjenom jezičnog procesora koji prihvata opisanu klasu *XML* dokumenata i automatski se gradi na temelju opisa ostvarenog u jeziku *XML Schema*.

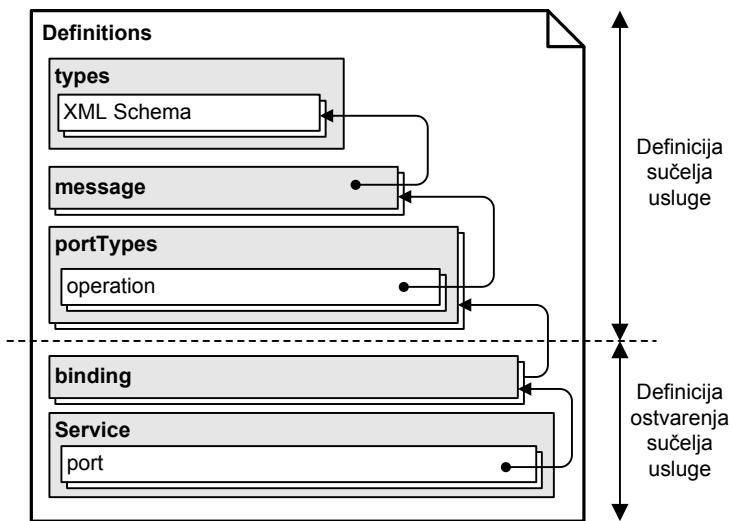
Osnovni elementi Web Services tehnologije

Osnovne i najšire prihvaćene specifikacije *WS-** skupa standarda su *SOAP* [47], *WSDL* (*Web Service Description Language*) [48] i *UDDI* (*Universal Description, Discovery, and Integration*) [49].

SOAP je otvoreni komunikacijski protokol za razmjenu poruka između raznovrsnih aplikacija putem globalne mreže Internet. Osnovne značajke protokola *SOAP* su jednostavnost, proširivost i prilagodljivost. Osnovni element *SOAP* poruke je omotnica (engl. *envelope*) koja sadrži zaglavlj (engl. *header*) i tijelo (engl. *body*) poruke. Zaglavlj *SOAP* poruke omogućava prijenos upravljačkih informacija kao što su sigurnosne postavke, jedinstvene oznake poruka, te adrese izvorišta i odredišta poruke. Tijelo *SOAP* poruke sadrži

aplikacijske podatke koji se razmjenjuju tijekom komunikacije. Protokol *SOAP* omogućava dva osnovna obrasca za razmjenu poruka: *zahtjev-odgovor* (engl. *request-response*) i jednosmjerna komunikacija (engl. *one-way messaging*). Za potrebe pakiranja i prijenosa *SOAP* poruka moguće je koristiti proizvoljan transportni komunikacijski protokol. Najčešće se primjenjuju protokoli *HTTP* (*Hyper Text Transfer Protocol*), *HTTPS* (*Hyper Text Transfer Protocol Secure*), *SMTP* (*Simple Mail Transfer Protocol*) ili *TCP* (*Transmission Control Protocol*).

WSDL je standardni jezik koji se opisuju pristupna sučelja usluga. Svaki *WSDL* dokument kojim se opisuje sučelje usluge sadrži dvije cjeline: definiciju sučelja usluga i definiciju ostvarenja sučelja usluge. Izdvajanjem definicije sučelja od definicije ostvarenja sučelja omogućava se prilagodljivost usluge. Opis sučelja neovisan je o programskom ostvarenju sučelja i moguće ga je koristiti u paru s različitim komunikacijskim protokolima. Nadalje, usluga može izlagati pristupno sučelje kojem je moguće istovremeno pristupiti korištenjem različitih transportnih komunikacijskih protokola. Definicija sučelja usluge sadrži opis tipova podataka, struktura ulaznih i izlaznih *SOAP* poruka i operacija koje sadrži pristupno sučelje usluge. Definicija ostvarenja sučelja usluge sadrži opis pravila pakiranja *SOAP* poruka, opis korištenog transportnog protokola i mrežnu adresu usluge.



Slika 6: Struktura *WSDL* dokumenta

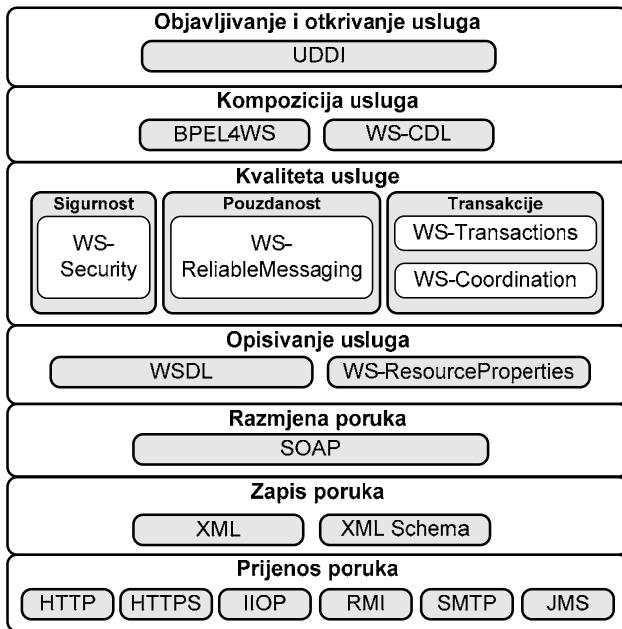
Slika 6 opisuje strukturu *WSDL* dokumenta. Definicija ostvarenja sučelja sadrži element *service* i skup elemenata *binding*. Element *service* sadrži ime usluge i mrežnu adresu na kojoj je usluga dostupna. Element *binding* služi za opisivanje komunikacijskih parametara koje je potrebno koristiti kako bi se ostvario pristup sučelju usluge. Komunikacijski

parametri sadrže vrstu transportnog protokola, primjerice *TCP*, *HTTP* ili *SMTP*. Nadalje, komunikacijski parametri sadrže i postavke koje definiraju način pakiranja *SOAP* poruka za odabrani transportni protokol. Definicija sučelja usluge sadrži skup elemenata *portTypes*, skup elemenata *message* i element *types*. Element *portTypes* sadrži listu operacija pristupnog sučelja usluge i kazaljke na opise *SOAP* poruka koje operacija koristi. Usluga može istovremeno imati više pristupnih sučelja. Element *message* sadrži kazaljke na opise *XML* dokumenata koje pojedina poruka spremu u svojem tijelu. Element *types* sadrži skup *XML Schema* dokumenata koji služe za opisivanje struktura i sadržaja *XML* dokumenata koje usluga razmjenjuje tijekom komunikacije s korisnicima.

UDDI specifikacija definira arhitekturu imeničkog sustava i standardne mehanizme za objavlјivanje, pretraživanje i odabir usluga prijavljenih u imeniku usluga. Imenik usluga sadrži informacije o uslugama spremljene u obliku bijelih stranica (engl. *white pages*), žutih stranica (engl. *yellow pages*) i zelenih stranica (engl. *green pages*). Bijele stranice sadrže kontakt informacije za poslovne organizacije koje poslužuju usluge prijavljene u imeniku usluga. Žute stranice ostvaruju razredbu usluga prema vrsti poslovanja koje je moguće ostvariti njihovom primjenom. Zelene stranice sadrže tehničke informacije koje su potrebne za ostvarivanje pristupa i korištenje objavljenih usluga.

Prošireni skup specifikacija WS-*

Primjenom *SOAP*, *WSDL* i *UDDI* specifikacija moguće je izgraditi osnovne elemente sustava zasnovanih na uslugama. Prošireni skup specifikacija *WS-** sadrži dodatne specifikacije koje definiraju naprednije mehanizme za izgradnju sustava zasnovanih na uslugama kao što su pouzdanost, sigurnost, transakcije i poslovni procesi. Neke od specifikacija *WS-** skupa još nisu dovršene i njihova standardizacija je u tijeku. Najznačajnija tijela za standardizaciju *WS-** skupa specifikacija su *W3C (World Wide Web Consortium)* [50], *OASIS (Organization for the Advancement of Structured Information Standards)* [51] i *WS-I (Web Services Interoperability)* [52].

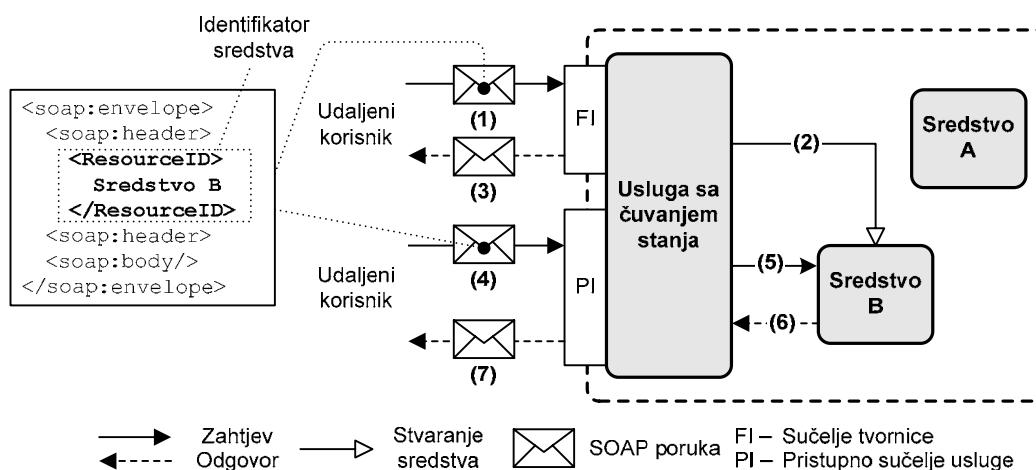


Slika 7: Prošireni stog *WS-** skupa specifikacija

Slika 7 prikazuje prošireni stog *WS-** skupa specifikacija [43]. Prikazani stog sadrži slojeve koji definiraju mehanizme za prijenos poruka, zapis poruka, razmjenu poruka, opisivanje usluga, osiguravanje kvalitete usluge, kompoziciju usluga, te objavljivanje i otkrivanje usluga. Sloj za prijenos poruka uključuje osnovne prijenosne komunikacijske protokole koji omogućavaju prijenos podataka između usluga dostupnih u globalnoj mreži Internet. Obzirom da *Web Services* tehnologija ne propisuje vrstu prijenosnog komunikacijskog protokola, za komunikaciju je moguće koristiti neki od standardnih prijenosnih protokola kao što su *HTTP*, *HTTPS*, ili *SMTP*. Sloj za zapis poruka služi za opisivanje podataka i struktura poruka primjenom jezika *XML* i *XML Schema*. Sloj za razmjenu poruka uključuje komunikacijske mehanizme za razmjenu *SOAP* poruka. Sloj za opis usluga uključuje jezik *WSDL* koji se koristi za ostvarivanje opisa sučelja usluga. Dodano, sloj uključuje i novu specifikaciju *WS-ResourceProperties* [53] koja omogućava opisivanje značajki usluga s čuvanjem stanja. Sloj za osiguravanje kvalitete usluge sadrži specifikacije koje definiraju mehanizme za ostvarivanje sigurnosti, pouzdanosti, transakcija i koordinacije usluge. Sigurnost usluga moguće je ostvariti korištenjem radnog okvira *WS-Security* [54] koji propisuje načine razmjene i primjene upravljačkih informacija za ostvarivanje sigurnosnih mehanizama. Pouzdana komunikacija razmjenom *SOAP* poruka ostvaruje se primjenom specifikacije *WS-ReliableMessaging* [55]. Specifikacije *WS-Transactions* i *WS-Coordination* [56] definiraju načine korištenja transakcijskih i koordinacijskih mehanizama u sustavima zasnovanim na *Web Services* tehnologiji. Sloj za kompoziciju usluga sadrži skup specifikacija koje definiraju mehanizme za ostvarivanje

složenih usluga. Specifikacije *BPEL4WS* [57] (*Business Process Execution Language for Web Services*) i *WS-CDL* [58] (*Web Service Choreography Description Language*) definiraju jezike za opisivanje složenih usluga. Sloj za objavljivanje i otkrivanje usluga sadrži specifikaciju *UDDI* koja definira arhitekturu imenika usluga.

Usluge ostvarene primjenom *Web Services* tehnologije prvenstveno služe za ostvarivanje pristupa raznovrsnim računalnim sustavima, kao što su sustavi za dohvaćanje, obradu i spremanje podataka. Navedeni sustavi u pravilu spremaju različita sredstva čije se stanje mijenja tijekom izvođenja i međudjelovanja sustava s udaljenim korisnicima. Kako bi se omogućio standardni način stvaranja, naslovljanja, korištenja i uništavanja različitih sredstava koje sustavi spremaju, definiran je radni okvir *WS-ResourceFramework* [61]. Radni okvir *WS-ResourceFramework* sadrži skup specifikacija: *WS-ResourceLifetime* [62], *WS-ResourceProperties* [53], *WS-ServiceGroup* [63] i *WS-BaseFaults* [64]. Specifikacija *WS-ResourceLifetime* definira mehanizme za upravljanje životnim vijekom sredstava, specifikacija *WS-ResourceProperties* definira mehanizme za opisivanje svojstava sredstava, specifikacija *WS-ServiceGroup* definira mehanizme za grupiranje sredstava, dok specifikacija *WS-BaseFaults* definira mehanizme za prijavu pogrešaka tijekom rada sredstava. Opisi ostvareni primjenom *WS-ResourceFramework* skupa specifikacija ugrađuju se kao proširenja u *WSDL* opis usluge s čuvanjem stanja.



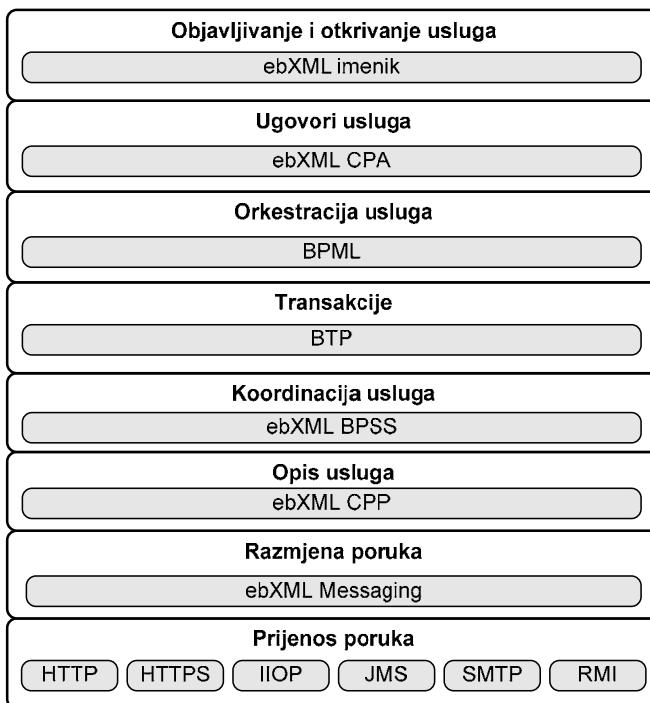
Slika 8: Primjer usluge s čuvanjem stanja

Slika 8 prikazuje jednostavan primjer stvaranja i korištenja sredstva usluge s čuvanjem stanja ostvarene primjenom radnog okvira *WS-ResourceFramework*. Usluga sadrži dva sučelja: sučelje tvornice (engl. *factory interface*) i pristupno sučelje (engl. *access interface*). Sučelje tvornice služi za stvaranje sredstava, dok pristupno sučelje služi za pristup sredstvima usluge s čuvanjem stanja. Udaljeni korisnik putem sučelja tvornice zahtjeva

stvaranje sredstva B (1). *SOAP* poruka zahtjeva u zaglavlju sadrži jedinstvenu oznaku stvorenog sredstva koja služi za njegovo naslovljavanje. Identifikator sredstva može biti zadan u poruci zahtjeva ili ga usluga samostalno pridjeljuje nakon stvaranja sredstva. Nakon primjeka *SOAP* poruke zahtjeva, usluga stvara sredstvo B (2) i prosljeđuje *SOAP* poruku odgovora s pozitivnom potvrdom udaljenom korisniku (3). Nakon uspješnog stvaranja sredstva B , udaljeni korisnik ima mogućnost uporabe stvorenog sredstva. Udaljeni korisnik prosljeđuje *SOAP* poruku zahtjeva stvorenom sredstvu putem pristupnog sučelja (4). *SOAP* poruka zahtjeva u zaglavlju sadrži jedinstvenu oznaku sredstva B . Usluga prihvata *SOAP* poruku zahtjeva i prosljeđuje primljene podatke sredstvu B (5). Sredstvo B obrađuje primljene podatke i prosljeđuje rezultat usluzi (6). Usluga prima rezultate obrade i pakira ih u *SOAP* poruku odgovora, koju prosljeđuje udaljenom korisniku (7).

2.4.2 Electronic Business using eXtensible Markup Language

Za razliku od općenite *Web Services* tehnologije, *ebXML* [41] tehnologija definira minimalni skup mehanizama koji su potrebni za ostvarivanje elektroničke poslovne suradnje između različitih organizacija putem globalne mreže Internet. Elementi i mehanizmi *ebXML* tehnologije omogućavaju učinkovito usklajivanje i razmjenu podataka između raznovrsnih programskih sustava različitih poslovnih organizacija. Nadalje, *ebXML* omogućava oblikovanje, analizu, izvođenje i nadgledanje poslovnih procesa koji se primjenjuju u svrhu provođenja elektroničke poslovne suradnje. Tehnologija *ebXML* ostvarena je pod pokroviteljstvom organizacija *UN/CEFACT (United Nations Centre for Trade Facilitation and Electronic Business)* [65] i *OASIS* [51].



Slika 9: Stog protokola *ebXML* arhitekture

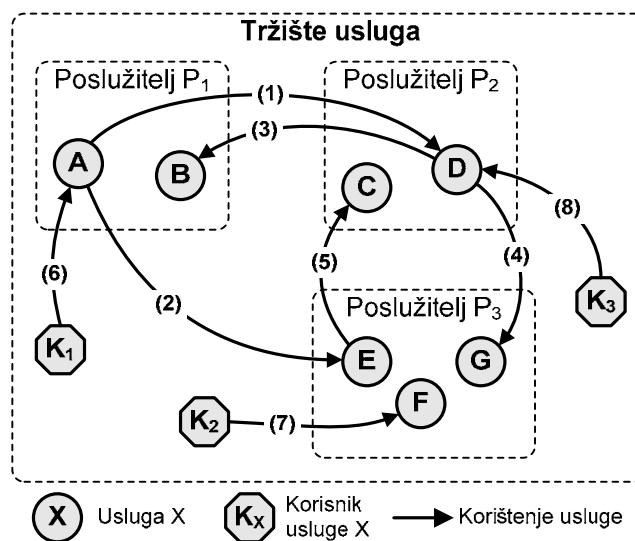
Slika 9 prikazuje stog protokola *ebXML* arhitekture [41]. Stog sadrži slojeve za prijenos poruka, razmjenu poruka, opis usluga, koordinaciju usluga, transakcije, orkestraciju usluga, ugovore usluga, te objavljivanje i otkrivanje usluga. Sloj za prijenos poruka sadrži komunikacijske protokole koji omogućavaju prijenos poruka putem globalne mreže Internet. Kao i *Web Services* tehnologije, *ebXML* ne propisuje vrstu komunikacijskog protokola. Moguće je koristiti proizvoljni prijenosni komunikacijski protokol, kao što su protokoli *HTTP*, *HTTPS*, *IIOP*, *JMS*, *SMTP* i *RMI*. Sloj za razmjenu poruka sadrži protokol *SOAP* koji je proširen mehanizmima za ostvarivanje sigurne i pouzdane komunikacije. Sloj za opis usluga sadrži specifikaciju *CPP* (*Collaboration Protocol Profile*) [66] koja omogućava opisivanje pristupnih sučelja, komunikacijskih protokola i sigurnosnih postavki usluge. Sloj za koordinaciju usluga sadrži specifikaciju *BPSS* (*Business Process Specification Schema*) [67] koja omogućava opisivanje slijeda razmjene poruka tijekom izvođenja poslovnih procesa za ostvarivanje električke poslovne suradnje. Sloj za ostvarivanje transakcija sadrži specifikaciju *BTP* (*Business Transaction Protocol*) [68] koja omogućava korištenje transakcijskih mehanizama između sudionika poslovne suradnje. Sloj za orkestraciju usluga sadrži specifikaciju *BPML* (*Business Process Modeling Language*) [69] koja omogućava oblikovanje, analizu i izgradnju poslovnih procesa za ostvarivanje poslovne suradnje između različitih poslovnih organizacija. Sloj za izgradnju ugovora usluga uključuje specifikaciju *CPA* (*Collaboration Partner Agreement*) koja služi za izgradnju ugovora između usluga (engl. *service level agreements*) različitih organizacija tijekom provođenja električke

poslovne suradnje. Sloj za objavljivanje i otkrivanje usluga najviši je sloj arhitekture *ebXML* koji omogućava objavljivanje i otkrivanje usluga različitih poslovnih organizacija. Središnja komponenta sloja je *ebXML* imenik koji sprema informacije potrebne za pronalaženje, odabir i pristup poslovnim uslugama.

3. poglavlje

Kompozicija usluga

Preduvjet za globalnu primjenu načela računarstva zasnovanog na uslugama je postojanje otvorenog tržišta usluga (engl. *open service marketplace*) [13]. Otvoreno tržište usluga omogućava povezivanje poslužitelja usluga i korisnika usluga s ciljem ostvarenja elektroničkog poslovanja. Poslužitelji ostvaruju, nadziru pristup i oglašavaju usluge različitih funkcijskih i nefunkcijskih značajki. Korisnici usluga pretražuju, odabiru i koriste usluge dostupne na tržištu. Korisničke zahtjeve u pravilu nije moguće ispuniti izravnom primjenom neke od postojećih usluga dostupnih na tržištu. Tražene funkcionalnosti korisnici u većini slučajeva pokušavaju ostvariti grupiranjem nekoliko usluga koje su dostupne na tržištu. Nadalje, poslužitelji usluga također imaju mogućnost grupiranja usluga dostupnih na tržištu u svrhu izgradnje novih usluga s naprednim funkcionalnostima. Usluge ostvarene grupiranjem nekoliko usluga s ciljem ostvarivanja složenih funkcionalnosti nazivaju se složene usluge (engl. *composite services*). Postupak pretraživanja, odabira, povezivanja i uskladivanja tijeka izvođenja skupine usluga s ciljem izgradnje složenih usluga naziva se kompozicija usluga (engl. *service composition*).



Slika 10: Primjer otvorenog tržišta usluga

Slika 10 prikazuje jednostavan primjer otvorenog tržišta usluga. Prikazano tržište usluga uključuje tri poslužitelja usluga (P_1, P_2, P_3) i tri korisnika usluga (K_1, K_2, K_3). Poslužitelj usluga P_1 poslužuje usluge A i B , poslužitelj usluga P_2 poslužuje usluge C i D , dok poslužitelj usluga P_3 poslužuje usluge E, F i G . Usluge A, D i E su složene usluge.

Složena usluga A ostvarena je uporabom usluga D (1) i E (2). Složena usluga D ostvarena je uporabom usluga B (3) i G (4). Složena usluga E djelomično ostvaruje lokalnu obradu zahtjeva i dodatno koristi uslugu C (5). Usluge B , C , G , i F su osnovne usluge. Korisnik K_1 koristi složenu uslugu A (6), korisnik K_2 koristi osnovnu uslugu F (7), dok korisnik K_3 koristi složenu uslugu D (8).

3.1 Zahtjevi kompozicije usluga

Postoje četiri skupine zahtjeva koje je potrebno ispuniti kako bi se ostvarila učinkovita kompozicija usluga [70]: povezanost (engl. *connectivity*), kvaliteta usluge (engl. *Quality of Service*), ispravnost (engl. *correctness*) i svojstvo razmjernog rasta (engl. *scalability*).

Zahtjev povezanosti nalaže učinkovitu, pouzdanu i usklađenu komunikaciju usluga pomoću kojih se ostvaruje složena usluga. Učinkovitost komunikacije ostvaruje se primjenom komunikacijskih protokola koji omogućavaju primjenu sinkronih (engl. *synchronous*) i asinkronih (engl. *asynchronous*) obrazaca za razmjenu poruka tijekom komunikacije. Sinkroni obrazac komunikacije primjenjuje se za ostvarivanje poziva udaljenih procedura (engl. *remote procedure call*) [71]. Asinkroni obrazac komunikacije primjenjuje se u slučajevima kada je potrebno osigurati slabu povezanost (engl. *loose coupling*) između usluga i korisnika usluga. Pouzdanost komunikacije postiže se primjenom mehanizama za ostvarivanje neosjetljivosti (engl. *tolerance*) i otpornosti (engl. *resilience*) na pogreške tijekom razmjene poruka. Usklađenost komunikacije ostvaruje se primjenom standardnih komunikacijskih protokola i zapisa strukture poruka koje usluge međusobno razmjenjuju tijekom komunikacije. Ostvarivanjem usklađene komunikacije usluga osigurava se cjelovitost složene usluge tijekom izvođenja.

Zahtjev za kvalitetom usluge nalaže ostvarivanje složenih usluga s odgovarajućim nefunkcijskim značajkama. Kvaliteta usluga najčešće se izražava značajkama [23] [72] dostupnosti (engl. *availability*), pouzdanosti (engl. *dependability*), sigurnosti (engl. *security*), vremena odziva (engl. *response time*) i propusnosti (engl. *throughput*) usluga. Značajka dostupnosti usluge određena je vjerojatnošću da će složena usluga biti dostupna za korištenje u bilo kojem trenutku njezina životnog vijeka. Značajka pouzdanosti određena je svojstvom složene usluge da bez pogreške poslužuje korisnike tijekom svojeg životnog vijeka. Složene usluge sa značajkama dostupnosti i pouzdanosti prilagodljive su i otporne na pogreške. Kako bi se ostvarila svojstva prilagodljivosti i otpornosti na pogreške, složene usluge tijekom

izvođenja primjenjuju mehanizme za pravovremeno nadomještanje nedostupnih usluga ili usluga s pogreškama u radu. Značajka sigurnosti nalaže primjenu mehanizama za ostvarivanje sigurnosti kao što su mehanizmi za ostvarivanje autentikacije (engl. *authentication*), povjerljivosti (engl. *confidentiality*), vjerodostojnosti (engl. *integrity*) i neporicljivosti (engl. *nonrepudiation*). Značajka vremena odziva određuje vrijeme koje je potrebno kako bi složena usluga obradila pristiglu poruku zahtjeva i dostavila odgovarajuću poruku odgovora. Značajka propusnosti određuje brzinu kojom složena usluga obrađuje prispevke poruke zahtjeva i dostavlja poruke odgovora.

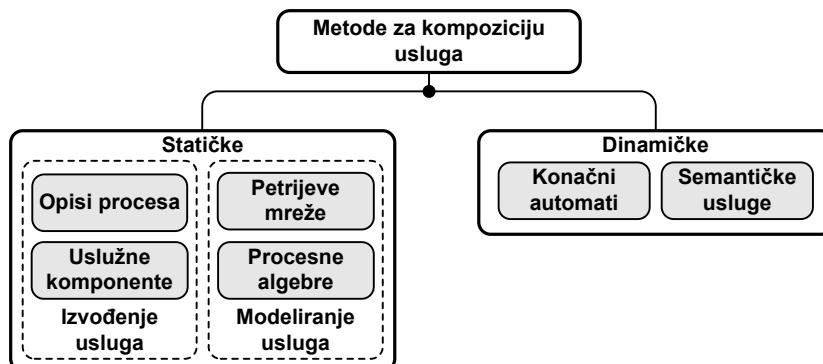
Zahtjev za ispravnošću usluge nalaže ostvarivanje složene usluge koja ispunjava funkcijeske zahtjeve korisnika usluge. Provjera značajki ispravnosti ostvaruje se primjenom različitih formalnih metoda koje omogućavaju izgradnju modela složenih usluga. Model složene usluge opisuje složenu uslugu i omogućava analizu njezinih ponašajnih svojstava. Najčešće se primjenom modela složene usluge ispituju sljedeća svojstva [73]: neškodljivost (engl. *safety*), dosežljivost (engl. *reachability*), zaustavljanje (engl. *termination*), potpuni zastoj (engl. *deadlock*) i ograničenost (engl. *boundedness*). Svojstvo neškodljivosti nalaže ispravan redoslijed izvršavanja akcija složene usluge prema funkcijskim zahtjevima složene usluge. Tijekom izvođenja složena usluga ne izvršava akcije koje narušavaju cijelovitost složene usluge ili usluga koje ona koristi. Ispitivanjem svojstva dosežljivosti moguće je utvrditi koje se usluge izvode tijekom izvođenja složene usluge. Svojstvo zaustavljanja nalaže pravilno i pravovremeno završavanje tijeka izvođenja složene usluge za sve moguće ulazne parametre. Analizom potpunog zastaja moguće je utvrditi da li će izgrađena složena usluga uspješno završiti s izvođenjem za zadane ulazne parametre. U slučaju potpunog zastaja, složena usluga nije u mogućnosti pravilno završiti izvođenje obzirom da neke usluge zahtijevaju pristup trajno zaključanim dijeljenim sredstvima. Analizom svojstva ograničenosti moguće je odrediti količinu sredstava koje složena usluga koristi tijekom izvođenja.

Zahtjev za ostvarivanje razmjernog rasta nalaže izgradnju složenih usluga koje ostvaruju učinkovito izvođenje i posluživanje korisnika unatoč značajnom porastu broja korisnika i usluga koje koristi složena usluga. Svojstvo razmjernog rasta složene usluge ostvaruje se raspodjeljivanjem tijeka izvođenja složene usluge, te primjenom metoda kasnog odabira usluga (engl. *late-binding*) ili odabira usluga tijekom izvođenja (engl. *runtime-binding*). Primjenom metoda kasnog odabira, usluge pomoću kojih se ostvaruje složena usluga odabiru se prije izvođenja složene usluge. Primjenom metoda odabira tijekom izvođenja, usluge pomoću kojih se ostvaruje složena usluga odabiru se tijekom izvođenja

složene usluge, neposredno prije njihova korištenja. Primjena opisanih metoda omogućava raspoređivanje opterećenja tijekom izvođenja složenih usluga, što ima pozitivan učinak na značajku razmjernog rasta složenih usluga.

3.2 Metode za kompoziciju usluga

Metode za kompoziciju usluga omogućavaju oblikovanje, analizu i izvođenje složenih usluga. Područje kompozicije usluga trenutno se aktivno istražuje i ne postoji sveobuhvatna metoda koja istovremeno ispunjava sve zahtjeve za kompoziciju usluga navedene u odjeljku 3.1. Slika 11 prikazuje razredbu metoda za kompoziciju usluga koje se dijele na statičke metode za kompoziciju usluga (engl. *static service composition methods*) i dinamičke metode za kompoziciju usluga (engl. *dynamic service composition methods*).



Slika 11: Razredba metoda kompozicije usluga

Statičke metode za kompoziciju usluga dijele se na metode za izvođenje složenih usluga i metode za modeliranje složenih usluga. Metode za izvođenje složenih usluga omogućavaju izgradnju opisa složenih usluga koje je moguće izvoditi primjenom sustava za kompoziciju usluga. Opisi složenih usluga sadrže definicije strukture poruka, pravila za pretvorbu sadržaja poruka, obrasce za razmjenu poruka između usluga i dodatne komunikacijske parametre. Odabir usluga pomoću kojih se ostvaruje složena usluga u pravilu se ostvaruje za vrijeme razvoja (engl. *design time*) složene usluge. Usluge koje čine složenu uslugu izravno se povezuju prema zahtjevima tijeka izvođenja složene usluge. Svaka usluga koja se koristi tijekom izgradnje složene usluge jedinstveno je određena adresom i opisom pristupnog sučelja. Najpoznatije metode za izvođenje složenih usluga su metode zasnovane na opisima procesa (engl. *process descriptions*) ili uslužnim komponentama (engl. *service components*).

Statičke metode za modeliranje složenih usluga zasnovaju se na primjeni formalnih jezika koji omogućavaju izgradnju modela složenih usluga. Modeli složenih usluga opisuju tijekove izvođenja složenih usluga i sadrže opise međuzavisnosti u izvođenju usluga koje čine složenu uslugu. Analizom tijeka izvođenja moguće je ispitivati različita ponašajna svojstva složenih usluga kao što su neškodljivost, dosežljivost, zaustavljanje, potpuni zastoj i ograničenost. Osnovni nedostaci primjene modela u svrhu analize ponašajnih svojstava složenih usluga su nemogućnost izgradnje sveobuhvatnih modela i složenost postupaka njihove analize. Modeli složenih usluga nisu sveobuhvatni i ne mogu sadržavati opise svih svojstava složenih usluga. U slučaju da model složene usluge nije dovoljno precizan ili sadrži pogreške, rezultati postupka analize ponašajnih svojstava neće biti točni. Postupci analize ponašajnih svojstava modela složenih usluga vrlo su složeni i njihovo provođenje zahtijeva značajna računalna sredstva. Ponekad su postupci analize i sami modeli složenih usluga toliko složeni da tražena ponašajna svojstva složene usluge nije moguće odrediti u konačnom vremenu. Modeli složenih usluga najčešće se ostvaruju primjenom formalizama zasnovanih na Petrijevim mrežama (engl. *Petri-nets*) i procesnim algebrama (engl. *process algebras*).

Metode za dinamičku kompoziciju usluga omogućavaju automatsku izgradnju složenih usluga [74]. Automatska kompozicija ostvaruje se primjenom opisa funkcijskih i nefunkcijskih značajki složenih usluga izgrađenih primjenom formalnih deklarativnih jezika. Tijekom izgradnje složenih usluga primjenjuju se opisi koji definiraju značajke usluge koju zahtijeva korisnik i opisi značajki usluga koje su dostupne na globalnom tržištu usluga. Primjenom korisničkog opisa tražene usluge i opisa usluga dostupnih na tržištu, sustav za automatsku kompoziciju usluga odabire najprikladniju uslugu dostupnu na tržištu koja ispunjava korisnički zahtjev. U slučaju da na tržištu nije dostupna usluga s traženim značajkama, sustav za automatsku kompoziciju usluga pokušava odrediti skup usluga dostupnih na tržištu koje je moguće povezati u složenu uslugu koja ispunjava korisnički zahtjev. Povezivanje usluga koje čine složenu uslugu moguće je provoditi tijekom oblikovanja (engl. *design-time binding*), neposredno prije izvođenja (engl. *late binding*) ili tijekom izvođenja (engl. *run-time binding*) složene usluge. Opisi složenih usluga najčešće se ostvaruju primjenom opisa zasnovanih na konačnim automatima (engl. *finite state machines*) i semantičkim uslugama (engl. *semantic services*).

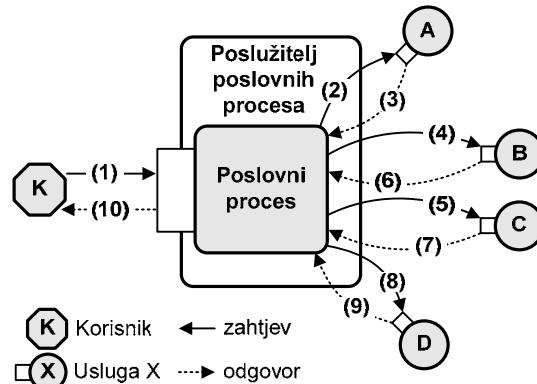
3.2.1 Kompozicija usluga zasnovana na opisima procesa

Metoda kompozicije usluga zasnovana na opisima procesa ostvaruje se izgradnjom opisa procesa koje je moguće izvoditi uporabom sustava za izvođenje složenih usluga. Opisi procesa sadrže definicije strukture poruka, pravila za pretvorbu sadržaja poruka i obrasce za razmjenu poruka između usluga koje čine složenu uslugu. Kompoziciju usluga zasnovanu na opisima procesa moguće je ostvariti primjenom metoda orkestracije ili koreografije usluga [75].

Orkestracija usluga

Orkestracija usluga (engl. *service orchestration*) je metoda za izgradnju složenih usluga koja je zasnovana na primjeni poslovnih procesa. Poslovni procesi ostvaruju obrasce za razmjenu i transformaciju poruka između usluga koje čine složenu uslugu. Obrasci za razmjenu poruka definiraju redoslijed razmjene poruka između usluga s ciljem njihove koordinacije prema zahtjevima tijeka izvođenja složene usluge. Postupci za pretvorbu poruka omogućavaju usklađivanje sadržaja izlaznih i ulaznih poruka između usluga koje čine složenu uslugu.

Metoda orkestracije složene usluge zahtijeva izgradnju opisa pristupnog sučelja poslovnog procesa i opisa logike poslovnog procesa, te korištenje opisa sučelja usluga koje poslovni proces koristi tijekom izvođenja. Opis pristupnog sučelja poslovnog procesa sadrži definicije struktura poruka i operacija izloženih putem pristupnog sučelja poslovnog procesa. Udaljeni korisnici koriste operacije pristupnog sučelja poslovnog procesa kako bi ostvarili komunikaciju s poslovnim procesom. Opis logike poslovnog procesa ostvaruje se primjenom skupa naredbi jezika za opis procesa. Tipične naredbe za izgradnju opis logike procesa su naredbe za pozivanje vanjskih usluga, pretvorbu sadržaja poruka i obradu zahtjeva pristiglih putem pristupnog sučelja poslovnog procesa. Opis pristupnih sučelja usluga koje koristi poslovni proces uključuje definicije struktura poruka i operacija pristupnih sučelja udaljenih usluga koje koristi poslovni proces tijekom izvođenja.



Slika 12: Primjer složene usluge ostvarene primjenom orkestracije usluga

Slika 11 prikazuje primjer složene usluge ostvarene primjenom orkestracije usluga. Složena usluga ostvarena je primjenom poslovnog procesa koji povezuje usluge *A*, *B*, *C* i *D*. Po primitku zahtjeva upućenog od korisnika (1), poslovni proces prosljeđuje primljeni zahtjev usluzi *A* (2). Usluga *A* obrađuje primljeni zahtjev i vraća poruku odgovora s rezultatima obrade (3). Proces orkestracije istovremeno prosljeđuje po jednu poruku s primljenim rezultatima obrade uslugama *B* i *C* (4, 5) koje približno istovremeno započinju obradu. Nakon primitka rezultata obrade od usluga *B* i *C* (6, 7), poslovni proces objedinjuje primljene rezultate i prosljeđuje ih u poruci zahtjeva usluzi *D* (8). Nakon završetka obrade, usluga *D* vraća poruku odgovora s rezultatima obrade poslovnom procesu (9) koji prosljeđuje primljenu poruku odgovora korisniku (10).

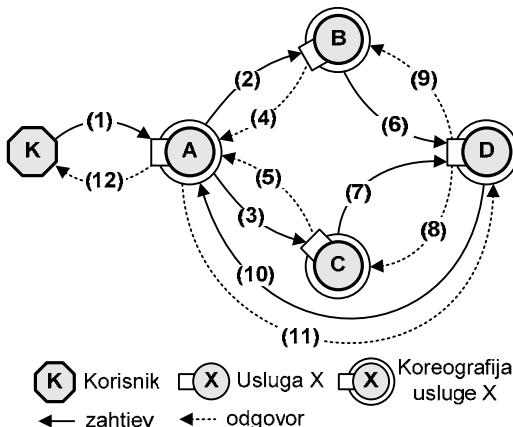
Osnovne prednosti primjene orkestracije usluga su jednostavan nadzor izvođenja složene usluge, mogućnost primjene sinkronih i asinkronih obrazaca komunikacije s vanjskim uslugama i mogućnost višestruke kompozicije usluga. Poslovni procesi koji ostvaruju složene usluge izvode se primjenom središnjih poslužitelja poslovnih procesa koji omogućavaju jednostavan nadzor izvođenja složene usluge. Središnje upravljanje složenim uslugama primjenom poslužitelja olakšava održavanje i praćenje rada složene usluge. Zbog primjene središnjeg upravljanja, metoda orkestracije usluga najčešće se primjenjuje u svrhu kompozicije usluga koje se nalaze u istoj organizacijskoj domeni. Održavanje složene usluge je olakšano obzirom da se izgrađena složena usluga nalazi u istoj organizacijskoj domeni kao i usluge koje se koriste za kompoziciju. Kako bi se ostvarila učinkovita komunikacija s uslugama koje koristi poslovni proces, koriste se sinkroni i asinkroni obrasci komunikacije. Obzirom da svaki poslovni proces sadrži pristupno sučelje, složene usluge ostvarene poslovnim procesima mogu se ponovno iskoristiti kao usluge koje čine nove složene usluge u postupku višestruke kompozicije usluga.

Osnovni nedostatak primjene metode orkestracije je korištenje središnjeg poslužitelja u svrhu izvođenja poslovnih procesa. Primjenom središnjeg poslužitelja nije moguće ostvariti razmjerni rast složene usluge s obzirom na brojnost korisnika i usluga koje koristi složena usluga. Poslužitelj poslovnih procesa je kritična točka zagušenja (engl. *bottleneck*) tijekom rada složene usluge. Porast broj korisnika složene usluge i broja usluga koje koristi složena usluga tijekom izvođenja negativno utječe na učinkovitost izvođenja složene usluge. Nadalje, poslužitelj poslovnih procesa također predstavlja kritičnu točku slabosti (engl. *single point of failure*) složene usluge. U slučaju prestanka rada poslužitelja poslovnih procesa, sve složene usluge koje poslužitelj izvodi više nisu dostupne, a sve aktivne transakcije i stanja složenih usluga u izvođenju izgubljene su.

Koreografija usluga

Koreografija usluga (engl. *service choreography*) je metoda za izgradnju složenih usluga primjenom opisa uloge svake od usluga koja čini složenu uslugu. Opis koreografije definira obrasce za međusobnu razmjenu poruka i prilagodbu sadržaja poruka koje usluge razmjenjuju tijekom izvođenja složene usluge. Obrasci za razmjenu poruka definiraju redoslijed razmjene poruka između usluga s ciljem izvođenja raspodijeljenog tijeka izvođenja prema zahtjevima složene usluge. Postupci za prilagodbu poruka omogućavaju usklađivanje sadržaja poruka koje usluge međusobno razmjenjuju tijekom izvođenja složene usluge.

Opis koreografije usluga ostvaruje se kao proširenje opisa pristupnog sučelja svake usluge koja čini složenu uslugu. Svakoj od operacija pristupnog sučelja pridružuje se skup naredbi za ostvarivanje koreografije. Naredbe pridružene određenoj operaciji usluge izvode se tijekom poziva operacije usluge. Naredbe za koreografiju usluga omogućavaju poziv udaljenih usluga, poziv lokalnih operacija usluge i pretvorbu sadržaja poruka koje usluga prima i prosljeđuje.



Slika 13: Primjer složene usluge ostvarene primjenom metode koreografije usluga

Slika 13 prikazuje primjer složene usluge ostvarene primjenom metode koreografije usluga. Složena usluga zasniva se na uslugama A , B , C i D . Na početku izvođenja složene usluge, korisnik K usmjerava poruku zahtjeva usluzi A (1). Usluga A istovremeno prosljeđuje rezultate obrade primljenog zahtjeva uslugama B i C (2, 3). Usluge B i C potvrđuju primitak zahtjeva (4, 5) i usporedno obrađuju primljene zahtjeve. Nakon što završe obradu primljenih zahtjeva, usluge B i C šalju rezultate obrade usluzi D (6, 7) koja potvrđuje primitak poruka zahtjeva (8, 9). Nakon što je potvrdila primitak oba zahtjeva, usluga D obrađuje primljene zahtjeve i šalje poruku s konačnim rezultatom usluzi A (10). Usluga A potvrđuje primitak poruke s konačnim rezultatom (11) i prosljeđuje primljenu poruku korisniku složene usluge (12).

Osnovna prednost primjene metode koreografije usluga je raspodijeljeno upravljanje tijekom izvođenja složene usluge, proširivost i prilagodljivost složene usluge, te mogućnost kompozicije usluga koje se nalaze u različitim organizacijskim domenama. Raspodijeljeno upravljanje tijekom izvođenja složene usluge pozitivno utječe na svojstvo razmjernog rasta složene usluge obzirom na brojnost korisnika i usluga koje koristi složena usluga. Složene usluge ostvarene primjenom metode koreografije usluga imaju svojstvo proširivosti i prilagodljivosti. Ponašajna svojstva svake usluge koja čini složenu uslugu moguće je proširiti ili prilagoditi izmjenom opisa koreografije usluga bez utjecaja na ostale usluge. Poslovne aktivnosti i povezanosti između različitih organizacija vrlo su dinamične. Postavljanje i održavanje složenih usluga između različitih organizacijskih domena nije moguće ostvariti središnjim upravljanjem zbog dinamičkih i promjenjivih tržišnih uvjeta. Koreografija usluga koristi se, stoga, kako bi se uspostavili dogovori suradnje između usluga različitih organizacija. Osnova za ostvarivanje elektroničke poslovne suradnje je usuglašavanje

obrazaca za međusobnu razmjenu poruka i sadržaja razmijenjenih poruka, što je moguće ostvariti primjenom metode koreografije.

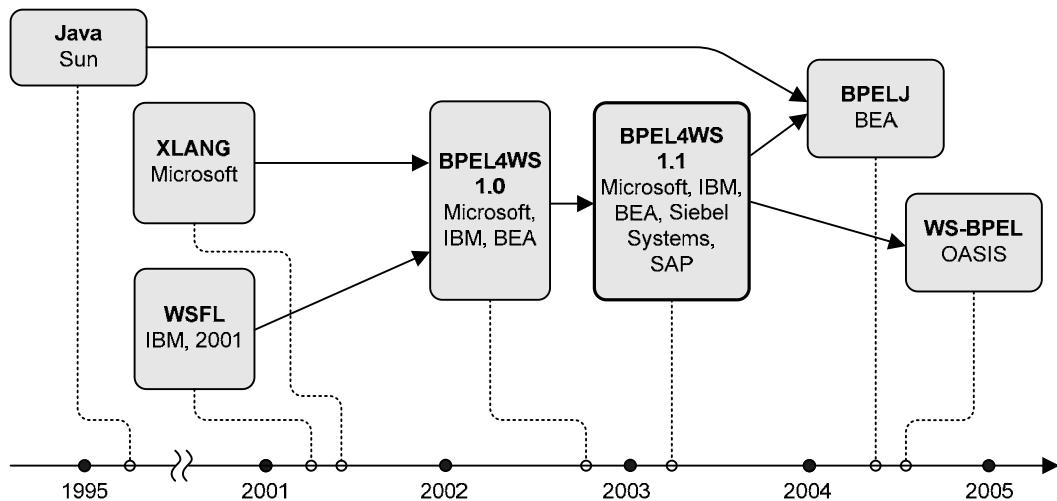
Osnovni nedostatci primjene koreografije usluga su nepostojanje središnjeg upravljačkog mjesta s uvidom u globalno stanje tijeka izvođenja složene usluge i nemogućnost korištenja mehanizama za međusobno isključivanje izvođenja usluga koje čine složenu uslugu. Raspodijeljeno upravljanje tijekom izvođenja složene usluge otežava otkrivanje i uklanjanje pogrešaka tijekom izvođenja složene usluge. Svaka usluga koja čini složenu uslugu mora samostalno otkrivati i provoditi akcije za uklanjanje pogrešaka u radu primjenom vlastitog lokalnog znanja o tijeku izvođenja složene usluge. Primjenom metode koreografije usluga nije moguće izravno ostvariti međusobno isključivanje izvođenja usluga koje čine složenu uslugu. Za potrebe ostvarivanje međusobnog isključivanja usluga potrebno je koristiti specijalizirane usluge koje ostvaruju mehanizam za međusobno isključivanje usluga tijekom izvođenja.

Jezici za opis složenih usluga

Postoji veliki broj jezika za izgradnju složenih usluga koje su zasnovane na opisima procesa i omogućavaju izgradnju složenih usluga primjenom metoda orkestracija ili koreografije [76]. Mnogi jezici za opisivanje složenih usluga nisu zaživjeli u širokoj primjeni ili nikada nije završen postupak njihove standardizacije. Jezici za izgradnju složenih usluga omogućavaju primjenu različitih naredbi za ostvarivanje tijeka izvođenja složene usluge. Svojstva jezika za ostvarivanje orkestracije i koreografije usluga većinom su naslijedjena od jezika koji se koriste u sustavima za upravljanje tijekom rada (engl. *workflow management systems*) [77]. Najšire prihvaćeni jezik za opisivanje složenih usluga primjenom metode orkestracije je jezik *BPEL4WS* (*Business Process Execution Language for Web Services*) [57]. Najpoznatiji jezik za opisivanje složenih usluga primjenom metode koreografije je jezik *WS-CDL* (*Web Service Choreography Description Langauge*) [58].

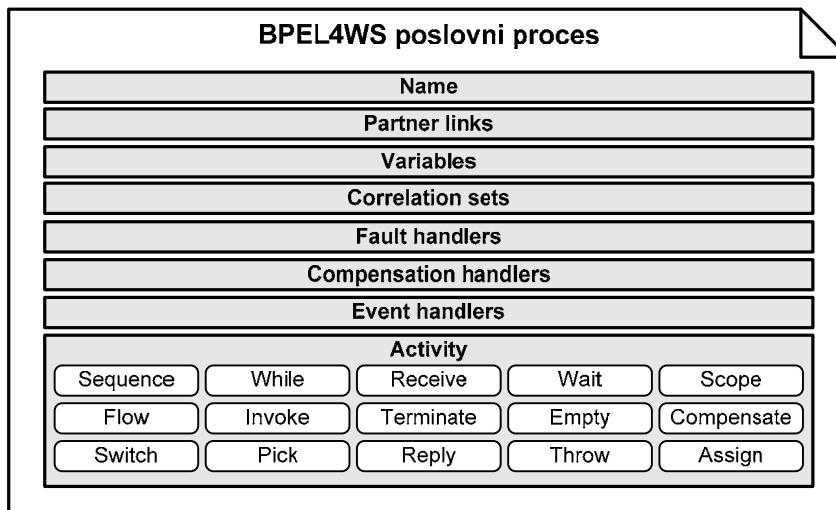
Jezik *BPEL4WS* omogućava izgradnju raspodijeljenih aplikacija zasnovanih na *Web Services* tehnologiji primjenom metode orkestracije usluga. Opisi *BPEL4WS* poslovnih procesa zasnovani su na jeziku *XML*, što osigurava njihovu prenosivost i uskladivost. Nadalje, opisi *BPEL4WS* poslovnih procesa koristite se u paru s *WSDL*, *WS-Transaction*, *WS-Coordination* i *WS-Addressing* [59] [60] specifikacijama proširenog *WS-** skupa specifikacija. Jezik *WSDL* omogućava opisivanje pristupnih sučelja poslovnog procesa. Opis sučelja uključuje definicije ulaznih i izlaznih poruka za svaku operaciju poslovnog procesa. Nadalje, jezik *WSDL* koristi se i za opisivanje pristupnih sučelja usluga kojim poslovni proces pristupa tijekom izvođenja. Specifikacije *WS-Transactions* i *WS-Coordination* koriste

se za izgradnju transakcija koje koriste *BPEL4WS* poslovni procesi. Specifikacije *WS-Addressing* primjenjuje se za naslovljavanje udaljenih usluga koje poslovni proces koristi tijekom izvođenja.



Slika 14: Razvoj jezika *BPEL4WS*

Slika 14 prikazuje razvoj jezika *BPEL4WS*. Prvu inačicu specifikacije jezika *BPEL4WS* zajednički su izradile tvrtke *Microsoft*, *IBM* i *BEA* krajem 2002. godine. Predložena specifikacija nastala je na osnovi jezika *XLANG (Web Services flow Language)* [78] i jezika *WSFL (Web Services flow Language)* [79]. Jezik *XLANG* omogućava opisivanje poslovnih procesa koje je moguće izvoditi primjenom prvih inačica *Microsoft BizTalk* poslužitelja poslovnih procesa [80]. Jezik *WSFL* je grafički jezik za opisivanje poslovnih procesa koje je moguće izvoditi primjenom sustava *IBM FLOWer*. Sredinom prve polovice 2003. godine, tvrtke *Microsoft*, *IBM*, *BEA*, *Siebel Systems* i *SAP* izradile su inačicu 1.1 specifikacije jezika *BPEL4WS*. Proširivanjem specifikacije 1.1 jezika *BPEL4WS*, kompanija *BEA* izradila je novi jezik nazvan *BPELJ*. Jezik *BPELJ* ostvaren je proširivanjem svojstva jezika *BPEL4WS* 1.1 s mogućnošću pisanja isječaka logike poslovnog procesa primjenom programskog jezika *Java*. Primjena programskog jezika *Java* u naredbama jezika *BPEL4WS* omogućava ostvarivanje naprednije kontrole toka i obrade podataka u poslovnom procesu. Na osnovi specifikacije jezika *BPEL4WS* 1.1, standardizacijska organizacija *OASIS* trenutno provodi postupak standardizacije nove inačice jezika *WS-BPEL* [81], koja proširuje jezik *BPEL4WS* sa standardnim konstruktima za napredniju kontrolu toka i upravljanje podacima poslovnog procesa.



Slika 15: Osnovna struktura opisa poslovnog procesa napisanog primjenom jezika *BPEL4WS*

Slika 15 prikazuje osnovnu strukturu opisa poslovnog procesa napisanog primjenom jezika *BPEL4WS*. Opis *BPEL4WS* poslovnog procesa sadrži sljedeće elemente: *Name*, *Partner links*, *Variables*, *Correlation sets*, *Fault handlers*, *Compensation handlers*, *Event handlers* i *Activity*. Element *Name* definira ime *BPEL4WS* poslovnog procesa. Element *Partner links* služi za definiranje simboličkih imena koja služe za naslovljavanje udaljenih usluga koje koristi poslovni proces tijekom izvođenja. Element *Variables* služi za definiranje varijabli koje spremaju stanje poslovnog procesa tijekom izvođenja. Element *Correlation sets* služi za definiranje pravila za uparivanje poruka zahtjeva i odgovora koje upućuju i primaju različiti primjerici istog poslovnog procesa. Element *Fault handlers* sadrži naredbe koje se izvode u slučaju pogreške tijekom izvođenja poslovnog procesa. Element *Compensation handlers* sadrži naredbe koje se izvode tijekom provođenja postupka oporavka od pogreške. Element *Event handlers* sadrži naredbe koje se izvode u slučaju pojave posebnih dogadaja tijekom izvođenja poslovnog procesa. Primjer mogućih dogadaja su istek brojača vremena (engl. *timer*) ili primitak poruke zahtjeva s određenim sadržajem. Element *Activity* sadrži naredbe koje opisuju logiku poslovnog procesa.

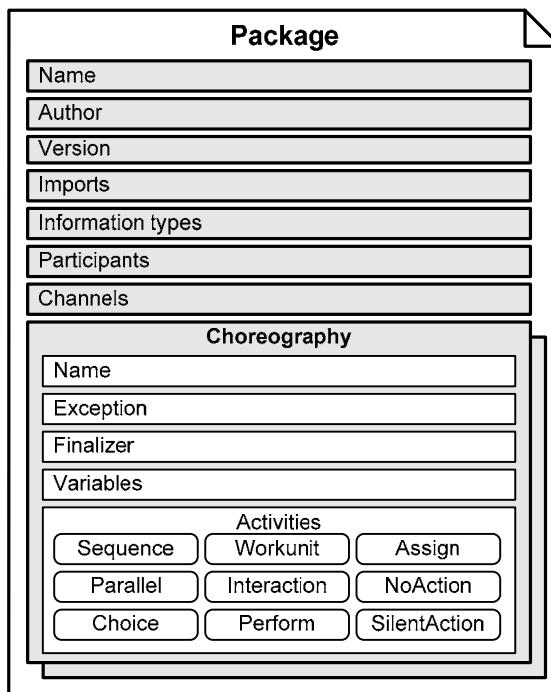
Naredbe jezika *BPEL4WS* grupirane su u osnovne (engl. *basic*) i strukturne (engl. *structural*) naredbe. Osnovne naredbe čine naredbe: *invoke*, *receive*, *reply*, *assign*, *throw*, *wait* i *empty*. Naredba *invoke* omogućava poziv udaljenih usluga koje poslovni proces koristi tijekom izvođenja. Naredba *receive* omogućava prihvatanje poruka pristiglih pozivom neke od operacija pristupnog sučelja poslovnog procesa. Naredba *reply* omogućava proslijedivanje poruke odgovora za poruku zahtjeva prethodno primljenu tijekom poziva određene operacije pristupnog sučelja poslovnog procesa. Naredba *assign* omogućava upravljanje vrijednostima

spremljenim u varijablama poslovnog procesa. Naredba *throw* omogućava dojavu iznimke u slučaju pogreške tijekom izvođenja poslovnog procesa. Naredba *wait* omogućava privremeno obustavljanje tijeka izvođenja poslovnog procesa za definirani vremenski period. Naredba *empty* omogućava izvođenje aktivnosti koja nema nikakvog učinka na stanje poslovnog procesa i udaljenih usluga koje proces koristi.

Strukturne naredbe su naredbe *sequence*, *switch*, *pick*, *while* i *flow*. Naredba *sequence* omogućava slijedno izvođenje niza naredbi. Naredba *switch* omogućava uvjetno grananje tijeka izvođenja poslovnog procesa. Naredba *pick* omogućava prihvaćanje poziva jedne od više navedenih operacija pristupnog sučelja poslovnog procesa. Naredba *while* omogućava uvjetno ponavljanje niza naredbi. Naredba *flow* omogućava istovremeno grananje tijeka izvođenja poslovnog procesa. Primjenom naredbe *flow* moguće je ostvariti istovremeni poziv nekoliko udaljenih usluga.

Web Services Choreography Description Language

Jezik *WS-CDL* omogućava izgradnju složenih usluga primjenom metode koreografije. Standardizaciju jezika *WS-CDL* provodi radna grupa koja je osnovana krajem 2004. godine u organizaciji *W3C*. Jezik *WS-CDL* omogućava izgradnju složenih usluga primjenom ugovora koji definiraju obrasce za razmjenu poruka i sadržaj poruka koje usluge međusobno razmjenjuju kako bi ostvarile složenu uslugu.



Slika 16: Osnovna struktura opisa koreografije složene usluge ostvarenog primjenom jezika *WS-CDL*

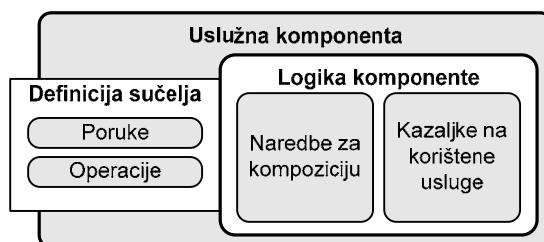
Slika 16 prikazuje osnovnu strukturu opisa koreografije složene usluge napisanog primjenom jezika *WS-CDL*. Opis koreografije složene usluge sadržan je u dokumentu imena *Package*. Dokument koreografije složene usluge sadrži elemente: *Name*, *Author*, *Version*, *Imports*, *Information types*, *Participants*, *Channels* i *Choreography*. Element *Name* određuje ime koreografije složene usluge. Element *Author* određuje ime tvorca dokumenta koreografije. Element *Version* određuje inačicu dokumenta koreografije. Element *Imports* određuje dodatne vanjske opise koreografija na kojima se zasniva ostvareni dokument koreografije. Element *Information types* definira tipove podataka koje usluge međusobno izmjenjuju tijekom izvođenja složene usluge. Element *Participants* definira osnovne usluge koje čine složenu uslugu. Element *Channels* sadrži skup simboličkih imena za naslovljavanje usluga koje čine složenu uslugu. Element *Choreography* sadrži opis koreografije za svaku od usluga koja čini složenu uslugu.

Element *Choreography* sadrži elemente: *Name*, *Exception*, *Finalizer*, *Variables* i *Activities*. Element *Name* određuje ime koreografije usluge. Element *Exception* sadrži naredbe koje se izvode u slučaju pogreške tijekom izvođenja koreografije usluge. Element *Finalizer* sadrži naredbe koje se izvode u slučaju uspješnog završetka izvođenja koreografije osnovne usluge ili pogreške tijekom izvođenja koreografije neke od udaljenih usluga koje čine složenu uslugu. Element *Variables* sadrži deklaracije varijabli koje koristi koreografija usluge tijekom izvođenja. Element *Activities* sadrži naredbe koje opisuju logiku koreografije određene osnovne usluge. Jezik *WS-CDL* podržava tri skupine naredbi: naredbe za upravljanje tijekom izvođenja (engl. *control flow activities*), naredbe za iterativno izvođenje slijeda naredbi (engl. *work unit activities*) i osnovne naredbe (engl. *basic activities*).

Naredbe za upravljanje tijekom izvođenja uključuju naredbe *sequence*, *parallel* i *choice*. Naredba *sequence* omogućava slijedno izvođenje skupa naredbi. Naredba *parallel* omogućava istovremeno izvođenje skupa naredbi. Naredba *choice* omogućava uvjetno grananje tijeka izvođenja. Jezik *WS-CDL* sadrži sljedeće osnovne naredbe: *Interaction*, *NoAction*, *SilentAction* i *Assign*. Naredba *Interaction* omogućava uspostavljanje komunikacije između usluga koje čine složenu uslugu. Naredba *NoAction* definira akciju koja nema utjecaja na ukupno izvođenje koreografija usluge. Naredba *SilentAction* omogućava definiranje akcije koja nema utjecaja na ostale sudionike koreografije, a ostvarena pozivom lokalne operacije usluge. Naredba *Assign* omogućava upravljanje sadržajem lokalnih varijabli koreografije usluge. Naredba *Perform* omogućava izvođenje vanjske koreografije kao dio aktivne koreografije.

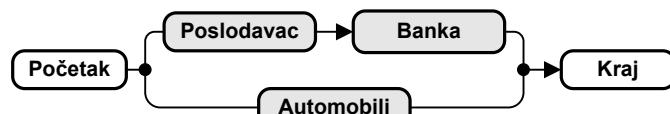
3.2.2 Uslužne komponente

Metoda kompozicije usluga zasnovana na uslužnim komponentama (engl. *service components*) omogućava razvoj složenih usluga primjenom hibridne metode razvoja zasnovane na uslugama i komponentama [82]. Uslužne komponente omogućavaju razvoj raspodijeljenih aplikacija zasnovanih na uslugama korištenjem mehanizama nasljeđivanja (engl. *inheritance*) i specijalizacije (engl. *specialization*). Mehanizam nasljeđivanja omogućava ponovno korištenje funkcionalnosti postojećih složenih usluga pri izgradnji novih složenih usluga. Mehanizam specijalizacije omogućava izgradnju novih složenih usluga izmjenom dijela funkcionalnosti koje ostvaruje neka od postojećih složenih usluga.



Slika 17: Struktura uslužne komponente

Slika 17 prikazuje strukturu uslužne komponente koja uključuje definiciju sučelja i definiciju logike uslužne komponente. Definicija sučelja uslužne komponente određuje strukturu poruka i operacije koje komponenta izlaže putem pristupnog sučelja. Logika uslužne komponente sadrži naredbe za kompoziciju i kazaljke na udaljene usluge koje komponenta koristi tijekom izvođenja. Konstrukti za kompoziciju opisuju obrasce razmjene poruka s udaljenim uslugama koje komponenta koristi tijekom izvođenja.



Slika 18: Tijek izvođenja složene usluge za odabir automobila

Slika 18 prikazuje tijek izvođenja složene usluge na primjeru kupovine automobila. Složena usluga za kupovinu automobila ostvarena je primjenom usluga *Poslodavac*, *Banka* i *Automobili*. Složena usluga započinje istovremenim izvođenjem usluga *Poslodavac* i *Automobili*. Usluga *Poslodavac* omogućava dohvaćanje prosjeka plaća kupca automobila. Usluga *Automobili* omogućava dohvaćanje trenutne ponude na tržištu automobila. Nakon završetka izvođenja usluge *Poslodavac*, izvodi se usluga *Banka* koja na osnovi utvrđenog prosjeka plaća kupca određuje mogući iznos kredita za kupovinu automobila. Na kraju

izvođenja složene usluge, uspoređuju se dozvoljeni iznos kredita i trenutna ponuda automobila na tržištu kako bi se odredili automobili koje kupac ima mogućnost kupiti.

```

ServiceComponentClass KupovinaAutomobila {
    Definition
        d1: PronadiZahtjev zahtjev
        d2: PronadiOdgovor odgovor
        d3: Pronadi( in zahtjev, out odgovor )
    Construction
        c1: parallel( c2, ponudaAutomobila )
        c2: sequential( kreditnaSposobnost, moguciKrediti )
    PortType
        pt1: PoslodavacPT.sposobnost kreditnaSposobnost
        pt2: BankaPT.krediti moguciKrediti
        pt3: AutomobiliPT.prodaja ponudaAutomobila
    Provider
        pv1: Poslodavac PoslodavacPT
        pv2: Banka BankaPT
        pv3: Automobili AutomobiliPT
    MessageHandling
        m1: messageDecomposition(
            Pronadi.zahtjev,
            KreditiPT.sposobnost.zahtjev,
            AutomobiliPT.prodaja.zahtjev
        )
        m2: messageSynthesis(
            KreditiPT.sposobnost.odgovor,
            BankaPT.krediti.zahtjev
        )
        m3: messageSynthesis(
            BankaPT.krediti.odgovor,
            AutomobiliPT.prodaja.odgovor,
            Pronadi.odgovor
        )
    }
}

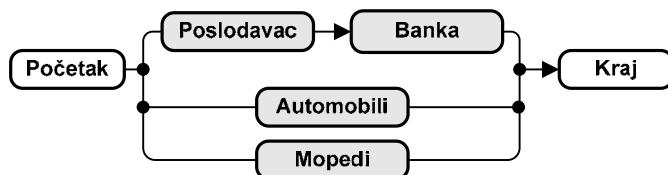
```

Slika 19: Ostvarenje složene usluge za odabir automobila primjenom uslužne komponente

Slika 19 prikazuje definiciju uslužne komponente koja ostvaruje složenu uslugu za odabir automobila. Definicija uslužne komponente sadrži elemente **Definition**, **Construction**, **PortType**, **Provider** i **MessageHandling**. Element **Definition** služi za definiranje ulaznih poruka, izlaznih poruka i operacija uslužne komponente. Uslužna komponenta za odabir automobila sadrži samo operaciju *Pronadi* (d3). Operacija *Pronadi* izvodi se nakon primjeka zahtjeva *PronadiZahtjev* (d1) koji sadrži podatke o kupcu automobila. Nadalje, ponudu automobila koje kupac može kupiti sadrži poruka odgovora *PronadiOdgovor* (d2) koju uslužna komponenta prosljeđuje nakon završetka izvođenja operacije *Pronadi*. Element **Construction** služi za definiranje slijeda pozivanja operacija usluga koje uslužna komponenta koristi tijekom izvođenja. Uslužna komponenta za kupovinu automobila istovremeno poziva dvije skupine operacija (c1). U prvoj skupini usluga slijedno se poziva operacija *kreditnaSposobnost* usluge *Poslodavac*, a zatim se poziva

operacija *moguciKrediti* usluge *Banka* (c2). U drugoj skupini usluga poziva se samo operacija *ponudaAutomobila* usluge *Automobili* (c1). Element **PortType** služi za definiranje simboličkih imena operacija svake udaljene usluge koju koristi uslužna komponenta. Uslužna komponenta koristi simbolička imena *kreditnaSposobnost* (pt1), *moguciKrediti* (pt2) i *ponudaAutomobila* (pt3) za naslovljavanje operacija *sposobnost*, *krediti* i *prodaja* udaljenih usluga *Poslodavac*, *Banka* i *Automobili*. Element **Provider** omogućava deklaraciju simboličkih imena pomoću kojih uslužna komponenta naslovljava sučelja udaljenih usluga koje koristi. Uslužna komponenta koristi simbolička imena *PoslodavacPT* (pv1), *BankaPT* (pv2) i *AutomobilPT* (pv3) koja služe za naslovljavanje pristupnih sučelja udaljenih usluga *Poslodavac*, *Banka* i *Automobil*. Element **MessageHandling** služi za definiranje pravila za pretvorbu poruka koje se izmjenjuju između usluga koje koristi uslužna komponenta. Postupci pretvorbe podataka definiraju se primjenom naredbi **messageDecomposition** i **messageSynthesis**. Naredba **messageDecomposition** definira izgradnju skupa poruka korištenjem jedne poruke. Izgrađene poruke mogu se koristiti kao poruke zahtjeva za više različitih udaljenih usluga. Naredba **messageSynthesis** definira prikupljanje izlaznih poruka više usluga u jednu poruku zahtjeva koja se prosljeđuje kao poruka zahtjeva jednoj usluzi. Uslužna komponenta za odabir automobila koristi jednu naredbu **messageDecomposition** za izgradnju ulaznih podataka za pristup uslugama *Poslodavac* i *Automobil* (m1). Nadalje, uslužna komponenta koristi jednu naredbu **messageSynthesis** za pretvorbu podataka između usluge *Poslodavac* i *Banka* (m2), te jednu naredbu **messageSynthesis** za izgradnju poruke odgovora koja sadrži podatke dobivene izvođenjem usluga *Banka* i *Automobili* (m3).

Uporabom izgrađenog opisa uslužne komponente moguće je izgraditi novu uslužnu komponentu primjenom metoda nasljedivanja i specijalizacije. Primjerice, postojeći tijek izvođenja složene usluge za odabir automobila moguće je proširiti uvođenjem nove usluge *Mopedi*. Usluga *Mopedi* omogućava dohvaćanje trenutne ponude na tržištu mopeda. Osim uvođenja nove usluge, operaciju *Pronadi* postojeće uslužne komponente moguće je izmijeniti uvođenjem novih izlaznih podataka koji sadrže dodatne informacije o rezultatima izvođenja uslužne komponente.



Slika 20: Tijek izvođenja složene usluge za odabir automobila i mopeda

Slika 20 prikazuje tijek izvođenja složene usluge za odabir automobila i mopeda. Prikazana složena usluga ostvarena je proširivanjem tijeka izvođenja složene usluge za odabir automobila prikazane na slici 18. Uvedena je nova usluga *Mopedi* koja se izvodi istovremeno s uslugom *Automobili*.

```

serviceComponentClass KupovinaAutomobilaIMopeda
  SubClassOf KupovinaAutomobila {
    Definition
      d3: Pronadi( in zahtjev, out odgovor, out detalji )
      d4: Detalji detalji
    Construction
      c3: parallel( c1, ponudaMopeda )
    PortType
      pt4: MopediPT.prodaja ponudaMopeda
    Provider
      pv4: Mopedi MopediPT
    MessageHandling
      m1: messageDecomposition(
        Pronadi.zahtjev,
        KreditiPT.sposobnost.zahtjev,
        AutomobiliPT.prodaja.zahtjev,
        MopediPT.prodaja.zahtjev,
      )
      m3: messageSynthesis(
        BankaPT.krediti.odgovor,
        AutomobiliPT.prodaja.odgovor,
        MopediPT.prodaja.odgovor,
        Pronadi.zahtjev
      )
      m4: messageSynthesis(
        BankaPT.krediti.odgovor,
        AutomobiliPT.prodaja.odgovor,
        AutomobiliPT.prodaja.odgovor,
        Pronadi.detalji
      )
  }
}

```

Slika 21: Ostvarenje složene usluge za odabir automobila i mopeda primjenom uslužne komponente

Slika 21 prikazuje opis uslužne komponente *KupovinaAutomobilaIMopeda* koja je ostvarena nasljeđivanjem opisa uslužne komponente *KupovinaAutomobila* prikazane na slici 19. Nasljeđivanje opisa uslužne komponente *KupovinaAutomobila* ostvaruje se primjenom ključne riječi **SubClassOf**. Element **Definitions** proširen je definicijom dodatnog izlaznog parametra *detalji* (d4) za operaciju *Pronadi* (d3). Element **Construction** proširen je dodatnom naredbom koja definira istovremeno pozivanje operacija *ponudaMopeda* (c3) usluge *Mopedi* i operacije *ponudaAutomobila* (c1) usluge *Automobili*. Element **PortType** proširen je dodatnim simboličkim imenom *ponudaMopeda* (pt4) koje služi za naslovljavanje operacije *prodaja* pristupnog sučelja usluge *Mopedi*. Element **Provider** proširen je dodatnim

simboličkim imenom *MopediPT* (pv4) koje služi za naslovljavanje pristupnog sučelja usluge *Mopedi*. Element **MessageHandling** sadrži naslijedene naredbe **messageSynthesis** (m1) i **messageDecomposition** (m3) koje su proširene tako da uključuju korištenje usluge *Mopedi*. Nadalje, uključena je i dodatna naredba **messageSynthesis** koja omogućava izgradnju sadržaja dodatnog izlaznog parametra *detalji* operacije *Pronadi* (m4).

Osnovna prednost metode kompozicije usluga zasnovane na uslužnim komponentama je mogućnost primjene metoda nasljeđivanja i specijalizacije tijekom oblikovanja složenih usluga. Primjena metoda nasljeđivanja i specijalizacije omogućava ubrzani razvoj složenih usluga ponovnim korištenjem funkcionalnosti postojećih složenih usluga. Osim navedenih mehanizama, primjena uslužnih komponenata omogućava provjeravanje svojstava zamjenjivosti (engl. *compatibility*) i uskladivosti (engl. *conformance*) uslužnih komponenti. Prema načelu zamjenjivosti, uslužna komponenta A ostvarena nasljeđivanjem roditeljske komponente B može biti izravno korištena i kao uslužna komponenta B, obzirom da je naslijedila sve njezine funkcionalnosti. Prema načelu uskladivosti, uslužne komponente A i B su uskladive ako ih je moguće izravno povezati tako da poruka odgovora primljena nakon izvođenja komponente A može biti izravno proslijeđena kao poruka zahtjeva komponenti B. Usklađenost poruka komponenata A i B moguća je samo ako operacije obje komponente koriste zajedničke definicije poruka.

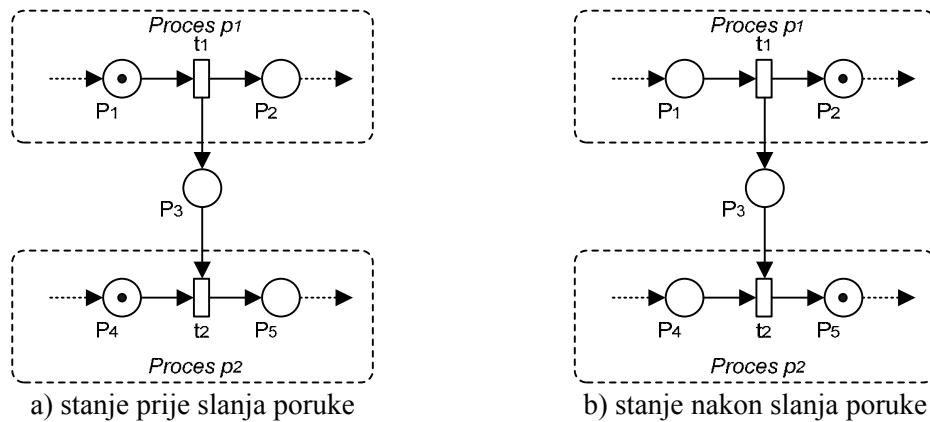
Osim navedenih prednosti, primjena uslužnih komponenata u svrhu kompozicije usluga ima nekoliko nedostataka. Obzirom da ne postoje standardne biblioteke s opisima već ostvarenih uslužnih komponenata, uslužne komponente u većini slučajeva potrebno je graditi iz početka. Opisi izgrađeni za većinu stvarnih uslužnih komponenata koje koriste veliki broj usluga u pravilu su veliki, nepregledni i složeni.

3.2.3 Petrijeve mreže

Petrijeve mreže (engl. *Petri-nets*) [83] su formalni grafički jezik koji omogućava izgradnju modela računalnih sustava koji sadrže aktivne komponente ostvarene procesima. Modeli zasnovani na Petrijevim mrežama omogućavaju simuliranje i analizu raznovrsnih ponašajnih svojstava računalnih sustava.

Petrijeva mreža je usmjereni i povezani graf koji sadrži dvije vrste čvorova: mjesta (engl. *places*) i prijelaze (engl. *transitions*). Svaka usmjereni grana Petrijeve mreže može povezivati samo jedno mjesto s prijelazom ili jedan prijelaz s mestom. Mjesta Petrijeve mreže prikazuju se kružnicama, a prijelazi pravokutnicima. Mjesta se koriste za modeliranje

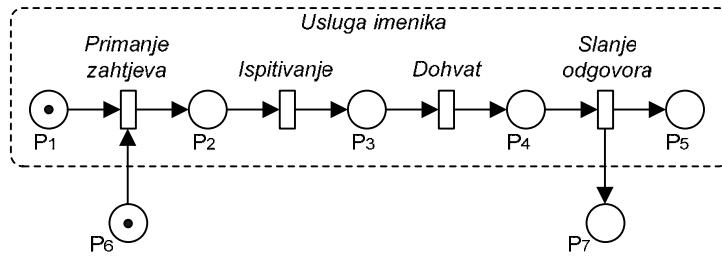
stanja sustava. U svakom mjestu Petrijeve mreže može se nalaziti nula ili više znački (engl. *tokens*) koje se označavaju točkom unutar kružnice mjesta. Broj značaka u određenom mjestu Petrijeve mreže određuje stanje mesta. Ukupno stanje Petrijeve mreže određeno je rasporedom značaka po njegovim mjestima. Prijelazi se koriste za modeliranje događaja u sustavu, pri čemu se događaji modeliraju okidanjem prijelaza (engl. *transition firing*). Prijelaz može biti okinut samo ako se u svim njegovim ulaznim mjestima nalazi barem jedna značka. Po okidanju prijelaza, iz svih ulaznih mesta prijelaza uklanja se jedna značka i postavlja se jedna dodatna značka u svako od izlaznih mesta prijelaza.



Slika 22: Primjer modela komunikacije između procesa ostvarenog Petrijevom mrežom

Slika 22 prikazuje odsječak jednostavne Petrijeve mreže koja modelira komunikaciju između dva procesa (p_1, p_2). Slika 22a) prikazuje početno stanje Petrijeve mreže u kojem se u mjestima P_1 i P_4 nalazi po jedna značka. Mjesto P_1 služi za modeliranje stanja u kojem je proces p_1 pripravan za slanje poruke procesu p_2 . Mjesto P_4 služi za modeliranje stanja u kojem je proces p_2 pripravan za primanje poruke od strane procesa p_1 . Komunikacijski kanal modelira se primjenom mesta P_3 . Mjesto P_3 služi za spremanje značke koja predstavlja poruku usmjerenu od procesa p_1 prema procesu p_2 . Okidanjem prijelaza t_1 , proces p_1 uklanja značku iz mesta P_1 i postavlja značku koja predstavlja poruku u mjesto P_3 . Okidanjem prijelaza t_2 , proces p_2 uklanja značku iz mesta P_4 i uklanja značku iz mesta P_3 , čime se modelira primitak poslane poruke. Slika 22b) prikazuje završno stanje Petrijeve mreže nakon razmjene poruke. Postavljanjem značke u mjesto P_2 modelira se stanje sustava u kojem je proces p_1 uspješno odaslao poruku. Nadalje, postavljanjem značke u mjesto P_5 modelira se stanje sustava kojem je proces p_2 uspješno primio odaslanu poruku.

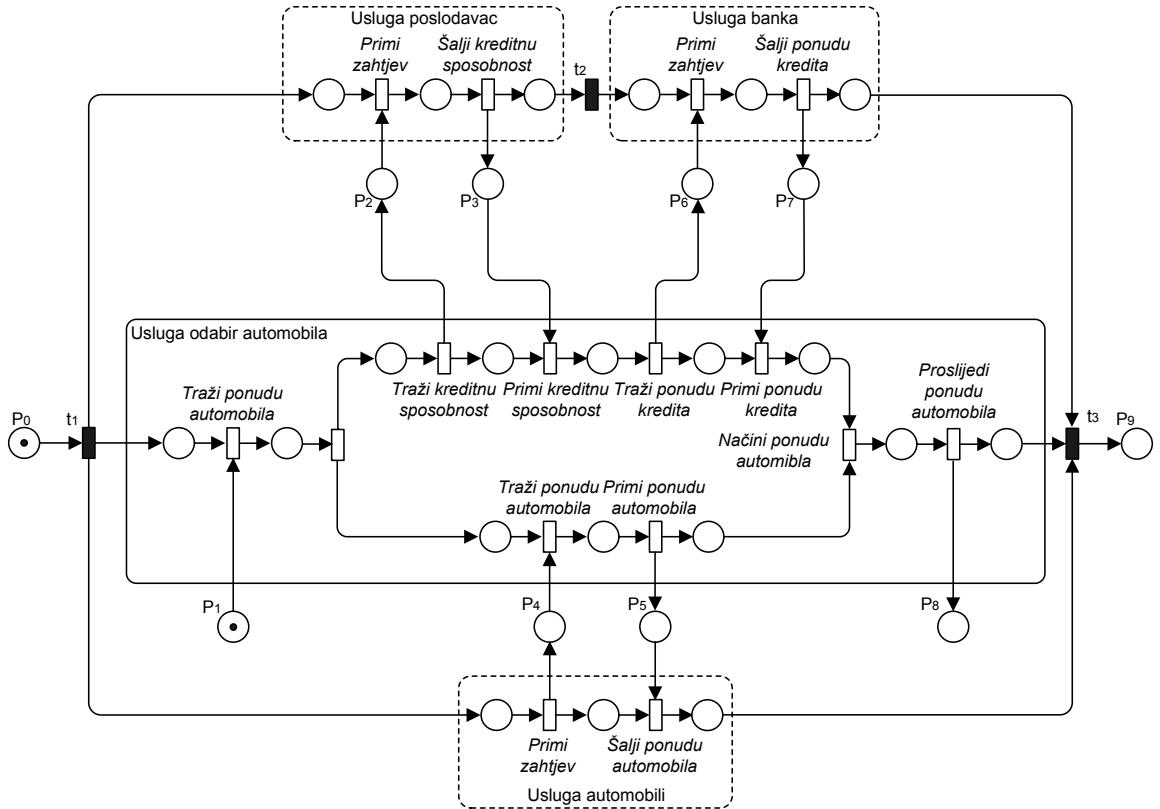
Petrijeve mreže moguće je primijeniti u svrhu izgradnje modela usluga [84]. Prijelazi Petrijevih mreža koriste se za modeliranje izvođenja akcija usluga. Nadalje, mesta se koriste za modeliranje stanja kroz koja usluga prolazi tijekom izvođenja.



Slika 23: Primjer modela usluge *Imenik* ostvaren Petrijevom mrežom

Slika 23 prikazuje primjer modela usluge *Imenik*, ostvarenog Petrijevom mrežom. Usluga *Imenik* omogućava dohvati imena osobe koja je korisnik određenog telefonskog broja. Usluga je u pripravnom stanju kada se u mjestu P_1 nalazi značka. Mjesto P_6 služi za spremanje značaka koje predstavljaju pristigle poruke zahtjeva. Postavljanjem značke u mjesto P_6 omogućeno je okidanje prijelaza *Primanje zahtjeva* koji služi za modeliranje primitka poruke zahtjeva. Nakon okidanja prijelaza *Primanje zahtjeva*, okida se prijelaz *Ispitivanje* koji modelira akciju ispitivanja valjanosti zapisa primljenog telefonskog broja. Nakon okidanja prijelaza *Ispitivanje*, okida se prijelaz *Dohvat* koji modelira dohvaćanje imena korisnika primljenog telefonskog broja iz baze podataka. Okidanjem prijelaza *Slanje odgovora* modelira se slanje poruke odgovora korisniku usluge. Poruku odgovora predstavlja značka postavljena u mjesto P_7 . Postavljanjem značke u mjesto P_5 modelira se završetak izvođenja usluge *Imenik*.

Modeli složenih usluga zasnovani na Petrijevim mrežama grade se međusobnim povezivanjem jednostavnijih modela usluga koje čine složenu uslugu. Modeli složenih usluga povezuju se primjenom različitih obrazaca za sinkronizaciju i komunikaciju usluga prema zahtjevima tijeka izvođenja složene usluge. Najčešće se koriste obrasci za slijedno izvođenje (engl. *sequence*), uvjetno grananje (engl. *choice*), grananje tijeka izvođenja (engl. *parallel split*), sinkronizaciju (engl. *synchronization*) i ponavljanje (engl. *iteration*) [84].



Slika 24: Primjer modela složene usluge ostvaren Petrijevom mrežom

Slika 24 prikazuje primjer modela složene usluge za odabir automobila čiji je tijek izvođenja prikazan na slici 18. Složena usluga za odabir automobila ostvarena je povezivanjem modela usluga *Poslodavac*, *Banka*, *Automobili* i *Odabir automobila* koji su ostvareni zasebnim Petrijevim mrežama. Usluge *Poslodavac*, *Banka*, *Automobili* i *Odabir automobila* povezane su prijelazom t_1 koji omogućava istovremeno grananje tijeka izvođenja usluge. Usluge *Poslodavac* i *Banka* međusobno su povezane prijelazom t_2 koji ostvaruje obrazac za slijedno izvođenje. Istovremeni tijekovi izvođenja usluga *Poslodavac*, *Banka*, *Automobili* i *Odabir automobila* povezani su prijelazom t_3 koji omogućava njihovu sinkronizaciju u jedinstveni tijek izvođenja. Početno stanje složene usluge modelira se postavljanjem značke u mjesto P_0 , dok se završno stanje složene usluge modelira postavljanjem značke u mjesto P_9 . Mjesto P_1 služi za spremanje znački koje predstavljaju pristigle poruke zahtjeva. Nadalje, mjesto P_8 služi za spremanje znački koje predstavljaju poruke odgovora. Usluga *Odabir automobila* ostvaruje komunikaciju s udaljenim uslugama razmjenom značaka putem mjesta P_2 , P_3 , P_4 , P_5 , P_6 i P_7 . Mjesta P_2 i P_3 služe za modeliranje komunikacije s uslugom *Poslodavac*. Mjesta P_4 i P_5 služe za modeliranje komunikacije s uslugom *Automobili*. Mjesta P_6 i P_7 služe za modeliranje komunikacije s uslugom *Banka*.

Opisana metoda za izgradnju modela složenih usluga zasniva se na primjeni osnovnih Petrijevih mreža koje se grade korištenjem mjesta i prijelaza (engl. *Place-Transition Nets, PT-Nets*). Postoje i napredniji modeli Petrijevih mreža, kao što su obojane Petrijeve mreže (engl. *Coloured Petri Nets, CP-Nets*) [85]. Obojane petrijeve mreže omogućavaju izgradnju modela složenih usluga koji uključuju pojmove strukture podataka, vremena i hijerarhije. Primjenom opisa struktura podataka moguće je definirati strukturu i sadržaj poruka koje usluge međusobno razmjenjuju tijekom izvođenja složene usluge. Primjenom koncepta vremena moguće je modelirati trajanje vremenskih odziva tijekom komunikacije usluga. Primjenom pojma hijerarhije moguće je graditi preglednije modele složenih usluga grupiranjem usluga u više zasebnih hijerarhijskih razina.

Primjena Petrijevih mreža za potrebe modeliranja složenih usluga ima sljedeće prednosti [86]: formalna matematička osnova, grafička priroda jezika, izražajnost i općenitost. Formalna matematička osnova Petrijevih mreža omogućava provođenje analize ponašajnih svojstava izgrađenih modela složenih usluga. Najčešće se analiziraju svojstva potpunog zastoja, ograničenosti, dosežljivosti i pravednosti (engl. *fairness*) [87]. Grafička priroda jezika Petrijevih mreža omogućava jednostavniju i učinkovitiju izgradnju modela složenih usluga. Izražajnost Petrijevih mreža omogućava istovremeno modeliranje stanja i događaja složenih usluga. Primjenom Petrijevih mreža moguće je jednostavno modelirati različite obrasce za sinkronizaciju i komunikaciju usluga koji se koriste za izgradnju složenih usluga. Modeli složenih usluga ostvareni primjenom Petrijevih mreža ne ovise o tehnologiji za ostvarivanje složenih usluga, pa ih je stoga moguće primijeniti za modeliranje složenih usluga ostvarenih primjenom proizvoljne tehnologije za kompoziciju usluga.

Osnovni nedostaci primjene modela složenih usluga zasnovanim na Petrijevim mrežama su nepreglednost modela i složenost postupaka analize. Većina složenih usluga sadrži veliki broj usluga koje su međusobno povezane primjenom različitih obrazaca za sinkronizaciju i komunikaciju. Modeli složenih usluga ostvareni primjenom osnovnog modela Petrijevih mreža u pravilu su veliki i nepregledni. Složenost i nepreglednost modela moguće je umanjiti primjenom naprednijih Petrijevih mreža, koje sadrže strukture podataka, vrijeme i hijerarhiju. Većina postupaka za analizu ponašajnih svojstava Petrijevih mreža iznimno su zahtjevni i njihovo provođenje zahtjeva značajnu količinu računalnih sredstava.

3.2.4 Procesne algebre

Procesne algebre (engl. *process algebras*) čine skup formalizama koji omogućavaju modeliranje usporednih (engl. *parallel*) i raspodijeljenih (engl. *distributed*) računalnih

sustava [88]. Modeli računalnih sustava zasnivaju se na procesnim izrazima koji se grade primjenom operatora i skupa pravila procesne algebre. Primjenom modela ostvarenih procesnim algebrama moguće je ostvariti analizu ponašajnih svojstava računalnih sustava.

Osnovni elementi procesnih algebra su varijable i operatori. Varijable se primjenjuju za imenovanje procesa i komunikacijskih kanala putem kojih procesi međusobno razmjenjuju poruke. Operatori se koriste za modeliranje tijeka izvođenja procesa i razmjene poruka putem komunikacijskih kanala. Postoje različite vrste procesnih algebri koje je moguće koristiti za modeliranje različitih svojstava računalnih sustava. Procesne algebri razlikuju se prema stupnju izražajnosti, vrsti obrazaca komunikacije koje je moguće modelirati, pokretljivosti procesa, te mogućnostima dinamičkog stvaranja i uništavanja komunikacijskih kanala. Algebra π [89] jedna je od najšire prihvaćenih procesnih algebri. Osnovna svojstva algebri π su mogućnost modeliranja podataka, primjena sinkronog obrasca komunikacije razmjenom poruka, te mogućnost dinamičkog stvaranja i uništavanja komunikacijskih kanala.

Tablica 1: Popis produkcija gramatike za izgradnju procesnih izraza algebri π

Producija	Opis produkcije
$P_1 \rightarrow 0$	Proces koji ne sadrži akcije
$P_1 \rightarrow \bar{a}[x].P_2$	Proces šalje vrijednost pridruženu varijabli x putem kanala a i nastavlja s izvođenjem akcija procesa P_2
$P_1 \rightarrow a(x).P_2$	Proces čita vrijednost zapisanu u kanalu a , pridružuje pročitanu vrijednost varijabli x i nastavlja s izvođenjem akcija procesa P_2
$P_1 \rightarrow P_2 + P_3$	Primjenom nedeterminističkog izbora izvode se akcije procesa P_2 ili procesa P_3
$P_1 \rightarrow P_2 P_3$	Usporedno se izvode akcije procesa P_2 i procesa P_3
$P_1 \rightarrow !P_2$	Stvaranje nove instance procesa P_2
$P_1 \rightarrow \text{new } x P_2$	Stvaranje nove varijable x čiji je djelokrug deklaracije unutar procesa P_2

Tablica 1 prikazuje gramatiku koja omogućava izgradnju procesnih izraza algebri π . Izraz 3.1 predstavlja primjer procesnog izraza koji opisuje ponašajna svojstva triju procesa koji se izvode istovremeno. Prvi proces zapisuje vrijednost pridruženu varijabli a u komunikacijski kanal x , a drugi proces zapisuje vrijednost pridruženu varijabli b u komunikacijski kanal x . Treći proces čita vrijednost u komunikacijskom kanalu x , pridružuje pročitanu vrijednost varijabli u i zapisuje vrijednost pridruženu varijabli u u komunikacijski kanal y .

$$\bar{x}[a] | \bar{x}[b] | x(u). \bar{y}[u] \quad (3.1)$$

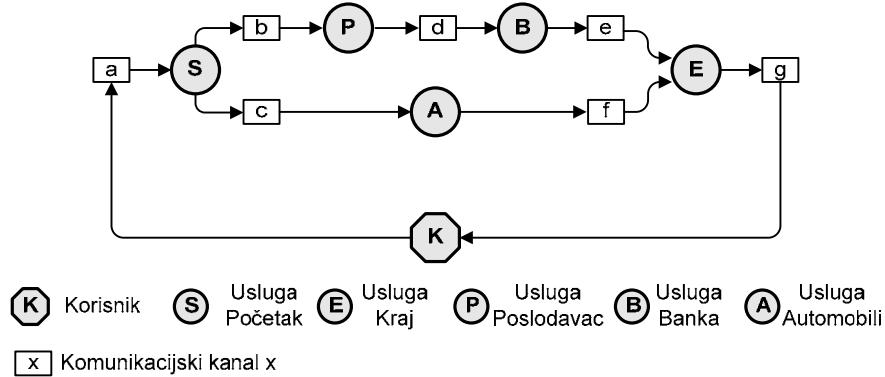
Pretvorbom izraza 3.1 moguće je provesti analizu ponašajnih svojstva izgrađenog modela. Analiza izgrađenog modela provodi se primjenom binarne relacije redukcije koja se označava oznakom \rightarrow . Relacija redukcije omogućava pretvorbu izraza procesne algebre s ciljem utvrđivanja rezultata izvođenja procesa zadanog izraza. S desne strane relacije redukcije nalazi se procesni izraz dobiven izvršavanjem procesnog izraza koji je naveden na lijevoj strani relacije redukcije. Primjenom relacije redukcije moguće je ostvariti pretvorbu procesnog izraza 3.1 u procesne izraze 3.2 ili 3.3.

$$\bar{x}[a]|\bar{x}[b]|x(u).\bar{y}[u] \rightarrow \bar{x}[b]|\bar{y}[a] \quad (3.2)$$

$$\bar{x}[a]|\bar{x}[b]|x(u).\bar{y}[u] \rightarrow \bar{x}[a]|\bar{y}[b] \quad (3.3)$$

Procesni izrazi 3.2 i 3.3 opisuju dva moguća slijeda izvođenja procesa u procesnom izrazu 3.1. Izraz 3.2 opisuje slijed izvođenja sustava procesa u kojem je prvi proces prvi upisao vrijednost pridruženu varijabli a u komunikacijski kanal x . Obzirom da je prvi proces uspješno završio s izvođenjem svojih naredbi, on se ne pojavljuje u procesnom izrazu 3.2 s desne strane relacije redukcije. Treći proces uspješno čita vrijednost upisanu u komunikacijski kanal x i pokušava upisati pročitanu vrijednost u kanal y . Obzirom algebra π omogućava modeliranje samo sinkronog obrasca komunikacije, a u procesnom izrazu 3.1 ne postoji niti jedan proces koji čita kanal y , treći proces čeka dok neki proces ne pokuša pročitati kanal y . Drugi proces ne uspijeva upisati vrijednost varijable b u kanal x zbog toga što nitko ne pokušava čitati vrijednost u kanalu x . Procesni izraz 3.3 opisuje drugi mogući slijed izvođenja procesa u procesnom izrazu 3.1 u kojem drugi proces prvi upisuje vrijednost varijable b u kanal x . U tom slučaju prvi proces se ne izvodi, dok treći proces pokušava upisati vrijednost varijable b u komunikacijski kanal y .

Algebra π moguće je primijeniti za izgradnju modela složenih usluga. Modeli složenih usluga ostvareni primjenom algebre π grade se opisivanjem ponašajnih svojstava svake od usluga koja čini složenu uslugu primjenom zasebnog procesnog izraza. Ukupni model složene usluge gradi se grupiranjem izgrađenih procesnih izraza u složeni procesni izraz primjenom operatora za istovremeno izvođenje.



Slika 25: Prikaz modela složene usluge za odabir automobila ostvarenog primjenom algebре π

Slika 25 prikazuje primjer modela složene usluge za odabir automobila čiji je tijek izvođenja prikazan na slici 18. Model složene usluge prikazuje usluge *Početak*, *Poslodavac*, *Banka*, *Automobili*, *Kraj* i *Korisnik* koje međusobno komuniciraju putem komunikacijskih kanala a , b , c , d , e , f i g . Ponašajna svojstava usluga koje čine složenu uslugu opisana su sljedećim procesnim izrazima:

$$\begin{aligned}
 K &= \bar{a}[zahtjev].g(moguciAutomobili) \\
 S &= a(zahtjev).\bar{b}[zahtjev] | \bar{c}[zahtjev] \\
 P &= b(zahtjev).\bar{d}[projekPlaca] \\
 B &= d(projekPlaca).\bar{e}[moguciKrediti] \\
 A &= c(zahtjev).\bar{f}[ponudaAutomobila] \\
 E &= (e(moguciKrediti) | f(ponudaAutomobila)).\bar{g}[moguciAutomobili]
 \end{aligned} \tag{3.4}$$

Procesni izraz K opisuje ponašajna svojstva usluge *Korisnik*, procesni izraz S opisuje ponašajna svojstva usluge *Početak*, procesni izraz P opisuje ponašajna svojstva usluge *Poslodavac*, procesni izraz B opisuje ponašajna svojstva usluge *Banka*, procesni izraz A opisuje ponašajna svojstva usluge *Automobili*, procesni izraz E opisuje ponašajna svojstva usluge *Kraj* i procesni izraz K opisuje ponašajna svojstva usluge *Korisnik*. Proces K zapisuje *zahtjev* u komunikacijski kanal a i čeka primitak odgovora u kanalu g . Zahtjev sadrži informacije o korisniku, dok odgovor sadrži popis automobila koje korisnik može kupiti korištenjem kredita. Proces S čita zahtjev zapisan u kanalu a i istovremeno ga prosljeđuje u kanale b i c . Proces P čita zahtjev putem kanala b i korištenjem primljenog zahtjeva određuje projek plaća korisnika, te ga usmjerava procesu B putem kanala d . Proces B čita projek plaća putem kanala d , korištenjem primljenog projekta plaća određuje moguće kredite za korisnika i prosljeđuje projek plaća procesu E . Proces A čita zahtjev primljen od procesa S

putem kanala c , određuje automobile koji su trenutno dostupni na tržištu i prosljeđuje rezultat procesu E putem kanala f . Proces E čita moguće kredite primljene putem kanala e i trenutnu ponudu automobila primljenu putem kanala f . Nadalje, korištenjem kreditnih mogućnosti i trenutne ponude na tržištu automobila, proces E određuje automobile koje korisnik može kupiti korištenjem kredita i prosljeđuje rezultat korisniku putem kanala g . Složeni procesni izraz U koji opisuje složenu uslugu i korisnika ostvaruje se grupiranjem procesnih izraza K, S, P, B, A i E primjenom operatora za istovremeno izvođenje:

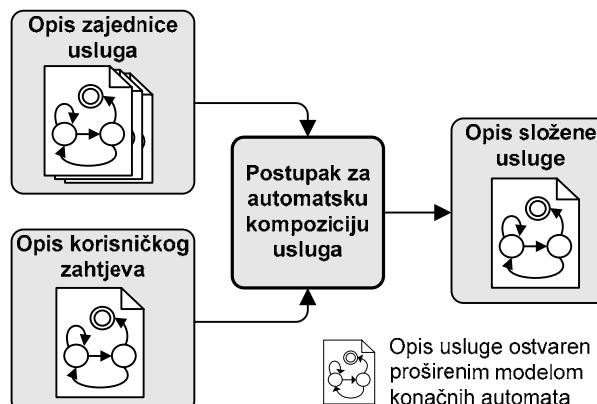
$$U = K | S | P | B | A | E \quad (3.5)$$

Osim opisanog osnovnog modela algebri π , postoje različiti prošireni modeli algebri koji omogućavaju modeliranje asinkronog obrasca komunikacije, primjenu složenih tipova podataka i pokretljivost procesa. Osnovne prednosti korištenja procesnih algebri su mogućnost analize složenih usluga, mogućnost opisivanja struktura podataka i sažeti zapis opisa složene usluge. Modele složenih usluga ostvarene primjenom procesnih algebri moguće je analizirati u svrhu ispitivanja ponašajnih svojstava složenih usluga. Postupci analize provode se primjenom pravila za pretvorbu procesnih izraza koje je moguće provoditi u potpunosti automatizirano primjenom računala. Postupci analize najčešće se provode s ciljem određivanja različitih ponašajnih svojstava složenih usluga kao što su dosežljivosti, potpuni zastoj, neškodljivost i sigurno zaustavljanje [90]. Osim za opisivanje tijeka izvođenja složenih usluga, procesne algebri omogućavaju opisivanje struktura podataka koje procesi međusobno razmjenjuju. Strukture podatka opisuju se primjenom tipova (engl. *types*) [91]. Zrnatost i detaljnost opisa tipova podatka može biti proizvoljna i ovisi o svojstvima koja se analiziraju primjenom izgrađenog modela. Opisi složenih usluga ostvareni primjenom procesnih algebri sažeti su i sadrže nužne semantičke informacije potrebne za opisivanje različitih ponašajnih svojstava složenih usluga.

Osnovni nedostatci primjene procesnih algebri u svrhu kompozicije usluga su složenost postupaka analize i nemogućnost opisivanja nefunkcijskih svojstava složenih usluga. Postupci analize modela složenih usluga ostvarenih procesnim algebrama iznimno su složeni i zahtijevaju značajna računalna sredstva. Nadalje, uspješnost praktične primjene postupaka za analizu modela složenih usluga značajno ovisi o složenosti i stupnju detaljnosti izgrađenog modela složene usluge. Primjenom procesnih algebri nije moguće ostvariti opise nefunkcijskih značajki kao što su vrijeme odziva, trajanje obrade i propusnost složene usluge.

3.2.5 Konačni automati

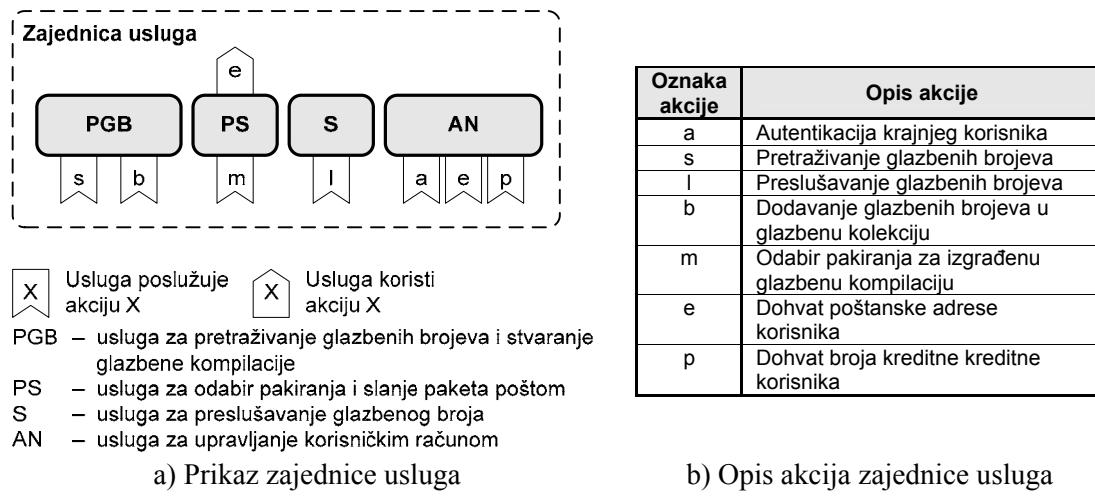
Automatsku kompoziciju usluga moguće je ostvariti primjenom formalnog radnog okruženja (engl. *formal framework*) zasnovanog na proširenom modelu konačnih automata [92]. Prošireni model konačnih automata koristi se za opisivanje ponašajnih svojstava usluga. Opisi ponašajnih svojstava usluga ne sadrže pojedinosti programskog ostvarenja usluga, već samo opisuju akcije koje usluge provode tijekom izvođenja. Usluge opisane primjenom modela zasnovanog na konačnim automatima svrstane su u tri skupine: poslužiteljske usluge, korisničke usluge i hibridne usluge. Poslužiteljske usluge (engl. *pure-servant services*) ne koriste usluge vanjskih udaljenih usluga tijekom izvođenja akcija koje poslužuju. Sve akcije koje poslužiteljska usluga poslužuje izvodi samostalno na računalu domaćinu. Korisničke usluge (engl. *pure-initiator services*) za izvođenje svih akcija koje poslužuju koriste isključivo udaljene vanjske usluge. Mješovite usluge (engl. *mixed-servant services*) dio akcija koje poslužuju ostvaruju samostalno na računalu domaćinu, dok za izvođenje preostalih akcija koriste vanjske udaljene usluge.



Slika 26: Elementi formalnog radnog okruženja za automatsku kompoziciju usluga zasnovanu na proširenom modelu konačnih automata

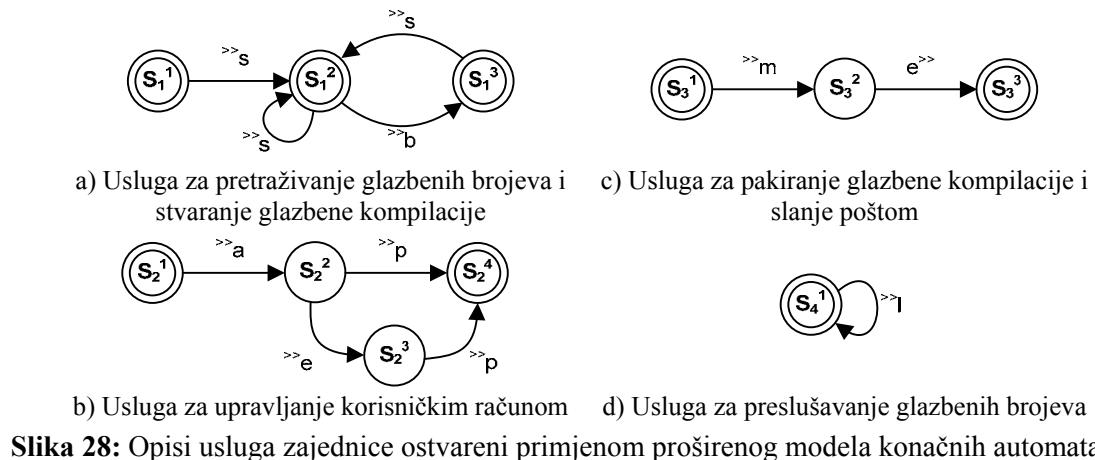
Slika 26 prikazuje elemente formalnog radnog okruženja za automatsku kompoziciju usluga zasnovanu na proširenom modelu konačnih automata. Model uključuje opis zajednice usluga, opis korisničkog zahtjeva, postupak za automatsku kompoziciju usluga i opis složene usluge. Zajednicu usluga (engl. *community of services*) čini skup usluga koje ostvaruju različite funkcionalnosti. Ponašajna svojstva svake od usluga unutar zajednice opisana su primjenom zasebnog konačnog automata. Korisnički zahtjev opisuje ponašajna svojstva usluge koja ostvaruje funkcionalnosti koje želi koristiti korisnik. Postupak za automatsku kompoziciju usluga koristi opise usluga u zajednici usluga i opis korisničkog zahtjeva kako bi izgradio složenu uslugu koja ispunjava korisničke zahtjeve. Rezultat postupka za

automatsku kompoziciju je opis izgrađene složene usluge ostvaren primjenom proširenog modela konačnih automata.



Slika 27: Primjer zajednice usluga

Slika 27 predstavlja primjer zajednice usluga. Slika 27a) prikazuje usluge u odabranoj zajednici usluga. Prikazana zajednica usluga uključuje uslugu za pretraživanje glazbenih brojeva i stvaranje glazbene kompilacije, uslugu za odabir pakiranja i slanje paketa poštom, uslugu za preslušavanje glazbenog broja i uslugu za upravljanje korisničkim računom. Slika 27b) opisuje akcije koje koriste ili poslužuju usluge u prikazanoj zajednici usluga. Usluge za pretraživanje glazbenih brojeva i stvaranje glazbene kompilacije, usluga za preslušavanje glazbenih brojeva i usluga za upravljanje korisničkim računom su poslužiteljske usluge obzirom da ne koriste akcije udaljenih usluga. Usluga za odabir pakiranja i slanje paketa je mješovita usluga obzirom da poslužuje akcije za odabir pakiranja i koristi akciju za dohvatanje adrese korisnika. Usluge u opisanoj zajednici usluga omogućavaju automatsku kompoziciju složene usluge za izgradnju i kupovinu glazbenih kompilacija.



Slika 28: Opisi usluga zajednice ostvareni primjenom proširenog modela konačnih automata

Slika 28 prikazuje opise ponašajnih svojstava usluga u odabranoj zajednici usluga ostvarene primjenom proširenog modela konačnih automata. Prijelazi konačnog automata predstavljaju izvršavanje akcije koju usluga poslužuje ili koristi tijekom izvođenja. Ako usluga poslužuje akciju a , tada prijelaz sadrži oznaku $\gg a$. U slučaju da usluga zahtjeva korištenje udaljene usluge za izvođenje akcije a prijelaz sadrži oznaku $a \gg$. Stanja konačnog automata predstavljaju različita stanja kroz koja usluga prolazi tijekom izvođenja.

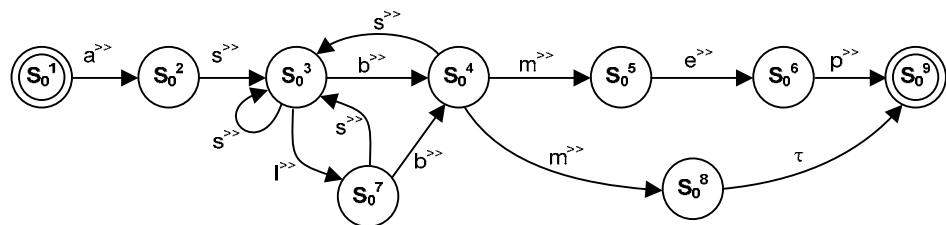
Slika 28a) prikazuje konačni automat koji opisuje ponašajna svojstva usluge za pretraživanje glazbenih brojeva i stvaranje glazbene kompilacije. U početnom stanju S_1^1 usluga omogućava pretraživanje baze glazbenih brojeva izvođenjem akcije s . Izvođenjem akcije s usluga prelazi u stanje S_1^2 u kojem je akciju s moguće provoditi proizvoljan broj puta. U stanju S_1^2 također je moguće izvođenje akcije b koja omogućava odabir skupa glazbenih brojeva za izgradnju vlastite glazbene kompilacije. Nakon izgradnje glazbene kompilacije usluga, usluga prelazi u stanje S_1^3 . U stanju S_1^3 moguće je započeti postupak izgradnje nove glazbene kompilacije izvođenjem akcije s .

Slika 28b) prikazuje konačni automat koji opisuje ponašajna svojstva usluge za upravljanje korisničkim računom. U početnom stanju S_2^1 usluga omogućava autentikaciju korisnika izvođenjem akcije a . Nakon uspješne autentikacije korisnika usluga se nalazi u stanju S_2^2 u kojem može završiti izvođenje na dva moguća načina. Ako je potrebno ostvariti samo naplatu na računu korisnika, izvodi se akcija p i usluga završava izvođenje u konačnom stanju S_2^4 . U slučaju da je potrebno doznati kućnu adresu korisnika prije naplate, izvodi se akcija e i usluga prelazi u stanje S_2^3 . Nakon dohvata kućne adrese, moguće je izvesti akciju p kako bi se ostvarila naplata na računu korisnika, nakon čega usluga prelazi u završno stanje S_2^4 .

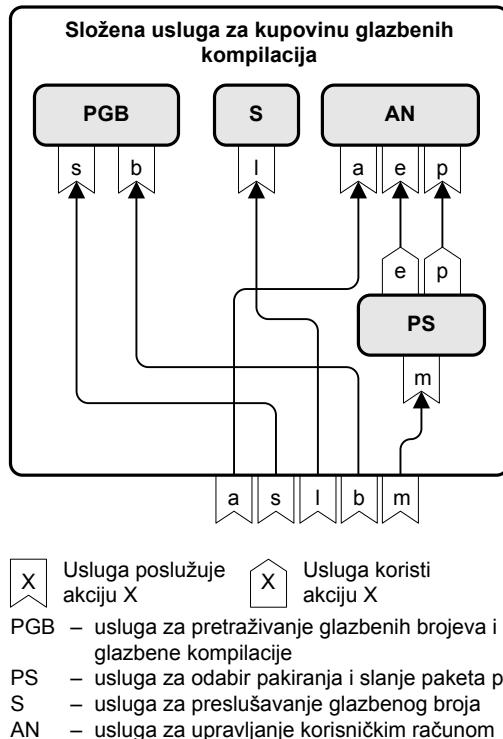
Slika 27c) prikazuje konačni automat koji opisuje ponašajna svojstva usluge za pakiranje i slanje paketa poštom. U početnom stanju S_3^1 usluga omogućava odabir vrste pakiranja za paket izvođenjem akcije m . Nakon odabira vrste pakiranja, usluga prelazi u stanje S_3^2 u kojem zahtjeva korištenje akcije e udaljene usluge kako bi dohvatila informaciju o kućnoj adresi na koju je potrebno poslati paket. Nakon izvođenja akcije e , usluga prelazi u konačno stanje S_3^3 .

Slika 27d) prikazuje konačni automat koji opisuje ponašajna svojstva usluge za preslušavanje glazbenih brojeva. Usluga u početnom stanju S_4^1 omogućava preslušavanje proizvoljnog broja glazbenih brojeva izvođenjem akcije l .

Primjenom prikazanih konačnih automata koji opisuju ponašajna svojstva usluga u zajednici usluga moguće je ostvariti automatsku kompoziciju usluge za izgradnju i kupovinu glazbenih kompilacija. Kako bi se ostvarila automatska kompozicija usluge za izgradnju i kupovinu glazbenih kompilacija, potrebno je izgraditi odgovarajući korisnički zahtjev koji opisuje traženu složenu uslugu. Korisnički zahtjev opisuje ponašajna svojstava usluge za kupovinu glazbenih kompilacija koju je potrebno ostvariti automatskom kompozicijom usluga dostupnih u zajednici usluga. Opis korisničkog zahtjeva ostvaren je na sličan način kao i opisi ponašajnih svojstava usluga u zajednici, uz primjenu tri dodatna svojstva. Prvo svojstvo nalaže da složena usluga koju zahtjeva korisnik ne ostvaruje niti jednu akciju samostalno već za svaku akciju koristi neku od udaljenih usluga koja je dostupna u odabranoj zajednici usluga. Prema tom svojstvu izgrađeni automat sadrži samo grane koje imaju oznaku oblika $a^{>>}$, pri čemu je a akcija čije izvođenje zahtjeva složena usluga. Drugo svojstvo omogućava korištenje nedeterminističkih prijelaza za ostvarivanje opisa korisničke složene usluge. Nedeterministički prijelazi omogućavaju odabir jednog od više mogućih načina kompozicije usluge tijekom provođenja postupka automatske kompozicije. Treće svojstvo omogućava korištenje nedefiniranih prijelaza koji su naznačeni oznakom τ . Tijekom provođenja postupka automatske kompozicije, svaki nedefinirani prijelaz moguće je zamijeniti proizvoljnim nizom akcija koje izvršavaju usluge dostupne u zajednici usluga kako bi ostvarile funkcionalnosti složene usluge. Slika 29 prikazuje korisnički zahtjev koji primjenom konačnog automata opisuje ponašajna svojstava složene usluge za kupovinu glazbenih kompilacija.

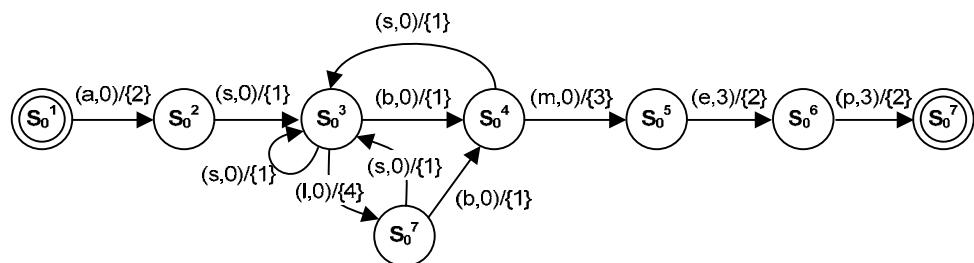


Slika 29: Korisnički zahtjev za složenu uslugu za izgradnju i kupovinu glazbenih kompilacija



Slika 30: Složena usluga za izgradnju i kupovinu glazbenih kompilacija

Slika 30 prikazuje automatski izgrađenu složenu uslugu za kupovinu glazbenih kompilacija koja odgovara korisničkom zahtjevu priказанom na slici 29. Izgrađena složena usluga koristi uslugu *AN* za provođenje postupka autentikacije korisnika. Nakon uspješne autentikacije, složena usluga koristi usluge *PGB* i *S* kako bi omogućila korisniku pretraživanje glazbenih brojeva, preslušavanje glazbenih brojeva i izgradnju glazbenih kompilacija. Izgrađena složena usluga ostvarena je primjenom grane koja sadrži nedefinirani prijelaz konačnog automata koji opisuje korisnički zahtjev priказанog na slici 29. Složena usluga koristi uslugu *PS*, kako bi omogućila korisniku odabir vrste pakiranja za izgrađenu kompilaciju. Nadalje, usluga *PS* koristi uslugu *AN* kako bi odredila poštansku adresu korisnika. Usluga *PS* također koristi uslugu *AN* kako naplatila cijenu izgrađene kompilacije i poštanskih troškova dostave korisniku.



Slika 31: Mealyev konačni automat koji opisuje ponašajna svojstva automatski izgrađene složene usluge za kupovinu glazbenih kompilacija

Slika 31 prikazuje konačni automat koji opisuje automatski izgrađenu složenu uslugu prikazanu na slici 30. Rezultat postupaka za automatsku kompoziciju usluga je Mealyev konačni automat koji opisuje ponašajna svojstva izgrađene složene usluge. U slučaju da automat koji opisuje korisnički zahtjev sadrži nedeterminističke prijelaze ili nedefinirane prijelaze, rezultat postupka je skup Mealyevih konačnih automata od kojih svaki opisuje jednu od mogućih složenih usluga koja ispunjava korisnički zahtjev. Stanja izgrađenih Mealyevih automata predstavljaju stanja kroz koja složena usluga prolazi tijekom izvođenja. Prijelazi izgrađenog Mealyevog automata zapisani su u obliku $(a,i)/\{j\}$, pri čemu a predstavlja akciju koju složena usluga zahtjeva tijekom izvođenja, i predstavlja indeks usluge koja zahtjeva izvršavanje akcije a i j predstavlja indeks usluga koja je poslužitelj za akciju a . Indeksi usluge 0 predstavljaju složenu uslugu, dok ostali indeksi predstavljaju usluge dostupne u odabranoj zajednici usluga.

Primjena metode kompozicije usluga zasnovane na konačnim automatima ima nekoliko prednosti i nedostataka. Metoda kompozicije usluga zasnovana na proširenom modelu konačnih automata omogućava dinamičku kompoziciju usluga i nepotpuno definirane korisničke zahtjeve za kompoziciju usluga. Osnovni nedostatci primjene metode kompozicije zasnovane na modelu konačnih automata su iznimna složenost postupka za automatsku kompoziciju usluga, nemogućnost opisivanja struktura podatka i nemogućnost opisivanja nefunkcijskih značajki složenih usluga.

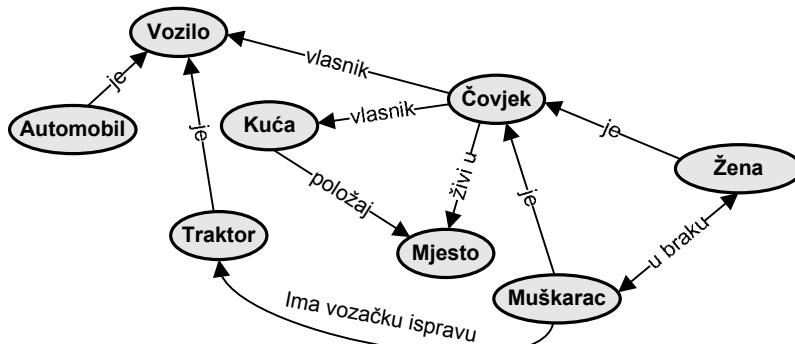
3.2.6 Semantičke usluge

Globalna mreža Internet prvo bitno je ostvarena kako bi se omogućila razmjena, objavljivanje i pretraživanje podataka razumljivih isključivo čovjeku. Pregledavanjem sadržaja stranica i poznavanjem načela definiranih u određenoj ljudskoj djelatnosti, čovjek je u stanju tumačiti objavljene podatke i donositi zaključke o njihovim međusobnim zavisnostima. Obzirom da globalna mreža Internet sadrži veliku količinu podataka koja se svakim danom povećava, čovjeku postaje sve teže pronaći i analizirati tražene podatke. Kako bi se omogućilo učinkovito pretraživanje i analiza podataka objavljenih na Internetu, potrebno je ostvariti automatiziranu obradu podatka primjenom računala. Automatiziranu obradu podatka moguće je ostvariti proširivanjem objavljenih podataka s meta-podacima (engl. *metadata*) [93]. Meta-podaci se koriste za opisivanje značenja podataka i međuzavisnosti različitih vrsta podataka.

Rezultati najnovijih istraživanja u području predstavljanja i otkrivanja znanja (engl. *knowledge representation and discovery*) [94] omogućavaju izgradnju sustava za djelomičnu

i potpunu strojnu analizu podataka objavljenih na globalnoj mreži Internet. Opisivanjem značenja objavljenih podataka, globalna mreža Internet postupno se razvija u novu globalnu mrežu zasnovanu na značenju podataka (engl. *Semantic Web*) [95]. Opisi značenja podataka objavljenih na Internetu ostvaruju se primjenom standardnih jezika. Najšire prihvaćeni standardni jezik za opis značenja podataka je jezik *RDF* (*Resource Description Framework*) [96]. Jezik *RDF* je standardni jezik koji omogućava opisivanje značajki i odnosa između sredstava objavljenih na globalnoj mreži Internet primjenom ontologija.

Ontologija (engl. *ontology*) je formalni zapis skupa koncepata i njihovih međusobnih zavisnosti u određenoj domeni primjene [97]. Ontologije omogućavaju predstavljanje znanja u obliku koji je pogodan za automatsku strojnu obradu i analizu podataka u svrhu provođenja automatskog strojnog zaključivanja. Područje primjene ontologija vrlo je široko i uključuje primjene kao što su objedinjavanje zasnovano na podacima (engl. *data-driven integration*), kooperativni informacijski sustavi (engl. *cooperative information systems*), analiza podataka (engl. *data analysis*), elektroničko poduzetništvo (engl. *electronic commerce*) i upravljanje znanjem (engl. *knowledge management*).



Slika 32: Isječak ontologije koja opisuje koncepte u životu čovjeka

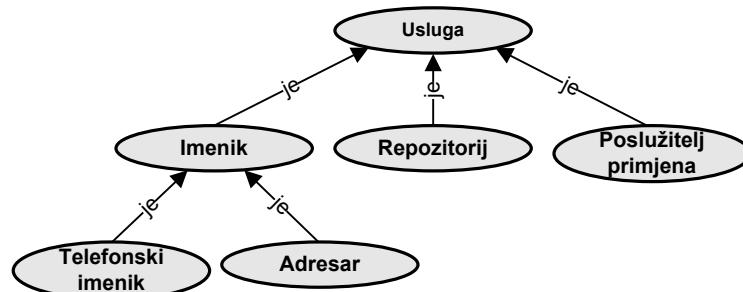
Slika 32 prikazuje isječak ontologije koja opisuje koncepte koji se pojavljuju u životu čovjeka. Prikazani isječak ontologije definira koncepte *Čovjek*, *Mjesto*, *Muškarac*, *Žena*, *Kuća*, *Vozilo*, *Traktor* i *Automobil*. Osim koncepata, ontologija uključuje i relacije koje definiraju pravila koja vrijede za koncepte ontologije. Relacije prikazane ontologije definiraju pravila kao što su *svaka žena je čovjek*, *čovjek može biti vlasnik kuće s položajem na određenom mjestu*, *muškarac može imati vozačku ispravu za upravljanje traktorom i traktor je vrsta vozila*.

Primjena ontologija omogućava izgradnju semantičkih usluga (engl. *Semantic services*) [98]. Semantičke usluge ostvaruju se proširivanjem opisa usluge s informacijama

koje definiraju svojstva, uvjete korištenja, učinke korištenja i ostale značajke usluge. Opisi semantičkih usluga ostvaruju se primjenom formalnih jezika među kojima je najšire prihvaćeni standardni jezik *OWL-S* (*Web Ontology Language for Services*) [99]. Primjena jezika *OWL-S* omogućava automatsko otkrivanje, pozivanje, kompoziciju i usklađivanje izvođenja usluga [100]. Jezik *OWL-S* nastao je specijalizacijom jezika *OWL* (*Web Ontology Language*) [101], pomoću kojeg su definirane standardne ontologije za opisivanje značajki semantičkih usluga. Jezik *OWL-S* je definiran primjenom tri osnovne ontologije koje omogućavaju opisivanje značajki semantičkih usluga: *profile*, *process model* i *grounding*.

Ontologija *profile* primjenjuje se za izgradnju opisa usluge kojim su određene mogućnosti i svrha semantičke usluge. Opis ostvaren korištenjem *profile* ontologije spremi se u imenik usluga tijekom oglašavanja usluge. Korisnici usluga koji zahtijevaju određenu uslugu također koriste ontologiju *profile* kako bi izgradili upit koji opisuje značajke tražene usluge. Imenik usluga korištenjem opisa objavljenih usluga i korisničkog upita automatski pretražuje objavljene usluge i odabire odgovarajućih usluga naziva se postupak uparivanja usluge (engl. *service matchmaking*) [102]. Opise usluga ostvarene primjenom ontologije *profile* moguće je izgraditi opisivanjem funkcionalnih (engl. *functional properties*) i nefunkcionalnih značajki (engl. *non-functional properties*) usluga.

Opisi funkcionalnih značajki usluge zasnivaju se na opisima *ulaza*, *izlaza*, *preduvjeta* i *učinaka* (engl. *Inputs-Outputs-Preconditions-Effects*, *IOPE*). Opis *ulaza* definira strukturu i značenje podataka koje usluga zahtijeva na ulazu prije početka izvođenja. Opis *izlaza* definira strukturu i značenje podataka koje usluga vraća nakon izvođenja. Opis *preduvjeta* definira uvjete koji moraju biti ispunjeni kako bi bilo moguće koristiti uslugu. Opis *učinaka* definira činjenice koje postaju važeće nakon uspješnog završetka izvođenja usluge.



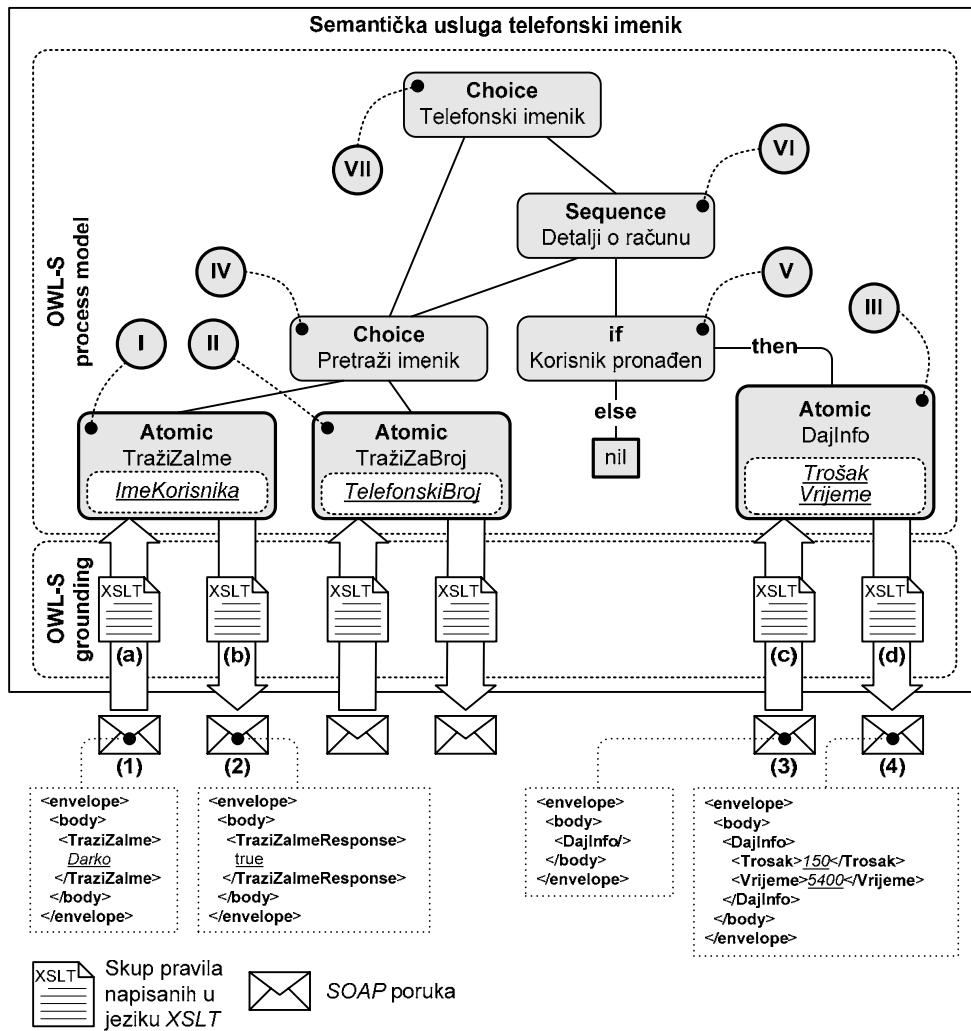
Slika 33: Primjer ontologije za razredbu usluga

Opis nefunkcionalnih značajki ostvaruje se primjenom ontologije za razredbu usluga prema funkcionalnostima koje usluge ostvaruju. Slika 33 prikazuje primjer ontologije

pomoću koje je ostvarena razredba usluge telefonskog imenika. Prema prikazanoj razredbi, usluga telefonskog imenika pripada kategoriji usluga koje ostvaruju imenike.

Ontologija *process model* omogućava opisivanje ponašajnih svojstava usluge. Ponašajna svojstva usluge opisuju se primjenom nedjeljivih (engl. *atomic*), jednostavnih (engl. *simple*) i složenih (engl. *composite*) procesa. Nedjeljni proces opisuje primitak ulaznih i prosljeđivanje izlaznih podataka za jednu od operacija usluge. Jednostavni proces služi za predstavljanje niza akcija koje usluga izvršava tijekom izvođenja. Složeni procesi služe za izgradnju novih procesa kompozicijom jednostavnih i nedjeljivih procesa. Opis složenih procesa ostvaruje se primjenom skupa naredbi koje omogućavaju definiranje tijeka izvođenja akcija usluge. *OWL-S* definira naredbe: *sequence*, *split*, *join*, *unordered*, *choice*, *if-then-else*, *iterate* i *repeat-until*. Naredba *sequence* omogućava slijedno izvođenje niza naredbi. Naredba *split* omogućava grananje tijeka izvođenja u dva ili više istovremena tijeka izvođenja. Naredba *join* definira sinkronizaciju dva ili više istovremenih tijekova izvođenja u jedan tijek izvođenja. Naredba *unordered* definira neuređeno istovremeno izvođenje niza naredbi s međusobnim isključivanjem. Naredba *choice* služi za izvođenje jednog od skupa alternativnih tijekova izvođenja. Naredba *if-then-else* služi za uvjetno grananje slijeda izvođenja. Naredba *iterate* služi za ostvarivanje uvjetnog ponavljanja slijeda naredbi s ispitivanjem uvjeta na početku petlje. Naredba *repeat-until* služi za ostvarivanje uvjetnog ponavljanja niza naredbi s ispitivanjem uvjeta ponavljanja na kraju petlje.

Ontologija *grounding* primjenjuje se za opisivanje sadržaja poruka koje semantička usluga koristi tijekom komunikacije usluge s udaljenim uslugama i korisnicima. Ontologija *grounding* zasniva se na primjeni niza pravila za preslikavanje vrijednosti značajki koje su pridružene konceptima *process model* ontologije u sadržaj poruka semantičke usluge. Nadalje, ontologija *grounding* oslanja se na opise *SOAP* poruka koje usluga razmjenjuje tijekom rada i koji su opisani u *WSDL* dokumentu usluge.



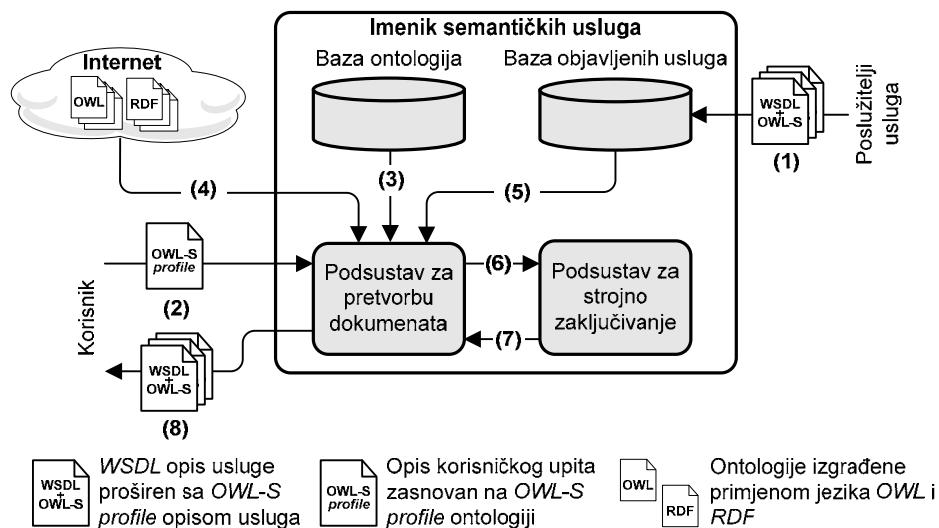
Slika 34: Primjer korištenja *process model* i *grounding* ontologija jezika *OWL-S* u svrhu izgradnje semantičke usluge *Telefonski imenik*

Slika 34 prikazuje primjer izgradnje semantičke usluge *Telefonski imenik* ostvarene korištenjem *process model* i *grounding* ontologija jezika *OWL-S*. Pretraživanje *Telefonskog imenika* moguće je ostvariti prema zadanom imenu korisnika pozivom operacije *TražiZaIme* i prema telefonskom broju korisnika pozivom operacije *TražiZaBroj*. U slučaju da je pretraživanje uspješno ostvareno, pozivom operacije *DajInfo* moguće je dohvatiti iscrpljive informacije o korisničkom računu.

Pravila *grounding* ontologije za semantičku uslugu *Telefonski imenik* ostvarena su primjenom jezika *XSLT* [103]. Jezik *XSLT* je standardni jezik koji omogućava definiranje pravila za pretvorbu *XML* dokumenata. Primjena jezika *XSLT* omogućava definiranje strukture *SOAP* poruka zahtjeva i odgovora koje usluga semantička usluga razmjenjuje s korisnicima tijekom poziva njezinih operacija.

Opis ponašajnih svojstava usluge *Telefonskog imenika* ostvaren *process model* ontologijom sadrži tri nedjeljiva procesa koji su pridruženi operacijama *TražiZaIme* (I), *TražiZaBroj* (II) i *DajInfo* (III). Na početku izvođenja moguće je ostvariti poziv operacije *TražiZaIme* ili *TražiZaBroj* što je određeno naredbom **choice** (IV). Nakon izvođenja neke od naredbi za pretraživanje, ako je korisnički račun pronađen moguće je ostvariti poziv operacije *DajInfo* što je određeno **if-then-else** naredbom (V).

Tijekom poziva operacije *TražiZaIme*, *XSLT* pravilo pridruženo nedjeljivom procesu *TražiZaIme* definira da tijelo *SOAP* poruke zahtjeva mora sadržavati *XML* element s imenom *TražiZaIme* koji sadrži ime korisnika (1). Primljeno ime preslikava se primjenom *XSLT* pravila (a) u atribut *ImeKorisnika* koji sadrži nedjeljivi proces *TražiZaIme*. Nakon definiranja atributa, usluga primjenom *XSLT* pravila (b) gradi poruku odgovora (2). U slučaju da je pretraživanje imenika uspješno, poziv operacije *DajInfo* moguće je ostvariti upućivanjem odgovarajuće *SOAP* poruke zahtjeva koja ispunjava *XSLT* pravilo (c) pridruženo nedjeljivom procesu *DajInfo* (3). Nakon dohvatanja podatka o korisničkom računu, *SOAP* poruka odgovora gradi se primjenom *XSLT* pravila (d) pridruženog nedjeljivom procesu *DajInfo* (4). Poruka odgovora sadrži ukupan broj utrošenih novčanih jedinica i vremensko razdoblje korištenja računa.



Slika 35: Arhitektura imenika semantičkih usluga

Primjena *profile*, *process model* i *grounding* ontologija jezika OWL-S omogućava automatsko pretraživanje, odabir, uskladjivanje i kompoziciju semantičkih usluga. Automatsko pretraživanje i odabir usluga provodi se primjenom imenika semantičkih usluga čija je arhitektura prikazana na slici 35. Imenik semantičkih usluga sadrži podsustav za pretvorbu dokumenata, podsustav za strojno zaključivanje, bazu ontologija i bazu

objavljenih usluga. *WSDL* i *OWL-S* dokumenti koji opisuju značajke dostupnih usluga objavljaju se u bazi objavljenih usluga (1). Korisnik koji zahtjeva uslugu određenih značajki upućuje korisnički zahtjev imeniku usluga (2). Korisnički zahtjev sadrži *OWL-S profile* dokument koji opisuje značajke tražene usluge. Podsustav za pretvorbu dokumenata prihvata korisnički zahtjev, dohvaca ontologije koje su lokalno spremljene u bazi ontologija (3), dohvaca ontologije dostupne na globalnoj mreži Internet koje koristi korisnički zahtjev (4) i *OWL-S profile* dokumente dostupne u bazi objavljenih usluga (5). Po primitku svih potrebnih dokumenata, podsustav za pretvorbu dokumenata pretvara primljene dokumente u zapis pogodan za provođenje strojnog zaključivanja i prosljeđuje rezultat pretvorbe podsustavu za strojno zaključivanje (6). Podsustav za strojno zaključivanje provodi analizu primljenih podataka, provodi postupak uparivanja korisničkog zahtjeva s opisima objavljenih usluga i prosljeđuje rezultat analize podsustavu za pretvorbu dokumenata (7). Korištenjem rezultata strojnog zaključivanja, podsustav za pretvorbu dokumenata odabire i korisniku prosljeđuje opise usluga koje najbolje ispunjavaju korisničke zahtjeve (8).

Automatsko usklađivanje usluga ostvaruje se primjenom opisa zasnovanih na ontologijama *process model* i *grounding*. Ontologije *process model* i *grounding* omogućavaju uslugama da samostalno ostvaruju međudjelovanje s udaljenim uslugama dostupnim u globalnoj mreži Internet. Usluge ostvaruju međusobnu komunikaciju primjenom različitih obrazaca za razmjenu poruka. Obrasci razmjene poruka i sadržaj poruka nisu zadani tijekom izgradnje usluga, već usluge samostalno provode analizu *process model* i *grounding* opisa udaljenih usluga kako bi ih utvrdili. Na osnovi rezultata analize, usluge samostalno grade poruke s odgovarajućim sadržajem i razmjenjuju ih s udaljenim uslugama odgovarajućim redoslijedom.

Automatska kompozicija usluga ostvaruje se primjenom opisa semantičkih usluga koji su dostupni na Internetu i ostvareni primjenom *profile* i *process model* ontologija. U slučaju da ne postoji usluga koja ispunjava korisničke zahtjeve, koristi se sustav za automatsku kompoziciju usluga koji ostvaruje kompoziciju primjenom postupka planiranja zasnovanog na umjetnoj inteligenciji (engl. *Artificial Intelligence Planning*) [104]. Sustav za planiranje započinje postupak kompozicije od početnog stanja koje je određeno opisima *preduvjeta* u *profile* opisu tražene usluge. Tijekom provodenja postupka planiranja, sustav za planiranje pokušava pronaći mogući način povezivanja dostupnih semantičkih usluga kojim su ostvareni *učinci* opisani u *profile* opisu tražene usluge. U slučaju primjene pojedine odabrane semantičke usluge, sustav uzima u obzir *učinke* njezina izvođenja koji su

specificirani u njezinom *profile* opisu i tako određuje doprinos odabrane usluge u ukupnom učinaku složene usluge koju sustav automatski gradi.

Osnovne prednosti primjene semantičkih usluga su mogućnost automatskog pretraživanja, odabira, usklađivanja i kompozicije usluga. Osim navedenih prednosti, primjena semantičkih usluga ima i sljedeće nedostatke: nepostojanje standardnih ontologija, detaljnost opisa i složenost postupaka strojnog zaključivanja. Osnovni nedostatak primjene ontologija je nemogućnost izgradnje jedinstvene, standardizirane i sveobuhvatne ontologije koja sadrži sve koncepte i relacije koje se pojavljuju u stvarnom životu. Ontologije je moguće primjenjivati za pojedine specijalizirane domene ljudskih djelatnosti za koje je moguće definirati ograničeni skup koncepata i konzistentnih relacija. Međutim, koncepte i odnose koji vrijede u određenim ljudskim djelatnostima nije moguće opisati primjenom jedinstvene ontologije, zbog toga što ih različiti ljudi na različite načine doživljavaju, koriste i vrednuju. Kako bi se omogućila široka primjena ontologija, potrebno je izgraditi standardne ontologije specijalizirane za pojedina područja ljudskih djelatnosti. Tijekom izgradnje ontologija koje opisuju značajke semantičkih usluga potrebno je ostvariti opise odgovarajuće razine detaljnosti. Korištenje nedovoljno preciznih opisa zasnovanih na ontologijama ne omogućava donošenje odgovarajućih zaključaka tijekom provođenja postupaka automatskog pretraživanja, odabira i kompozicije usluga. Nadalje, korištenje previše detaljnih opisa uzrokuje složene opise koje nije moguće praktično koristiti tijekom provođenja automatskog pretraživanja, odabira i kompozicije usluga. Većina navedenih postupaka su vrlo složeni i zahtijevaju značajne količine računalnih sredstava, stoga je njihova praktična primjena ograničena.

4. poglavlje

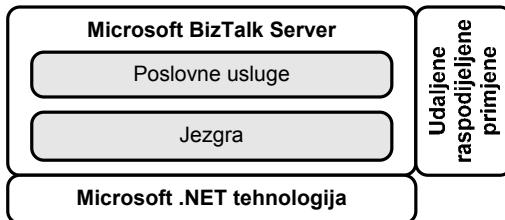
Sustavi za izgradnju raspodijeljenih aplikacija kompozicijom usluga

Raspodijeljene aplikacije zasnovane na uslugama u praktičnoj se primjeni najčešće ostvaruju primjenom orkestracije i koreografije usluga. Postavljanje, izvođenje, nadgledanje i upravljanje raspodijeljenim aplikacijama koje su ostvarene orkestracijom i koreografijom usluga omogućavaju specijalizirani poslužitelji za upravljanje poslovnim procesima (engl. *business process management servers*).

Na tržištu je trenutno dostupno nekoliko komercijalnih poslužitelja za upravljanje poslovnim procesima koje su razvile neke od vodećih *ICT (Information and Communication Technologies)* tvrtki. Najšire prihvaćeni poslužitelji za upravljanje poslovnim procesima su *BizTalk Server* tvrtke *Microsoft* i *BPEL Process Manager* tvrtke *Oracle*. Nadalje, na tržištu su također dostupni i *WebSphere Business Integration Server Foundation* tvrtke *IBM* i *WebLogic Process Edition* tvrtke *BEA*. Uz komercijalno dostupne poslužitelje za upravljanje poslovnim procesima, široko je prihvaćen i *ActiveBPEL* poslužitelj koji je razvijen u sklopu projekta otvorenog kôda (engl. *open source project*). Osim sustava za kompoziciju usluga zasnovanih na poslužiteljima za upravljanje poslovnim procesima, u sklopu različitih znanstveno-istraživačkih projekata razvijeni su raznovrsni sustavi za kompoziciju usluga zasnovanu na opisima procesa. Jedan od najpoznatijih sustava za kompoziciju usluga zasnovanu na procesima koji je razvijen u sklopu znanstveno-istraživačkog projekta je programski sustav *Self-Serv*.

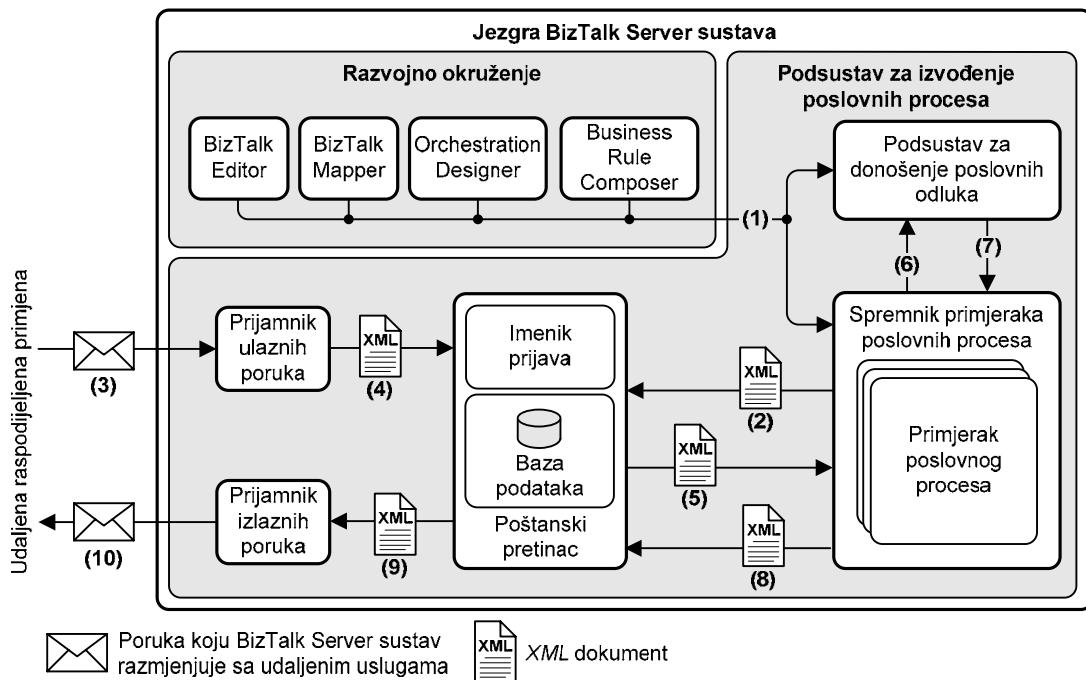
4.1 BizTalk Server

Sustav *BizTalk Server* [80] komercijalno je dostupni poslužitelj za upravljanje poslovnim procesima koji je razvila tvrtka Microsoft. Sustav *BizTalk Server* omogućava razvoj, postavljanje, izvođenje, nadgledanje i upravljanje poslovnim procesima koji se ostvaruju u svrhu objedinjavanja raznovrsnih raspodijeljenih aplikacija. Prema mogućnostima i dostupnoj programskoj potpori, sustav *BizTalk Server* je najsveobuhvatniji poslužitelj za upravljanje poslovnim procesima trenutno dostupan na tržištu.



Slika 36: Vršna arhitektura *BizTalk Server* sustava

Slika 36 prikazuje vršnu arhitekturu *BizTalk Server* sustava koji je zasnovan na *Microsoft .NET* tehnologiji. Sustav *BizTalk Server* sastoji se od jezgre i poslovnih usluga. Jezgra sustava omogućava oblikovanje, analizu, postavljanje i izvođenje poslovnih procesa. Poslovne usluge ostvaruju dodatne funkcionalnosti koje olakšavaju korisnicima *BizTalk Server* sustava nadzor, upravljanje, nadgledanje i analizu aktivnosti koje izvodi poslovni proces.



Slika 37: Arhitektura jezgre *BizTalk Server* sustava

Slika 37 prikazuje arhitekturu jezgre *BizTalk Server* sustava. Jezgra sustava sadrži razvojno okruženje i podsustav za izvođenje poslovnih procesa. Razvojno okruženje jezgre omogućava oblikovanje, analizu i postavljanje poslovnih procesa. Osnovni alati razvojnog okruženja su *BizTalk Editor*, *BizTalk Mapper*, *Orchestration Designer* i *Business Rule Composer*. Navedeni alati zasnovani su na grafičkim korisničkim sučeljima koja olakšavaju rad korisniku *BizTalk Server* sustava. Alat *BizTalk Editor* služi za izgradnju *XML Schema* dokumenata za opisivanje strukture i sadržaja *XML* dokumenata koje obrađuju poslovni

procesi tijekom izvođenja. Alat *BizTalk Mapper* služi za definiranje pravila za pretvorbu sadržaja *XML* dokumenata. Podsustav *Orchestration Designer* služi za oblikovanje, analizu i postavljanje poslovnih procesa koje izvodi podsustav za izvođenje poslovnih procesa. Alat *Business Rule Composer* omogućava definiranje poslovnih pravila koja služe za donošenje poslovnih odluka tijekom izvođenja poslovnih procesa. Podsustav za izvođenje poslovnih procesa ostvaruje infrastrukturu za postavljanje i izvođenje primjeraka različitih poslovnih procesa. Nadalje, podsustav za izvođenje poslovnih procesa sadrži prijamnik ulaznih poruka, prijamnik izlaznih poruka, poštanski pretinac, spremnik primjeraka poslovnih procesa i podsustav za donošenje poslovnih odluka.

Prijamnik ulaznih poruka sprema poruke koje poslužitelj prima od udaljenih raspodijeljenih aplikacija, ostvaruje pretvorbu primljenih poruka u *XML* dokumente i spremi izgrađene *XML* dokumente u poštanski pretinac. Prijamnik izlaznih poruka dohvaća *XML* dokumente spremljene u poštanskom pretincu, ostvaruje njihovu pretvorbu u poruke oblika prikladnog za udaljenu raspodijeljenu aplikaciju i prosljeđuje izgrađene poruke udaljenoj aplikaciji. Korisnicima jezgre *BizTalk Server* sustava omogućeno je razvijanje vlastitih prijamnika poruka koji omogućavaju komunikaciju s udaljenim aplikacijama korištenjem raznovrsnih komunikacijskih protokola. Jezgra *BizTalk Server* sustava standardno uključuje prijamnike za komunikacijske protokole *SOAP*, *IOP*, *JMS* i *RMI*.

Poštanski pretinac sprema ulazne i izlazne *XML* dokumente koje poslovni procesi obrađuju tijekom izvođenja. Poštanski pretinac sadrži imenik prijava i bazu podatka. Imenik prijava služi za prijavljivanje primjeraka poslovnih procesa koji žele primiti određenu vrstu *XML* dokumenata spremljenih u poštanskom pretincu. Zapis u imeniku prijava sadrži definiciju strukture i sadržaja *XML* dokumenta koje određeni primjerak poslovnog procesa želi primiti. Baza podatka služi za spremanje *XML* dokumenata koji se nalaze u poštanskom pretincu.

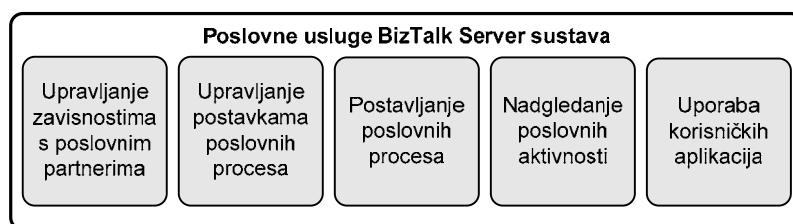
Spremnik primjeraka poslovnih procesa omogućava spremanje stanja i izvođenje primjeraka poslovnih procesa. Spremnik sprema primjerke različitih poslovnih procesa koji se izvode istovremeno i međusobno nezavisno.

Podsustav za donošenje poslovnih odluka omogućava upravljanje tijekom izvođenja poslovnih procesa primjenom zadanih poslovnih ciljeva. Poslovna pravila definirana su korištenjem jednostavnog jezika koji je prilagođen krajnjim korisnicima *BizTalk Server* sustava, kao što su poslovni analitičari (engl. *business analysts*). Pravila određuju akcije koje poslovni proces mora provoditi kako bi ostvario poslovne ciljeve, kao što su odabir usluge s

minimalnom cijenom, minimalnim vremenom odziva i maksimalnom pouzdanošću. Primjena poslovnih pravila povećava prilagodljivost izgrađenih poslovnih procesa, te omogućava krajnjim korisnicima jednostavnu izmjenu ponašanja poslovnih procesa bez poznavanja detalja njihova ostvarenja.

Slika 37 prikazuje primjer akcija koje izvodi jezgra sustava *BizTalk Server* tijekom izvođenja poslovnog procesa. Stručnjak za razvoj poslovnih procesa koristi alate *BizTalk Mapper*, *BizTalk Editor*, *Orchestration Designer* i *Business Rule Composer* kako bi izgradio poslovni proces (1). Spremnik primjeraka poslovnih procesa spremi primjerak novog poslovnog procesa, prijavljuje vrstu *XML* dokumenata koje primjerak poslovnog procesa tijekom izvođenja želi primati iz poštanskog pretinca (2) i započinje izvođenje primjerka procesa. Tijekom izvođenja primjerka poslovnog procesa, prijamnik ulaznih poruka prihvaca ulaznu poruku od udaljene aplikacije (3), ostvaruje pretvorbu primljene poruke u *XML* dokument i spremi izgrađeni *XML* dokument u poštanski pretinac (4). Na osnovi aktivnih prijava u imeniku prijava, poštanski pretinac proslijedi primljeni *XML* dokument odgovarajućem primjerku poslovnog procesa u spremniku (5). Poslovni proces obrađuje primljeni *XML* dokument i usmjerava upit podsustavu za donošenje poslovnih odluka kako bi utvrdio odgovarajuću akciju (6). Podsustav za donošenje poslovnih odluka određuje odgovarajuću akciju poslovnog procesa i proslijedi odluku primjerku poslovnog procesa (7). Na osnovi primljene odluke, primjerak poslovnog procesa stvara izlazni *XML* dokument i spremi ga u poštanski pretinac (8). Poštanski pretinac proslijedi primljeni *XML* dokument prijamniku izlaznih poruka (9). Prijamnik izlaznih poruka pretvara primljeni *XML* dokument u poruku odgovora i dostavlja izgrađenu poruku udaljenoj aplikaciji (10).

Osim jezgre *BizTalk Server* sustava, važnu komponentu sustava čine i poslovne usluge. Poslovne usluge omogućavaju korisnicima *BizTalk Server* sustava učinkovito upravljanje i nadgledanje poslovnih procesa koje izvodi jezgra sustava.



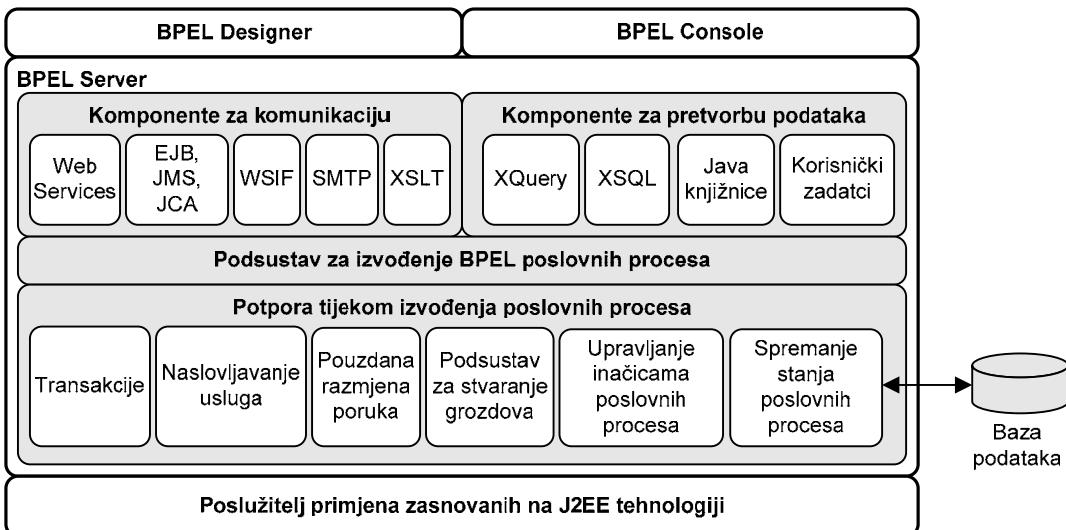
Slika 38: Poslovne usluge *BizTalk Server* sustava

Slika 38 prikazuje poslovne usluge *BizTalk Server* sustava koje omogućavaju upravljanje zavisnostima s poslovnim partnerima, upravljanje postavkama poslovnih

procesa, postavljanje poslovnih procesa, nadgledanje poslovnih aktivnosti i uporabu korisničkih aplikacija. Usluge za upravljanje zavisnostima s poslovnim partnerima (engl. *trading partner management*) olakšavaju upravljanje promjenjivim zahtjevima koje nameću različiti poslovni partneri tijekom provođenja elektroničke poslovne suradnje. Zahtjevi određuju vrstu komunikacijskog protokola, strukture i sadržaj poruka koje poslovni partneri međusobno razmjenjuju, sigurnosne postavke i ostale informacije potrebne za uspješno provođenje elektroničkog poslovanja. Usluge za upravljanje postavkama poslovnih procesa (engl. *business process configuration*) omogućavaju korisnicima *BizTalk Server* sustava definiranje postavki poslovnih procesa. Usluge za postavljanje poslovnih procesa (engl. *business process provisioning*) omogućavaju postavljanje, izvođenje i upravljanje poslovnim procesima na udaljenim računalima. Usluge za nadgledanje poslovnih aktivnosti (engl. *business activity monitoring*) omogućavaju nadgledanje poslovnih aktivnosti koje poslovni procesi provode tijekom izvođenja. Prikupljene informacije koriste se tijekom provođenja analize elektroničke poslovne suradnje s ciljem otkrivanja kritičnih i neučinkovitih aktivnosti, te mogućih poboljšanja u korištenim poslovnim procesima. Usluge za uporabu korisničkih aplikacija (engl. *human workflow services*) omogućavaju objedinjavanje *BizTalk Server* sustava s raznovrsnim korisničkim aplikacijama. Povezivanje korisničkih aplikacija omogućava međudjelovanje krajnjih korisnika i poslovnih procesa. Krajnji korisnici imaju mogućnost zadavanja ulaznih podataka za poslovne procese i prikupljanja rezultata izvođenja poslovnih procesa primjenom različitih aplikacija. Tipične aplikacije koje je moguće objediniti sa sustavom *Microsoft BizTalk Server* su *Microsoft Word*, *Microsoft Outlook* i *Microsoft Excel*.

4.2 BPEL Process Manager

Sustav *BPEL Process Manager* [105] je poslužitelj za upravljanje poslovnim procesima koji je razvila tvrtka Oracle primjenom *J2EE (Java 2 Enterprise Edition)* tehnologije. Poslužitelj je specijaliziran za upravljanje poslovnim procesima zasnovanim na jeziku *BPEL4WS* i ostvaruje nešto manji skup funkcionalnosti u odnosu na *BizTalk Server* sustav tvrtke Microsoft.



Slika 39: Arhitektura BPEL Process Manager sustava

Slika 39 prikazuje arhitekturu sustava *BPEL Process Manager* koji sadrži podsustave *BPEL Designer*, *BPEL Console*, *BPEL Server* i bazu podataka. Podsustav *BPEL Designer* omogućava oblikovanje, analizu i izgradnju *BPEL4WS* poslovnih procesa primjenom grafičkog korisničkog sučelja. Podsustav *BPEL Console* omogućava postavljanje, upravljanje, nadgledanje i ispitivanje izgrađenih *BPEL4WS* poslovnih procesa.

Podsustav *BPEL Server* omogućava postavljanje i izvođenje poslovnih procesa i zahtjeva korištenjem poslužitelja aplikacija zasnovanog na *J2EE* tehnologiji. Podsustav sadrži podsustav za izvođenje poslovnih procesa i tri proširiva skupa komponenata: komponente za komunikaciju, komponente za pretvorbu podataka i komponente za potporu tijekom izvođenja poslovnih procesa.

Komponente za komunikaciju omogućavaju izgradnju poslovnih procesa koji koriste raznovrsne komunikacijske protokole tijekom komunikacije s udaljenim raspodijeljenim aplikacijama. Sustav *BPEL Process Manager* sadrži standardni skup komponenata za komunikaciju koje omogućavaju primjenu tehnologija *Web Services*, *EJB (Enterprise Java Beans)*, *JMS (Java Message Service)*, *JCA (Java Connector Architecture)*, *SMTP (Simple Mail Transfer Protocol)* i *WSIF (Web Service Invocation Framework)*. Navedeni standardni skup komponenata za komunikaciju moguće je proširiti komponentama koje podržavaju različite komunikacijske protokole potrebne za objedinjavanje naslijedenih programskih sustava.

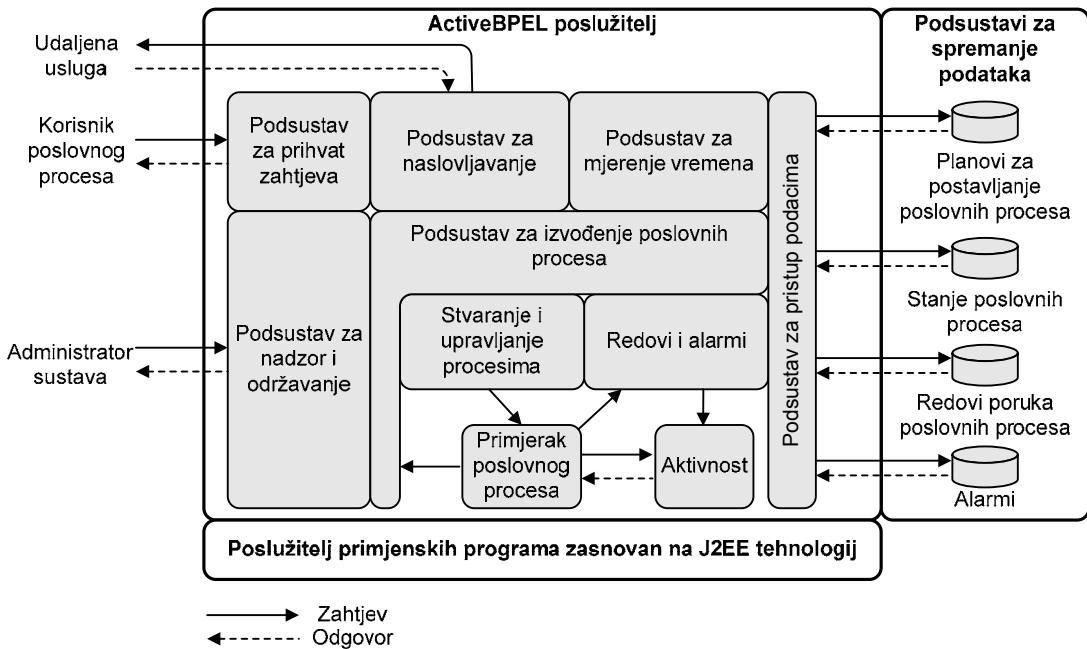
Komponente za pretvorbu podatka omogućavaju napredno pretraživanje i obradu *XML* dokumenata koje poslovni procesi tijekom izvođenja razmjenjuju s udaljenim

aplikacijama. Mogućnost primjene komponenata za pretvorbu podataka iznimno je važna obzirom da jezik *BPEL4WS* ima tek potporu za provođenje jednostavne obrade *XML* dokumenata. Sustav *BPEL Process Manager* uključuje komponente za pretvorbu podataka koje omogućavaju primjenu jezika *XSLT* [103], *XQuery* [106] i *XSQL* [107]. Navedeni jezici omogućavaju izgradnju upita za pretraživanje i obradu *XML* dokumenata. Osim jezika za izgradnju upita, sustav *BPEL Process Manager* omogućava izgradnju općenitih komponenata za pretvorbu podataka primjenom programskih knjižnica i zadataka napisanih u programskom jeziku *Java*.

Podsustav za izvođenje poslovnih procesa omogućava istovremeno izvođenje dva ili više nezavisna poslovna procesa zasnovana na jeziku *BPEL4WS*. Tijekom rada, podsustav za izvođenje poslovnih procesa koristi komponente za potporu tijekom izvođenja poslovnih procesa. Komponente za potporu tijekom izvođenja poslovnih procesa omogućavaju korištenje transakcija, naslovljavanje udaljenih usluga, pouzdanu razmјenu poruka, grupiranje dva ili više *BPEL Process Manager* sustava u grozd, upravljanje inačicama poslovnih procesa i spremanje stanja poslovnih procesa. Stanje poslovnih procesa spremi se u bazu podataka. Za spremanje stanja procesa moguće je koristiti sustave baza podataka kao što su *SQL Server* [108], *DB2* [109] i *Oracle* [110].

4.3 ActiveBPEL

Sustav *ActiveBPEL* [111] je proširivi i prilagodljivi programski sustav za postavljanje i izvođenje poslovnih procesa ostvarenih primjenom jezika *BPEL4WS*. Sustav je razvijen u sklopu projekta otvorenog kôda primjenom *J2EE* tehnologije. Obzirom da je programski kôd sustava javno dostupan, sustav *ActiveBPEL* često se primjenjuje u raznim znanstveno-istraživačkim projektima za izgradnju i ispitivanje svojstava prototipova sustava za kompoziciju usluga zasnovanih na novim arhitekturama.



Slika 40: Arhitektura ActiveBPEL poslužitelja

Slika 40 prikazuje arhitekturu ActiveBPEL poslužitelja i okolinu koja je potrebna za njegovu uporabu. Obzirom da je ActiveBPEL poslužitelj ostvaren primjenom J2EE tehnologije, za njegovo izvođenje potrebno je koristiti poslužitelj aplikacija koji omogućava izvođenje aplikacija zasnovanih na J2EE tehnologiji. Nadalje, kao potporu tijekom izvođenja ActiveBPEL poslužitelja potrebno je koristiti različite podsustave za spremanje podataka. Podsustavi za spremanje podataka spremaju planove za postavljanje poslovnih procesa, stanje poslovnih procesa tijekom izvođenja, redove poruka poslovnih procesa i vrijednosti alarme koje BPEL4WS poslovni proces koristi tijekom izvođenja. Sustav ActiveBPEL sadrži podsustav za prihvatanje zahtjeva, podsustav za nadzor i održavanje, podsustav za naslovljavanje, podsustav za brojanje vremena, podsustav za izvođenje poslovnih procesa, podsustav za stvaranje i upravljanje instancama poslovnih procesa, podsustav za upravljanje redovima i alarmima, te podsustave za izvođenje primjeraka poslovnih procesa i aktivnosti poslovnih procesa.

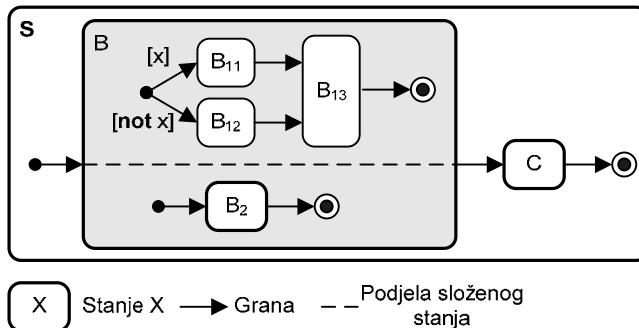
Podsustav za nadzor i održavanje omogućava upraviteljima (engl. *administrators*) sustava upravljanje postavkama sustava, te postavljanje, nadzor i nadgledanje poslovnih procesa. Podsustav za prihvatanje zahtjeva ostvaruje pristupna sučelja putem kojih poslovni procesi koje izvodi ActiveBPEL poslužitelj prihvaćaju poruke zahtjeva i proslijeduju poruke odgovora. Podsustav za naslovljavanje ostvaruje logiku za usmjeravanje i povezivanje poruka zahtjeva i poruka odgovora koje poslovni procesi razmjenjuju s udaljenim uslugama. Podsustav za mjerjenje vremena ostvaruje mjerače vremena koji se primjenjuju za

ostvarivanje alarma koje poslovni procesi koriste tijekom izvođenja. Podsustav za izvođenje poslovnih procesa središnji je podsustav koji upravlja tijekom izvođenja poslovnih procesa i ukupnog poslužitelja. Podsustav za upravljanje pristupa podacima omogućava pristup bazama podataka koje služe za spremanje informacija potrebnih za postavljanje i izvođenje poslovnih procesa. Spremanje podataka moguće je ostvariti u radnom spremniku računala, bazama podataka ili lokalnom diskovnom prostoru računala poslužitelja. Podsustav za stvaranje poslovnih procesa omogućava stvaranje primjeraka poslovnih procesa. Podsustav za redove i alarme ostvaruje logiku za poticanje izvođenja aktivnosti poslovnih procesa po primitku poruka zahtjeva ili isteku zadanog vremenskog razdoblja. Primjeri poslovnog procesa tijekom izvođenja koriste odgovarajuće aktivnosti, te upravljaju redovima poruka i alarmima poslovnog procesa.

4.4 Self-Serv

Sustav *Self-Serv* je dinamični i prilagodljivi programski sustav koji omogućava ubrzanu i učinkovitu izgradnju raspodijeljenih aplikacija sa svojstvom razmijernog rasta [112]. Osnovne značajke programskog sustava *Self-Serv* su primjena grafičkog jezika *statecharts* za izgradnju opisa raspodijeljenih aplikacija, korištenje zajednice usluga (engl. *service community*) za izgradnju raspodijeljenih aplikacija i raspodijeljeno upravljanje tijekom izvođenja raspodijeljenih aplikacija zasnovano na modelu ravnopravnih sudionika (engl. *peer-to-peer orchestration model*).

Jezik *statecharts* je standardni grafički jezik za opisivanje reaktivnih sustava (engl. *reactive systems*) koji je dio jezika *UML (Unified Modeling Language)* [113]. Primjena grafičkog jezika *statecharts* olakšava i ubrzava oblikovanje i izgradnju raspodijeljenih aplikacija zasnovanih na uslugama. Jezik podržava osnovne gradivne elemente za opisivanje tijeka izvođenja raspodijeljenih aplikacija kao što su slijedno izvođenje, ponavljajuće izvođenje, istovremeno grananje tijeka izvođenja i sinkronizacija. Dijagram koji opisuje tijek izvođenja raspodijeljene aplikacije ostvaren u jeziku *statecharts* sastoji se od stanja i grana. Stanja mogu biti osnovna (engl. *basic state*) i složena (engl. *compound state*). Osnovna stanja su nedjeljiva. Složena stanja ostvaruju se grupiranjem osnovnih i složenih stanja. Grane povezuju stanja i modeliraju tijek izvođenja raspodijeljene aplikacije. Grani je moguće pridružiti uvjet koji određuje kada grana postaje aktivna.



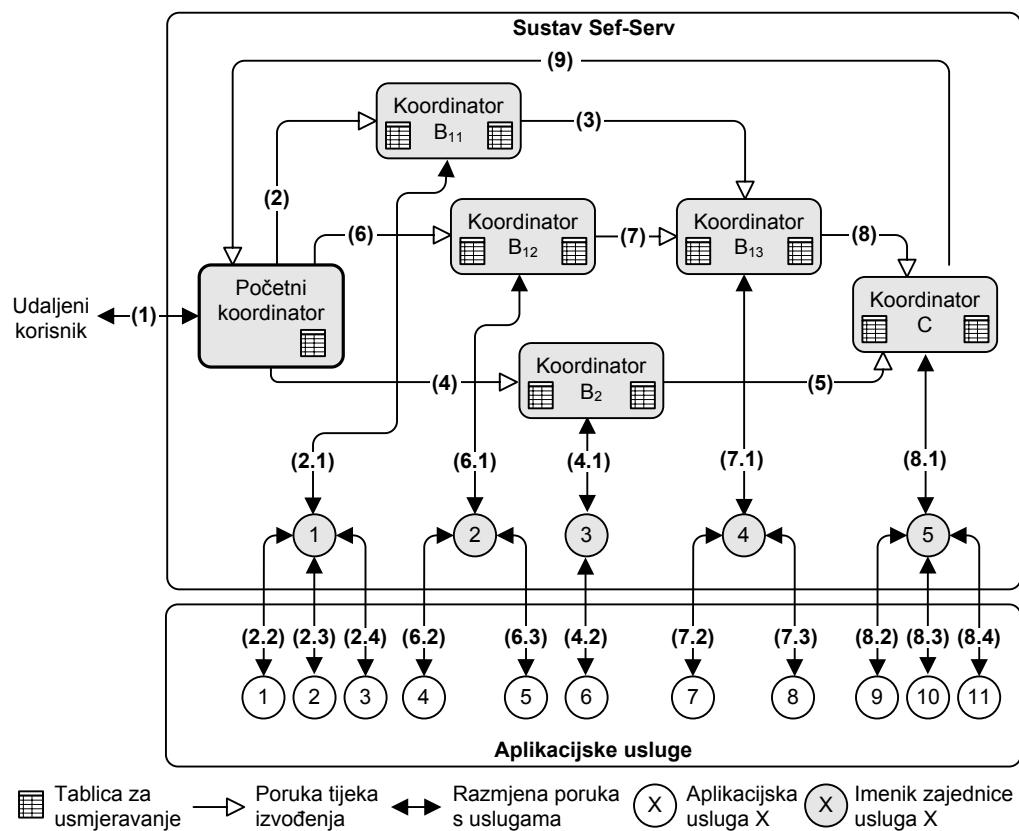
Slika 41: Primjer dijagrama ostvarenog u jeziku *statecharts* koji opisuje tijek izvođenja raspodijeljene aplikacije zasnovane na uslugama

Slika 41 prikazuje primjer dijagrama koji je ostvaren u jeziku *statecharts* i opisuje tijek izvođenja raspodijeljene aplikacije zasnovane na uslugama. Tijek izvođenja raspodijeljene aplikacije u primjeru započinje izvođenjem usluga pridruženih složenom stanju *B*. Složeno stanje *B* podijeljeno je u dva stanja u kojima se usluge izvode usporedno. U gornjem stanju složenog stanja *B* početnim granama pridružena su dva međusobno oprečna uvjeta. U slučaju da je ispunjen uvjet *x*, izvodi se usluga pridružena osnovnom stanju *B₁₁* i zatim se izvodi usluga pridružena osnovnom stanju *B₁₃*. U suprotnome, ako uvjet *x* nije ispunjen, izvodi se usluga pridružena osnovnom stanju *B₁₂*, a zatim usluga pridružena osnovnom stanju *B₁₃*. U donjem stanju složenog stanja *B* istovremeno s gornjim stanjem izvodi se usluga pridružena osnovnom stanju *B₂*. Nakon uspješnog završetka izvođenja svih usluga pridruženih složenom stanju *B*, izvodi se usluga pridružena osnovnom stanju *C*.

Tijekom oblikovanja raspodijeljene aplikacije zasnovane na uslugama primjenom *statecharts* dijagrama svakom osnovnom stanju dijagrama ne pridružuje se adresa konkretnе usluge, već adresa imenika zajednice usluga. Imenik zajednice usluga sadrži prijave skupine usluga jednakih funkcijskih i različitih nefunkcijskih značajki. Nadalje, svakom osnovnom stanju pridruženi su i opisi nefunkcijskih značajki tražene usluge na osnovi koji imenik zajednice tijekom izvođenja raspodijeljene aplikacije odabire najprikladniju od svih trenutno prijavljenih usluga. Primjena imenika zajednice usluga omogućava izgradnju prilagodljivih i dinamičnih raspodijeljenih aplikacija.

Programski sustav *Self-Serv* ostvaruje raspodijeljeno upravljanje tijekom izvođenja raspodijeljene aplikacije zasnovano na modelu ravnopravnih sudionika. Upravljanje tijekom izvođenja raspodijeljenih aplikacija ostvaruje se primjenom skupine međusobno ravnopravnih upravljačkih komponenata nazvanih koordinatorima. Zadaća koordinatora je upravljanje i nadgledanje izvođenja skupine aplikacijskih usluga i razmjena poruka s udaljenim koordinatorima s ciljem ostvarivanja raspodijeljenog upravljanja tijekom

izvođenja raspodijeljene aplikacije. Tijekom izvođenja raspodijeljene aplikacije, svaki koordinator provodi odgovarajuće akcije korištenjem tablica za usmjeravanje koje sadrže preduvjete i akcije. Tablica za usmjeravanje koja sadrži preduvjete određuje uvjete koji moraju biti ispunjeni kako bi koordinator započeo izvođenje. Tipični uvjet u tablici preduvjeta koordinatora je primitak upravljačkih poruka od odgovarajućih susjednih koordinatora. Nakon ispunjavanja uvjeta definiranih u tablici preduvjeta, koordinator pristupa pridruženom imeniku zajednice usluga kako bi pristupio aplikacijskoj usluzi i izvršio odsječak ukupnog tijeka izvođenja raspodijeljene aplikacije. Nakon završetka izvođenja aplikacijske usluge, koordinator koristi tablicu za usmjeravanje. U tablici za usmjeravanje zapisane su adrese susjednih koordinatora kojima je potrebno proslijediti upravljačke poruke s rezultatima izvođenja aplikacijske usluge. Sadržaj tablice preduvjeta i tablice akcija određuje se statički primjenom *statecharts* dijagrama koji opisuje tijek izvođenja raspodijeljene aplikacije.



Slika 42: Primjer izvođenja raspodijeljene aplikacije korištenjem *Self-Serv* sustava

Slika 42 prikazuje primjer izvođenja raspodijeljene aplikacije zasnovane na uslugama koja je opisana *statecharts* dijagramom prikazanim na slici 41. Izvođenje raspodijeljene aplikacije ostvareno je korištenjem početnog koordinatora i jednog

koordinatora za svako osnovno stanje *statechart* dijagrama raspodijeljene aplikacije (B_{11} , B_{12} , B_{13} , B_2 , C). Početni koordinator ostvaruje pristupno sučelje raspodijeljene aplikacije i započinje tijek izvođenja po primitku poruke zahtjeva od udaljenog korisnika (1). Korištenjem sadržaja primljene poruke zahtjeva i akcija zapisanih u tablici za usmjeravanje, početni koordinator prosljeđuje primljenu poruku koordinatoru B_{11} ili B_{12} (2, 6) i koordinatoru B_2 (2). Nakon primitka poruke zahtjeva, koordinator B_{11} prosljeđuje primljenu poruku zahtjeva imeniku zajednice usluga 1 (2.1). Imenik zajednice usluga 1 odabire jednu od trenutno prijavljenih aplikacijskih usluga i prosljeđuje joj primljenu poruku zahtjeva (2.2, 2.3, 2.4). Po primitku poruke odgovora od odabrane aplikacijske usluge, imenik zajednice usluga 1 prosljeđuje poruku odgovora koordinatoru B_{11} . Nadalje, koordinator B_{11} prosljeđuje primljenu poruku odgovora koordinatoru B_{13} . Na sličan način koordinatori B_{12} , B_{13} , B_2 i C međusobno razmjenjuju poruke i pozivaju pridružene im imenike zajednice usluga (4-9), kako bi ostvarili tijek izvođenja raspodijeljene aplikacije. Nakon primitka poruke odgovora od imenika zajednice usluga 5, Koordinator C usmjerava primljenu poruku odgovora početnom koordinatoru (9). Izvođenje raspodijeljene aplikacije završava nakon što početni koordinator primi poruku odgovora i prosljedi primljenu poruku odgovora udaljenom korisniku (10).

5. poglavlje

Raspodijeljeni interpretator programa

Postojeći sustavi za oblikovanje, izgradnju i izvođenje raspodijeljenih aplikacija primjenom metoda kompozicije usluga koji su dostupni na tržištu imaju nekoliko nedostataka. Većina dostupnih sustava za izvođenje složenih usluga ostvareni su kao središnji poslužitelji koji izvode opise složenih usluga. Središnji poslužitelji zasnivaju se na uporabi središnjeg modela za upravljanje tijekom izvođenja složenih usluga (engl. *centralized execution management model*), predstavljaju središnju točku nepouzdanosti (engl. *single point of failure*) složenih usluga i nemaju mogućnost razmjernog rasta (engl. *scalability*). Nadalje, poslužitelji za izvođenje složenih usluga imaju složenu građu i zasnivaju se na primjeni složenih jezika za ostvarivanje kompozicije usluga.

Kako bi se umanjili navedeni problemi sustava za izgradnju složenih usluga, u sklopu projekta *MidArc* posrednik oblikovan je i razvijen raspodijeljeni interpretator programa. Projekt *MidArc* ostvaren je u suradnji s istraživačkim odjelom kompanije *Ericsson Nikola Tesla d.d.* Nadalje, projekt *MidArc* također je uključen u nacionalni poliprojekt *CroGrid* koje je ostvaren uz potporu Ministarstva znanosti, obrazovanja i športa Republike Hrvatske.

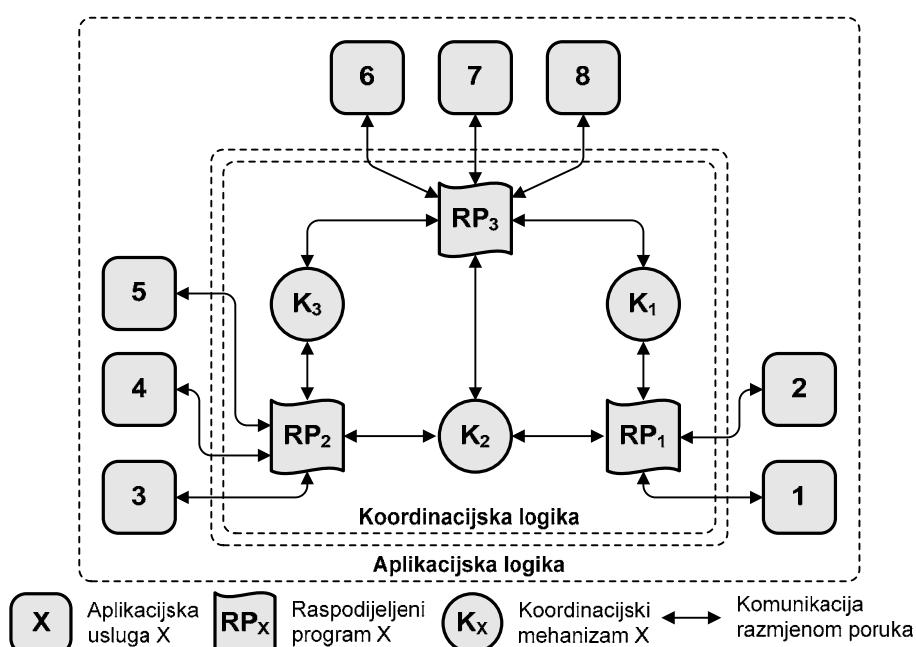
Raspodijeljeni interpretator programa [114] omogućava razvoj, postavljanje i izvođenje raspodijeljenih aplikacija zasnovanih na uslugama. Raspodijeljene aplikacije zasnovane na uslugama grade se primjenom skupa raspodijeljenih programa koji se međusobno natječu i surađuju kako bi ostvarili funkcionalnosti raspodijeljene aplikacije. Svaki raspodijeljeni program koristi skup aplikacijskih usluga i ostvaruje odsječak ukupnog tijeka izvođenja aplikacije. Logika raspodijeljenih programa opisuje se primjenom programskog jezika za suradnju i natjecanje (engl. *Coopetition Language, CL*). *CL* je jezik za opis procesa koji omogućava izgradnju raspodijeljenih aplikacija primjenom hibridne metode kompozicije usluga koja je zasnovana na orkestraciji i koreografiji. Opisi raspodijeljenih programa ostvareni primjenom jezika *CL* kompaktne su programske cjeline koje sadrže sve informacije potrebne za izvođenje raspodijeljenih programa. Zbog navedenog svojstva, raspodijeljene programe moguće je izvoditi primjenom raspodijeljenog interpretatora programa. Za razliku od središnjih poslužitelja za izvođenje složenih usluga,

raspodijeljeni interpretator programa omogućava izvođenje raspodijeljenih aplikacija primjenom raspodijeljenog upravljanja tijekom izvođenja. Primjena raspodijeljenog upravljanja tijekom izvođenja aplikacija omogućava izvođenje raspodijeljenih aplikacija sa svojstvima razmjerog rasta i otpornosti na pogreške.

5.1 Oblikovanje raspodijeljenih aplikacija zasnovanih na uslugama

Raspodijeljene aplikacije zasnovane na uslugama oblikuju se primjenom načela razdvajanja aplikacijske logike (engl. *application-specific logic*) i koordinacijske logike (engl. *coordination logic*) po uzoru na raspodijeljene sustave zasnovane na koordinaciji (engl. *coordination-based distributed systems*) [115]. Logika raspodijeljenih aplikacija zasnovanih na uslugama ostvaruje se primjenom skupa aplikacijskih usluga (engl. *application-specific services*), raspodijeljenih programa (engl. *distributed programs*) i koordinacijskih mehanizama (engl. *coordination mechanisms*).

Aplikacijske usluge ostvaruju logiku raspodijeljene aplikacije. Raspodijeljeni programi ostvaruju logiku za povezivanje aplikacijskih usluga prema zahtjevima tijeka izvođenja raspodijeljene aplikacije. Koordinacijski mehanizmi omogućavaju međusobnu komunikaciju i sinkronizaciju tijeka izvođenja raspodijeljenih programa i koriste se za ostvarivanje koordinacijske logike raspodijeljenih aplikacija.



Slika 43: Primjer raspodijeljene aplikacije zasnovane na uslugama

Slika 43 prikazuje primjer arhitekture raspodijeljene aplikacije zasnovane na uslugama. Prikazana raspodijeljena aplikacija sadrži osam aplikacijskih usluga (I_1 - I_8) koje ostvaruju odsječke logike raspodijeljene aplikacije. Nadalje, korištena su tri raspodijeljena programa (RP_1 - RP_3) i tri koordinacijska mehanizma (K_1 - K_3) u svrhu ostvarivanja koordinacijske logike za prikazanu raspodijeljenu aplikaciju.

5.1.1 Aplikacijske usluge

Aplikacijske usluge ostvaruju pristup različitim računalnim sredstvima, korisničkim programima i računalnim sustavima, kao što su sustavi za spremanje podataka (engl. *data storage systems*), grozdovi računala (engl. *clusters*), osjetila (engl. *sensors*) i aktuatori (engl. *actuators*). Najčešće se za aplikacijske usluge koriste usluge dostupne u globalnoj mreži Internet. U slučaju da odgovarajuće aplikacijske usluge nisu dostupne u globalnoj mreži Internet, potrebne aplikacijske usluge samostalno razvijaju razvijatelji raspodijeljene aplikacije ili ih naručuju od specijaliziranih razvijatelja usluga.

5.1.2 Raspodijeljeni programi

Raspodijeljeni programi omogućavaju međusobno povezivanje aplikacijskih usluga u svrhu ostvarivanja raspodijeljenih aplikacija. Raspodijeljene aplikacije sadrže skup raspodijeljenih programa koji međusobno se natječu i surađuju kako bi ostvarili njezinu koordinacijsku logiku. Logika raspodijeljenih programa ostvarena je korištenjem programskog jezika *CL*. *CL* je jezik koji omogućava izgradnju raspodijeljenih aplikacija zasnovanih na uslugama primjenom hibridne metode kompozicije usluga zasnovane na orkestraciji i koreografiji.

5.1.3 Koordinacijski mehanizmi

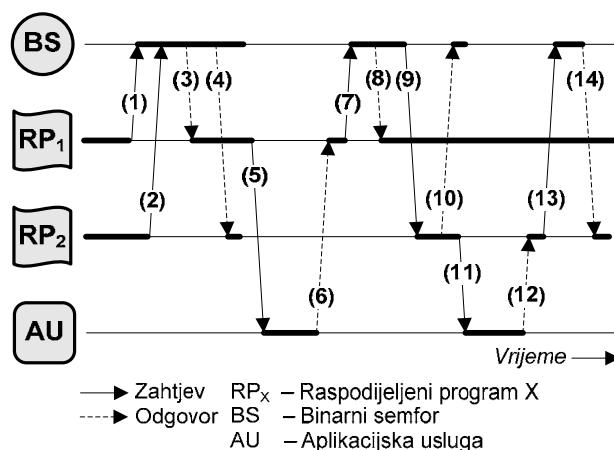
Koordinacijski mehanizmi [116] su općeniti mehanizmi koji omogućavaju komunikaciju, sinkronizaciju, suradnju i natjecanje raspodijeljenih programa. Koordinacijski mehanizmi uključuju binarni semafor (engl. *binary semaphore*), opći semafor (engl. *counting semaphore*), poštanski pretinac (engl. *mailbox*) i usmjernik događaja (engl. *event-channel*). Koordinacijski mehanizmi ostvareni su kao usluge s čuvanjem stanja zasnovane na *Web Services* tehnologiji i specifikaciji *WS-ResourceFramework*.

Binarni i opći semafor

Binarni i opći semafori omogućavaju sinkronizaciju tijeka izvođenja raspodijeljenih programa i međusobno isključivanje raspodijeljenih programa tijekom pristupa dijeljenim

sredstvima. Raspodijeljeni program ima pravo pristupa dijeljenom sredstvu koji je zaštićen semaforom samo ako je prethodno zauzeo semafor. Binarni semafor može istovremeno zauzeti samo jedan raspodijeljeni program, dok opći semafor može istovremeno zauzeti ograničeni broj raspodijeljenih programa.

Zauzimanje semafora moguće je ostvariti primjenom modela zauzimanja zasnovanog na ispitivanju (engl. *poll*) ili modela zauzimanja zasnovanog na dojavni (engl. *call back*). U slučaju da je semafor dostupan, raspodijeljeni program će uspješno zauzeti semafor bez obzira na odabrani model za zauzimanje semafora. U slučaju da semafor nije dostupan, a raspodijeljeni program koristi model zauzimanja zasnovan na ispitivanju, semafor vraća negativan odgovor. Raspodijeljeni program može ponoviti postupak zauzimanja semafora nakon isteka proizvoljnog vremenskog razdoblja. U slučaju da semafor nije dostupan, a raspodijeljeni program koristi model zauzimanja zasnovan na dojavni, semafor sprema adresu raspodijeljenog programa u rep čekanja. U slučaju da rep čekanja nije prazan tijekom oslobađanja semafora, semafor se pridružuje raspodijeljenom programu s najstarijim zapisom u repu čekanja. Semafor uklanja zapis odabranog raspodijeljenog programa iz repa čekanja i dojavljuje raspodijeljenom programu uspješno zauzeće semafora.



Slika 44: Primjer korištenja binarnog semafora za međusobno isključivanje pristupa dijeljenoj aplikacijskoj usluzi

Slika 44 prikazuje primjer korištenja binarnog semafora u svrhu ostvarivanja međusobnog isključivanja pristupa raspodijeljenih programa dijeljenoj aplikacijskoj usluzi. Raspodijeljeni programi RP_1 i RP_2 pokušavaju ostvariti pristup aplikacijskoj usluzi AU koja je zaštićena binarnim semaforom BS . Kako bi ostvarili pristup aplikacijskoj usluzi AU , raspodijeljeni programi RP_1 i RP_2 približno istovremeno pokušavaju zauzeti binarni semafor BS (1, 2) primjenom modela zauzimanja semafora zasnovanog na dojavni. Obzirom da

raspodijeljeni program RP_1 zahtjeva zauzeće semafora BS prije raspodijeljenog programa RP_2 , semafor BS se pridjeljuje raspodijeljenom programu RP_1 . Semafor BS prosljeđuje pozitivnu potvrdu raspodijeljenom programu RP_1 (3), spremi povratnu adresu raspodijeljenog programa RP_2 i prosljeđuje negativnu potvrdu raspodijeljenom programu RP_2 (4). Nakon primitka negativne potvrde, raspodijeljeni program RP_2 čeka dojavu od semafora BS o uspješnom zauzeću semafora. Nakon primitka pozitivne potvrde, raspodijeljeni program RP_1 pristupa aplikacijskoj usluzi AU (5), prima rezultat obrade aplikacijske usluge AU (6) i oslobađa binarni semafor BS (7). Binarni semafor BS potvrđuje oslobađanje raspodijeljenom programu RP_1 (8) i dojavljuje uspješno zauzeće semafora raspodijeljenom programu RP_2 (9). Nakon primitka dojave o zauzeću, raspodijeljeni program RP_2 potvrđuje primljenu dojavu binarnom semaforu BS (10), pristupa aplikacijskoj usluzi AU (11), dohvaća rezultate obrade aplikacijskoj usluge AU (12) i oslobađa binarni semafor BS (13). Binarni semafor BS potvrđuje uspješno oslobađanje semafora (14) raspodijeljenom programu RP_2 .

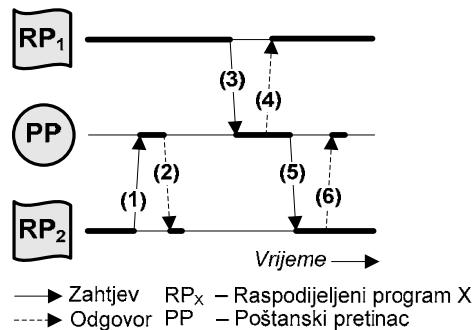
Poštanski pretinac

Poštanski pretinac [117] je koordinacijski mehanizam koji omogućava komunikaciju razmjenom poruka koja je postojana (engl. *persistent*), asinkrona (engl. *asynchronous*), vremenski razdvojena (engl. *temporally uncoupled*) i povezana prema naslovljavanju (engl. *referentially coupled*). Poruke koje se razmjenjuju primjenom poštanskog pretinca su *XML* dokumenti proizvoljnog sadržaja. Obzirom da su poruke zapisane primjenom jezika *XML*, poštanski pretinac omogućava razmjenu podataka koji ne ovise o korištenim računalnim platformama, operacijskim sustavima i razvojnim okolinama.

Poštanski pretinac sadrži dvije strukture podataka: rep poruka i rep zahtjeva. Rep poruka služi za spremanje poruka pristiglih u poštanski pretinac. Rep zahtjeva služi za spremanje povratnih adresa pozivatelja koji zahtjevaju primitak poruke iz poštanskog pretinca. U slučaju da je u trenutku prispijeća poruke u poštanski pretinac rep zahtjeva prazan, pristigla poruka spremi se u rep poruka. U protivnome, ako rep zahtjeva nije prazan, poštanski pretinac prosljeđuje primljenu poruku pozivatelju čiji je zapis najstariji u repu zahtjeva i uklanja odabrani zapis iz repa zahtjeva.

Poruku spremljenu u poštanskom pretincu moguće je dohvatiti primjenom modela zasnovanog na ispitivanju (engl. *poll*) ili primjenom modela zasnovanog na dojavi (engl. *call-back*). U slučaju da u repu poruka postoje spremljene poruke, bez obzira na odabrani model dohvaćanja poruka, poštanski pretinac uklanja najstariju poruku u repu poruka i prosljeđuje uklonjenu poruku pozivatelju. U slučaju da poštanski pretinac ne sadrži poruke u

repu poruka, a pozivatelj koristi model za dohvaćanje poruka zasnovan na ispitivanju, poštanski pretinac uzvraća negativnu potvrdu. Pozivatelj može ponoviti dohvaćanje poruke iz poštanskog pretinca nakon isteka proizvoljnog vremenskog razdoblja. Ako poštanski pretinac ne sadrži poruke u repu poruka, a pozivatelj koristi model dohvaćanja zasnovan na dojavi, poštanski pretinac spremu u rep zahtjeva novi zapis koji sadrži povratnu adresu pozivatelja.



Slika 45: Primjer asinkrone komunikacije između dva raspodijeljena programa primjenom poštanskog pretinca

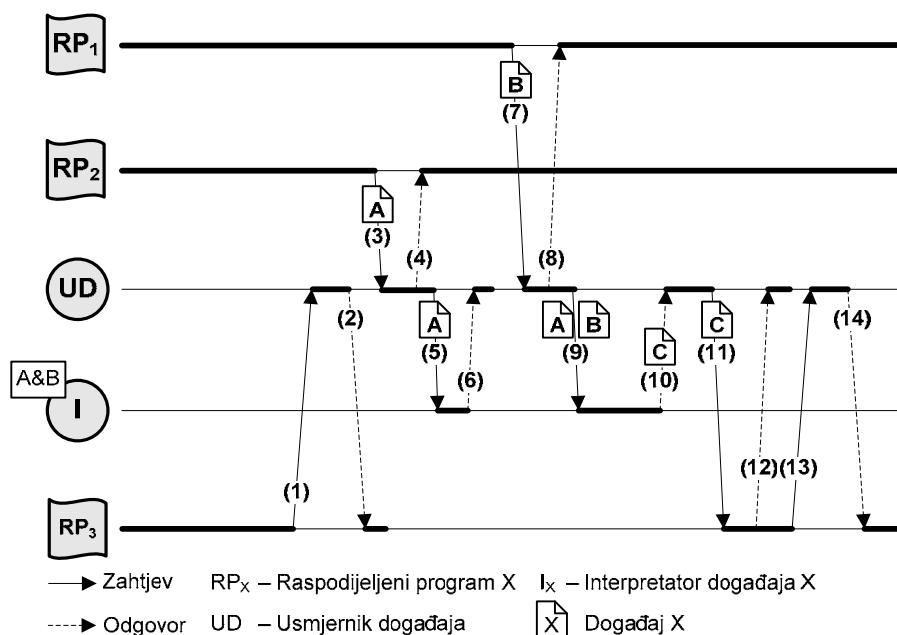
Slika 45 prikazuje primjer asinkrone komunikacije razmjenom poruka između dva raspodijeljena programa koja je ostvarena primjenom poštanskog pretinca. Raspodijeljeni program RP_2 koristi model dohvaćanja poruke zasnovan na dojavi (1). Obzirom da je rep poruka poštanskog pretinaca PP prazan, poštanski pretinac PP spremu povratnu adresu raspodijeljenog programa RP_2 u rep zahtjeva i proslijedi negativnu potvrdu raspodijeljenom programu RP_2 (2). Raspodijeljeni program RP_1 upućuje poruku u poštanski pretinac PP (3). Poštanski pretinac PP potvrđuje primitak poruke raspodijeljenom programu RP_1 (4) i proslijedi primljenu poruku raspodijeljenom programu RP_2 (5). Raspodijeljeni program RP_2 prima poruku i potvrđuje primitak poruke poštanskom pretincu PP (6).

Usmjernik događaja

Usmjernik događaja [118] je objava/preplata (engl. *publish/subscribe*) mehanizam s naprednim mogućnostima tumačenja događaja. Primjenom usmjernika događaja moguće je izgraditi raspodijeljene sustave poticane događajima (engl. *event-driven distributed systems*), raspodijeljene sustave zasnovane na dokumentima (engl. *document-oriented distributed systems*) i mreže zasnovane na sadržaju (engl. *content-based networks*). Okolina usmjernika događaja uključuje objavitelje (engl. *publishers*), pretplatnike (engl. *subscribers*) i interpretatore (engl. *interpreters*). Objavitelji objavljuju različite vrste događaja koje spremu usmjernik događaja. Pretplatnici se pretplaćuju za primanje događaja spremljenih u usmjerniku događaja. Interpretatori ostvaruju logiku za usmjeravanje objavljenih događaja

prema odgovarajućim pretplatnicima. Svaki interpretator je ostvaren kao zasebna usluga neovisna o logici usmjernika događaja, čime se postiže neovisnost usmjernika događaja o značenju i primjeni događaja.

Usmjernik događaja sadrži skup događaja i skup pretplata. Skup događaja služi za spremanje svih događaja koji su objavljeni u usmjerniku događaja. Skup pretplata služi za spremanje zapisa svih važećih pretplata. Zapis pretplate sadrži povratnu adresu pretplatnika, adresu interpretatora i pretplatnički dokument. Adresom interpretatora određen je interpretator koji je zadužen za tumačenje događaja važećih za zadalu pretplatu. Pretplatnički dokument sadrži dodatne informacije koje koristi interpretator kako bi mogao donositi odluke o vrsti događaja za koje je pretplata važeća.



Slika 46: Primjer uporabe usmjernika događaja za ostvarivanje komunikacije zasnovane na događajima

Slika 46 prikazuje primjer uporabe usmjernika događaja za ostvarivanje komunikacije zasnovane na događajima između raspodijeljenih programa. Raspodijeljeni program RP_3 prijavljuje usmjerniku događaja UD pretplatu na pojavu događaja za čije je tumačenje zadužen interpretator I (1). Uvjet koji ostvaruje interpretator I je istovremeno objavljivanje događaja A i B . Usmjernik događaja UD potvrđuje pretplatu raspodijeljenog programa RP_3 (2). Nakon toga, raspodijeljeni program RP_2 objavljuje događaj A na usmjerniku događaja UD (3). Usmjernik događaja UD potvrđuje primitak događaja raspodijeljenom programu RP_2 (4) i proslijedi primljeni događaj A interpretatoru I (5). Obzirom da nisu objavljena oba događaja A i B , već samo događaj A , interpretator I vraća

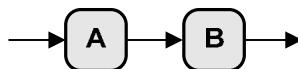
negativnu potvrdu usmjerniku događaja UD (6). Raspodijeljeni program RP_1 potom objavljuje događaj B (7) na usmjerniku događaja UD . Usmjernik događaja UD potvrđuje primitak događaja raspodijeljenom programu RP_1 (8) i prosljeđuje prethodno primljeni događaj A i novi događaj B interpretatoru I (9). Obzirom da je ispunjen uvjet pretplate, interpretator I koristi događaje A i B kako bi izgradio događaj C i prosljeđuje izgrađeni događaj usmjerniku događaja UD (10). Usmjernik događaja prima izgrađeni događaj C i prosljeđuje ga raspodijeljenom programu RP_3 (11). Raspodijeljeni program RP_3 potvrđuje primitak događaja C usmjerniku događaja UD (12). Nadalje, raspodijeljeni program RP_3 odjavljuje vlastitu pretplatu iz usmjernika događaja UD (13), koji potvrđuje odjavu pretplate (14).

5.1.4 Obrasci za oblikovanje raspodijeljenih aplikacija

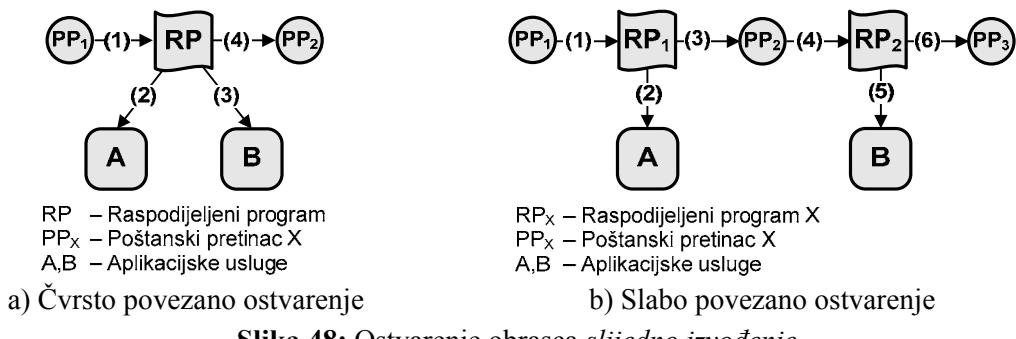
Raspodijeljene aplikacije oblikuju se primjenom obrazaca za izgradnju tijeka izvođenja aplikacije (engl. *control-flow patterns*) [119]. Obrasce za izgradnju tijeka izvođenja moguće je koristiti za povezivanje aplikacijskih usluga, raspodijeljenih programa i koordinacijskih mehanizama u svrhu izgradnje raspodijeljenih aplikacija zasnovanih na uslugama. Osnovne obrasce za izgradnju raspodijeljenih aplikacija zasnovanih na uslugama čine obrasci *slijedno izvođenje* (engl. *sequence*), *ponavljajuće izvođenje* (engl. *iteration*), *usporedno grananje* (engl. *parallel split*), *sinkronizacija* (engl. *synchronization*), *uvjetno grananje* (engl. *multi-choice*), *višestruko spajanje* (engl. *synchronizing merge*), *diskriminatore* (engl. *discriminator*) i *neuređeno slijedno izvođenje* (engl. *interleaved parallel routing*).

Slijedno izvođenje

Obrazac *slijedno izvođenje* primjenjuje se u slučajevima kad je potrebno ostvariti slijedno izvođenje skupine aplikacijskih usluga, pri čemu se rezultat izvođenja jedne usluge prosljeđuje kao ulaz idućoj usluzi u nizu. Slika 47 prikazuje primjer obrasca *slijedno izvođenje* ostvarenog primjenom dvije usluge. Rezultat izvođenja aplikacijske usluge A prosljeđuje se ako ulaz aplikacijskoj usluzi B .



Slika 47: Primjer obrasca *slijedno izvođenje*

Slika 48: Ostvarenje obrasca *slijedno izvođenje*

Slika 48 prikazuje dva moguća ostvarenja obrasca *slijedno izvođenje* prikazanog na slici 47. Slika 48a) prikazuje čvrsto povezano ostvarenje obrasca *slijedno izvođenje* koji je ostvaren primjenom dva poštanska pretinca (PP_1, PP_2) i jednog raspodijeljenog programa (RP). Poštanski pretinac PP_1 sprema poruke koje sadrže ulazne podatke za aplikacijsku uslugu A . Raspodijeljeni program RP dohvaća ulazne podatke iz poštanskog pretinca PP_1 (1) i proslijeđuje primljene podatke aplikacijskoj usluzi A (2). Nadalje, rezultate izvođenja aplikacijske usluge AU, raspodijeljeni program RP proslijeđuje aplikacijskoj usluzi B (3). Rezultate izvođenja aplikacijske usluge B raspodijeljeni program RP proslijeđuje u poštanski pretinac PP_2 (4).

Slika 48b) prikazuje slabo povezano ostvarenje obrasca *slijedno izvođenje* ostvareno primjenom tri poštanska pretinaca (PP_1, PP_2, PP_3) i dva raspodijeljena programa (RP_1, RP_2). Poštanski pretinac PP_1 sprema podatke koji služe kao ulaz za aplikacijsku uslugu A . Raspodijeljeni program RP_1 dohvaća podatke spremljene u poštanskom pretincu PP_1 (1), proslijeđuje ih kao ulaz aplikacijskoj usluzi A (2), a rezultat izvođenja aplikacijske usluge A šalje u poštanski pretinac PP_2 (3). Raspodijeljeni program RP_2 dohvaća podatke spremljene u poštanskom pretincu PP_2 (4), proslijeđuje ih kao ulaz aplikacijskoj usluzi B (5), dohvaća rezultate izvođenja aplikacijske usluge B i sprema rezultate izvođenja aplikacijske usluge B u poštanski pretinac PP_3 (6).

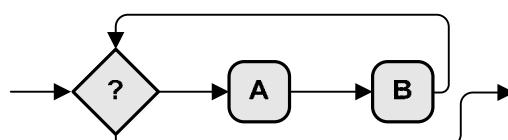
Prednost primjene čvrsto povezanog ostvarenja obrasca *slijedno izvođenje* je korištenje samo jednog raspodijeljenog programa i dva poštanska pretinca bez obzira na ukupan broj aplikacijskih usluga za koje se obrazac primjenjuje. Osnovni nedostatci primjene čvrsto povezanog ostvarenja su neučinkovito izvođenje i nemogućnost razmjerne rasta s obzirom na broj korisnika i korištenih aplikacijskih usluga. Neučinkovitost izvođenja nastupa u slučaju spremanja dva ili više nezavisnih zahtjeva u ulazni poštanski pretinac koji je moguće izvoditi usporedno. Budući da jedan raspodijeljeni program može istovremeno poslužiti samo jedan zahtjev, naknadno pristigli zahtjevi moraju čekati u repu ulaznog

poštanskog pretinca kako bi bili obrađeni. Obzirom da jedan raspodijeljeni program predstavlja kritičnu točku, čvrsto povezano ostvarenje nema svojstvo razmjernog rasta s obzirom na brojnost korištenih aplikacijskih usluga i pristiglih zahtjeva. Navedeni nedostatci više su izraženi čim je trajanje obrade koju provode aplikacijske usluge vremenski duže.

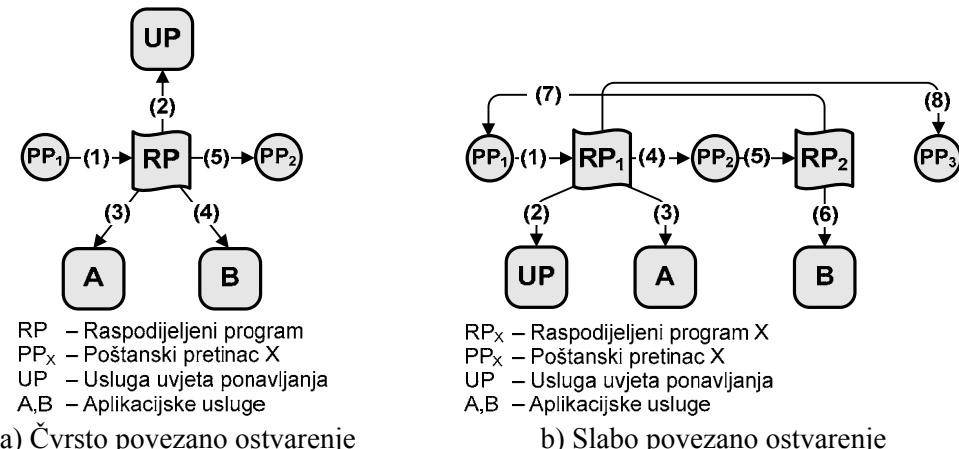
Prednost slabo povezanog ostvarenja obrasca *slijedno izvođenje* je povećanje učinkovitosti izvođenja i poboljšanje svojstava razmjernog rasta u odnosu na čvrsto povezano ostvarenje obrasca. Povećanje učinkovitosti izvođenja ostvaruje se obradom zahtjeva primjenom cjevovoda (engl. *pipeline*). Svaki raspodijeljeni program pridružen je samo jednoj aplikacijskoj usluzi, te ostvaruje obradu nezavisno od ostalih raspodijeljenih programa u nizu. Zahtjevi u ulaznom poštanskom pretincu ne moraju čekati na izvođenje svih aplikacijskih usluga u nizu, već samo na izvođenje aplikacijske usluge pridružene RP-u koji čita poruke iz određenog poštanskog pretinca. Korištenjem većeg broja raspodijeljenih programa koji se izvode istovremeno i nezavisno na različitim računalima, omogućeno je raspoređivanje opterećenja i ostvarivanje svojstva razmjernog rasta obzirom na brojnost aplikacijskih usluga i brojnost pristiglih zahtjeva. Nedostatak slabo povezanog ostvarenja u odnosu na čvrsto povezano je veći broj poštanskih pretinaca i raspodijeljenih programa kojima je potrebno učinkovito upravljati. U slučaju da aplikacijske usluge ne zahtjevaju značajnu količinu vremena za provođenje obrade, dodatno vrijeme utrošeno na komunikaciju razmjenom poruka putem poštanskih pretinaca može biti značajno u odnosu na ukupno vrijeme izvođenja. U takvim rubnim slučajevima slabo povezano ostvarenje je manje učinkovito u odnosu na čvrsto povezano ostvarenje obrasca.

Ponavljaće izvođenje

Obrazac *ponavljaće izvođenje* omogućava izgradnju petlji u kojima se slijedno izvodi skup aplikacijskih usluga. Odluka o ponavljanju petlje donosi se ispitivanjem uvjeta u trenutku ulaska u petlju. Slika 49 prikazuje primjer obrasca *ponavljaće izvođenje* koji ostvaruje slijedno izvođenje dviju aplikacijskih usluga u petlji s ispitivanjem uvjeta ponavljanja na početku petlje.



Slika 49: Primjer obrasca *ponavljaće izvođenje*

Slika 50: Ostvarenje obrasca *ponavljajuće izvođenje*

Slika 50 prikazuje dva moguća ostvarenja obrasca *ponavljajuće izvođenje*. Slika 50a) prikazuje čvrsto povezano ostvarenje obrasca primjenom dvaju poštanskih pretinaca (PP_1 , PP_2), usluge koja ostvaruje uvjet ponavljanja (UP) i jednog raspodijeljenog programa (RP). Raspodijeljeni program RP dohvaća ulazne podatke spremu u poštanskom pretincu PP_1 (1) i proslijedi ih kao ulaz usluzi uvjeta ponavljanja UP (2). U slučaju da usluga uvjeta ponavljanja UP odgovori negativnom potvrdom, raspodijeljeni program RP proslijedi primljene ulazne podatke u poštanski pretinac PP_2 (5), čime završava izvođenje obrasca. U protivnome, ako usluga uvjeta ponavljanja UP odgovori pozitivnom potvrdom, raspodijeljeni program RP ostvaruje slijedno izvođenje aplikacijskih usluga A i B , slično kao u ostvarenju obrasca *slijedno izvođenje* koje je prikazano na slici 48.a). Nakon završetka slijednog izvođenja aplikacijskih usluga A i B , raspodijeljeni program RP proslijedi rezultat izvođenja aplikacijske usluge B kao ulaz usluzi uvjeta ponavljanja UP i opisani postupak se ponavlja.

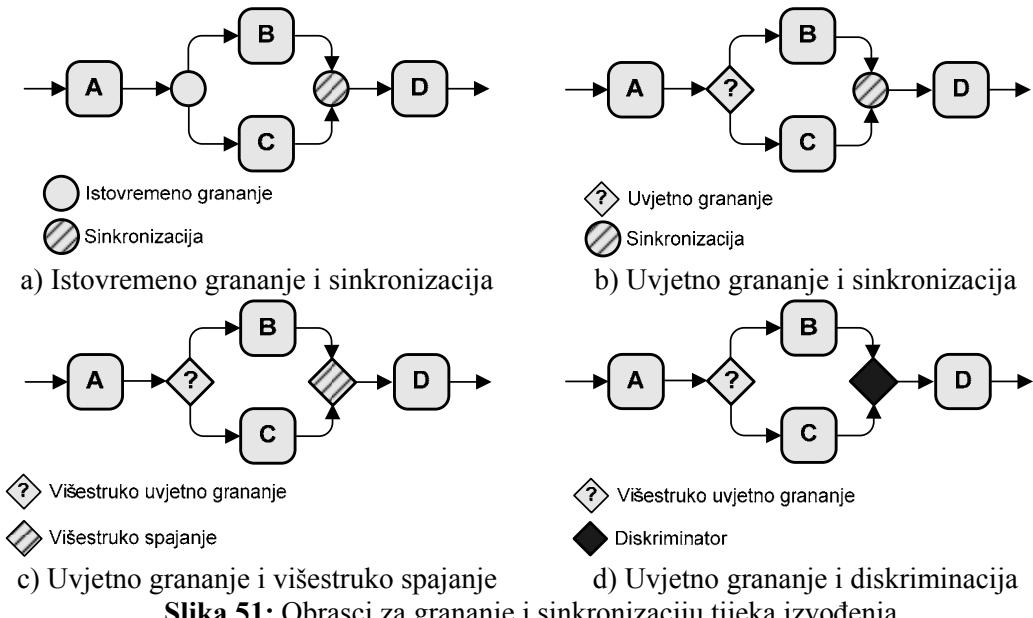
Slika 50.b) prikazuje slabo povezano ostvarenje uzorka *ponavljajuće izvođenje* ostvareno primjenom triju poštanskih pretinaca (PP_1 , PP_2 , PP_3), dva raspodijeljena programa (RP_1 , RP_2) i usluge uvjeta ponavljanja (UP). Poštanski pretinac PP_1 spremu podatke koji sadrže ulazne podatke. Raspodijeljeni program RP_1 dohvaća ulazne podatke iz poštanskog pretinca PP_1 (1) i proslijedi primljene podatke usluzi uvjeta ponavljanja UP (2). U slučaju da usluga uvjeta ponavljanja UP odgovori negativnom potvrdom, raspodijeljeni program RP_1 proslijedi primljene podatke u poštanski pretinac PP_3 (8), čime završava izvođenje obrasca. U protivnome, ako usluga uvjeta ponavljanja UP odgovori pozitivnom potvrdom, raspodijeljeni program RP_1 proslijedi primljene ulazne podatke u poštanski pretinac PP_2 (4) i ostvaruje slijedno izvođenje aplikacijskih usluga A i B slično kao u ostvarenju obrasca *slijedno izvođenje* prikazanom na slici 48.b). Po završetku slijednog izvođenja aplikacijskih

usluga A i B , raspodijeljeni program RP_3 proslijeđuje rezultate izvođenja aplikacijske usluge B u ulazni poštanski pretinac PP_1 (7), čime se opisani postupak ponavlja.

Prednosti i nedostatci čvrsto povezanog i slabo povezanog ostvarenja obrasca *ponavljajuće izvođenje* identične su prednostima i nedostatcima čvrsto povezanog i slabo povezanog ostvarenja obrasca *slijedno izvođenje*.

Usporedno grananje, sinkronizacija, uvjetno grananje, višestruko spajanje i diskriminator

Primjenom obrazaca *usporedno grananje*, *sinkronizacija*, *uvjetno grananje*, *višestruko spajanje* i *diskriminator* moguće je ostvariti različite vrste upravljanja i sinkronizacije tijeku izvođenja usluga. Slika 51 prikazuje najčešće korištene načine povezivanja obrazaca za izgradnju raspodijeljenih aplikacija zasnovanih na uslugama.



Slika 51: Obrasci za grananje i sinkronizaciju tijeka izvođenja

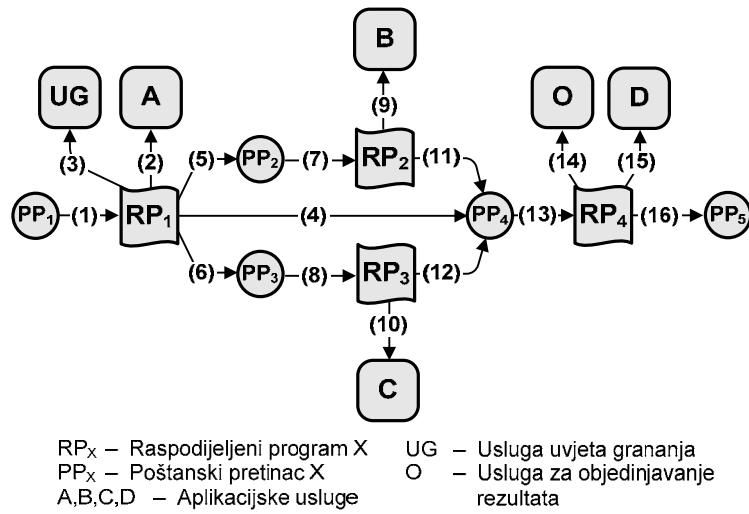
Slika 51a) prikazuje primjer korištenja obrazaca *grananje i sinkronizacija* koji je ostvaren s dvije grane. Aplikacijska usluga A izvodi se prije aplikacijskih usluga B , C i D . Nakon završetka izvođenja aplikacijske usluge A , aplikacijske usluge B i C primaju rezultate izvođenja aplikacijske usluge A i izvode se istovremeno. Nakon završetka izvođenja aplikacijskih usluga B i C , rezultati dobiveni njihovim izvođenjem objedinjuju se i proslijeđuju kao ulaz aplikacijskoj usluzi D .

Slika 51b) prikazuje primjer korištenja obrazaca *uvjetno grananje i sinkronizacija* koji je ostvaren s dvije grane. Aplikacijska usluga A izvodi se prije aplikacijskih usluga B , C i D . Ispitivanjem uvjeta grananja, za izvođenje se odabire aplikacijska usluga B ili C , ili se usporedno izvode obje usluge. U slučaju da se odabere samo jedna usluga, nakon završetka

izvođenja odabrane usluge, rezultat izvođenja odabrane usluge prosljeđuje se aplikacijskoj usluzi D . Ako se odabere usporedno izvođenje aplikacijskih usluga B i C , rezultati njihova izvođenja objedinjuju se i prosljeđuju aplikacijskoj usluzi D , koja zatim započinje izvođenje.

Slika 51c) prikazuje primjer korištenja obrazaca *uvjetno grananje* i *višestruko spajanje* koji je ostvaren s dvije grane. Aplikacijska usluga A izvodi se prije aplikacijskih usluga B , C i D . Ispitivanjem uvjeta grananja, za izvođenje se odabire aplikacijska usluga B ili C , ili se istovremeno izvode obje usluge. Ako se odabere samo aplikacijska usluga B ili C , rezultat izvođenja odabrane usluge prosljeđuje se kao ulaz usluzi D . Ako se odabere istovremeno izvođenje objiju aplikacijskih usluga, nakon završetka izvođenja pojedine usluge svaki dobiveni rezultat zasebno se prosljeđuje kao ulaz usluzi D . Zasebnim prosljeđivanjem rezultata izvođenja svake usluge stvaraju se dva međusobno nezavisna i usporedna tijeka izvođenja raspodijeljene aplikacije.

Slika 51d) prikazuje primjer korištenja obrazaca *uvjetno grananje* i *diskriminacija*. Aplikacijska usluga A izvodi se prije aplikacijskih usluga B , C i D . Ispitivanjem uvjeta grananja, za izvođenje se odabire aplikacijska usluga B ili C , ili se istovremeno izvode obje usluge. U slučaju da se odabere samo aplikacijska usluga B ili C , rezultat izvođenja odabrane usluge prosljeđuje se kao ulaz aplikacijskoj usluzi D . Ako se odabere istovremeno izvođenje objiju aplikacijskih usluga, čeka se na završetak izvođenja aplikacijske usluge B ili C . Ako sa izvođenjem prije završi aplikacijska usluga B , rezultat izvođenja aplikacijske usluge B se prosljeđuje usluzi D i rezultat izvođenja aplikacijske usluge C se zanemaruje. Ako sa izvođenjem prije završi aplikacijska usluga C , rezultat izvođenja aplikacijske usluge C se prosljeđuje usluzi D i rezultat izvođenja aplikacijske usluge B se zanemaruje.



Slika 52: Ostvarenje obrazaca *usporedno grananje, sinkronizacija, uvjetno grananje, višestruko spajanje i diskriminator*

Slika 52 prikazuje općenito ostvarenje obrazaca *usporedno grananje, sinkronizacija, uvjetno grananje, višestruko spajanje i diskriminator*. Raspodijeljeni program RP_1 dohvaća podatke spremljene u poštanskom pretincu PP_1 (1) i prosljeđuje ih kao ulaz aplikacijskoj usluzi A (2). Daljnje akcije koje provodi raspodijeljeni program RP_1 ovise o tome da li raspodijeljeni program ostvaruje obrazac *usporedno grananje* ili *uvjetno grananje*.

U slučaju da raspodijeljeni program RP_1 ostvaruje obrazac *usporedno grananje*, raspodijeljeni program RP_1 prosljeđuje rezultat izvođenja aplikacijske usluge A u poštanske pretince PP_2 i PP_3 (5, 6). Ako raspodijeljeni program RP_1 ostvaruje obrazac *uvjetno grananje*, raspodijeljeni program RP_1 prosljeđuje rezultate izvođenja aplikacijske usluge A usluzi uvjeta grananja UG (3). U slučaju da rezultat uvjeta grananja nalaže izvođenje obje izlazne grane, raspodijeljeni program RP_1 prosljeđuju poruku koja sadrži broj 2 u poštanski pretinac PP_4 (4). Poštanski pretinac PP_4 služi za spremanje broja grana koje su odabrane nakon ispitivanja uvjeta grananja. Nadalje, raspodijeljeni program RP_1 prosljeđuje rezultate izvođenja aplikacijske usluge A u poštanske pretince PP_2 i PP_3 . (5, 6).

Nakon spremanja podataka u poštanske pretince PP_2 i PP_3 , raspodijeljeni programi RP_2 i RP_3 dohvaćaju podatke u poštanskim pretincima PP_2 i PP_3 (7, 8), prosljeđuju ih aplikacijskim uslugama B i C (9, 10) i prosljeđuju rezultate izvođenja aplikacijskih usluga B i C u poštanski pretinac PP_4 (11, 12).

Logika raspodijeljenog programa RP_4 ovisi o tome da li raspodijeljeni program ostvaruje obrazac *sinkronizacija, višestruko spajanje* ili *diskriminator*. U slučaju da raspodijeljeni program RP_4 ostvaruje obrazac *sinkronizacija*, a grananje tijeka izvođenja je

ostvareno primjenom obrasca *istovremeno granje*, raspodijeljeni program RP_4 dohvaća sve poruke spremljene u poštanskom pretincu PP_4 (13). Nadalje, raspodijeljeni program RP_4 objedinjuje prihvaćene poruke korištenjem usluge za objedinjavanje rezultata O (14), te prosljeđuje objedinjene podatke kao ulaz usluzi D (15). Raspodijeljeni program PP_4 dohvaća rezultate izvođenja usluge D i prosljeđuje primljene rezultate u poštanski pretinac PP_5 (16).

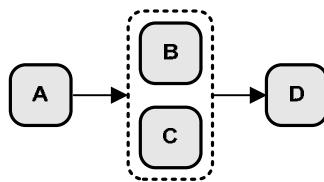
U slučaju da raspodijeljeni program RP_4 ostvaruje obrazac *sinkronizacija*, a granje tijeka izvođenja je ostvareno primjenom obrasca *uvjetno granje*, raspodijeljeni program RP_4 dohvaća prvu poruku spremljenu u poštanskom pretincu PP_4 . Prva poruka spremljena u poštanskom pretincu PP_4 odreduje broj grana odabranih tijekom izvođenja obrasca *uvjetno granje*. Poznavanjem broja odabranih grana, raspodijeljeni program RP_4 dohvaća odgovarajući broj poruka iz poštanskog pretinca PP_4 (8), te ih objedinjuje korištenjem usluge za objedinjavanje rezultata O (14). Nadalje, raspodijeljeni program RP_4 prosljeđuje objedinjene podatke kao ulaz aplikacijskoj usluzi D (15), dohvaća rezultat izvođenja usluge i prosljeđuje dohvaćeni rezultat u poštanski pretinac PP_5 (16).

U slučaju da raspodijeljeni program RP_4 ostvaruje obrazac *višestruko spajanje*, raspodijeljeni program RP_4 dohvaća svaku poruku spremljenu u poštanskom pretincu PP_4 zasebno (13). Nadalje, raspodijeljeni program RP_4 svaku primljenu poruku odvojeno prosljeđuje kao ulaz aplikacijskoj usluzi D (15), dohvaća rezultat izvođenja usluge i prosljeđuje dohvaćeni rezultat u poštanski pretinac RP_4 (16).

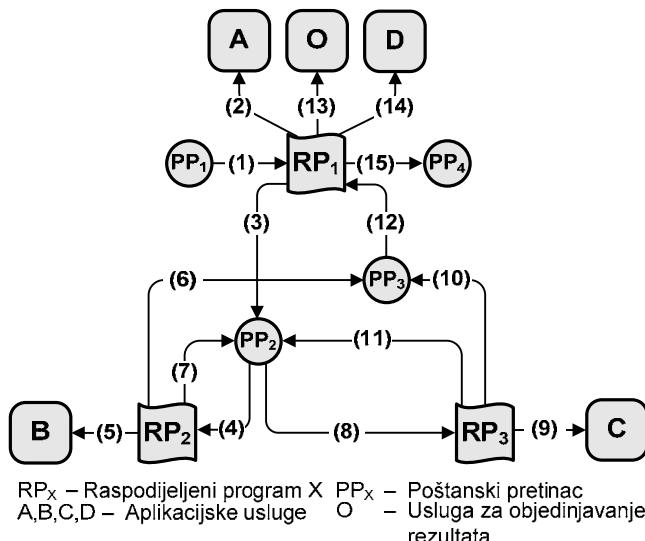
U slučaju da raspodijeljeni program RP_4 ostvaruje obrazac *diskriminatore*, raspodijeljeni program RP_4 dohvaća jednu poruku iz poštanskog pretinca PP_4 (13). Raspodijeljeni program RP_4 zatim prosljeđuje dohvaćenu poruku kao ulaz aplikacijskoj usluzi D (15), dohvaća rezultat izvođenja usluge i prosljeđuje primljeni rezultat u poštanski pretinac PP_4 (16). Sve preostale poruke spremljene u poštanskom pretincu PP_4 dohvaća i zanemaruje raspodijeljeni program RP_4 .

Neuređeno slijedno izvođenje

Obrazac *neuređeno slijedno izvođenje* primjenjuje se u slučajevima kada je potrebno koristiti skup međusobno nazavisnih aplikacijskih usluga koje je moguće izvoditi proizvoljnim redoslijedom, ali ne istovremeno.

Slika 53: Obrazac *neuređeno slijedno izvođenje*

Slika 53 prikazuje primjer obrasca *neuređeno slijedno izvođenje* koji koristi dvije aplikacijske usluge. Aplikacijska usluga *A* izvodi se prije izvođenja aplikacijskih usluga *B*, *C* i *D*. Nakon završetka izvođenja aplikacijske usluge *A*, za izvođenje se odabire aplikacijska usluga *B* ili *C*. Ako je odabrana aplikacijska usluga *B*, kao ulaz usluži prosljeđuje se rezultat izvođenja aplikacijske usluge *A*. Nakon završetka izvođenja aplikacijske usluge *B*, izvodi se aplikacijska usluga *C* kojoj se za ulaz prosljeđuje rezultat izvođenja aplikacijske usluge *A*. Nakon završetka izvođenja aplikacijske usluge *C*, rezultat izvođenja obje usluge objedinjuje se i prosljeđuje kao ulaz aplikacijskoj usluzi *D*. Osim navedenog scenarija izvođenja obrasca također je moguć scenarij izvođenja obrasca u kojem je zamijenjen redoslijed izvođenja aplikacijskih usluga *B* i *C*.

Slika 54: Ostvarenje obrasca *neuređeno slijedno izvođenje*

Slika 54 prikazuje ostvarenje obrasca *neuređeno slijedno izvođenje*. Obrazac je ostvaren primjenom četiri poštanska pretinaca (PP_1 , PP_2 , PP_3 , PP_4), tri raspodijeljena programa (RP_1 , RP_2 , RP_3) i usluge za objedinjavanje rezultata (*O*). Poštanski pretinac PP_1 sprema poruke koje sadrže ulazne podatke. Raspodijeljeni program RP_1 dohvata ulazne poruke spremljene u poštanskom pretincu PP_1 (1), prosljeđuje podatke primljene u ulaznim porukama aplikacijskoj usluzi *A* (2) i rezultate izvođenja usluge prosljeđuje u poštanski pretinac PP_2 (3). U slučaju da raspodijeljeni program RP_2 dohvati poruku spremljenu u

poštanskom pretinac PP_2 (4) prije raspodijeljenog programa RP_3 , raspodijeljeni program RP_2 prosljeđuje podatke dohvaćene iz poštanskog pretinca PP_2 aplikacijskoj usluzi B (5). Raspodijeljeni program RP_2 potom dohvaća rezultate izvođenja aplikacijske usluge B , prosljeđuje rezultate izvođenja aplikacijske usluge B u poštanski pretinac PP_3 (6) i prosljeđuje poruku primljenu iz poštanskog pretinca PP_2 ponovno u poštanski pretinac PP_2 (7). Raspodijeljeni program RP_3 zatim dohvaća poruku spremljenu u poštanskom pretincu PP_2 (8), prosljeđuje podatke primljene iz poštanskog pretinca PP_2 kao ulaz aplikacijskoj usluzi C (9), prosljeđuje rezultate izvođenja usluge u poštanski pretinac PP_3 (10), te prosljeđuje poruku primljenu iz poštanskog pretinca PP_2 ponovno u poštanski pretinac PP_2 (11). Raspodijeljeni program RP_1 dohvaća obje poruke spremljene u poštanskom pretincu PP_2 (12), te ih objedinjuje korištenjem usluge za objedinjavanje rezultata O (13). Nadalje, raspodijeljeni program RP_1 prosljeđuje objedinjene podatke aplikacijskoj usluzi D (14) i prosljeđuje rezultate izvođenja usluge u poštanski pretinac PP_4 (15). Obrnuti redoslijed izvođenja obrasca ostvaruje se ako raspodijeljeni program RP_3 uzme poruku iz poštanskog pretinca PP_2 prije raspodijeljenog programa RP_2 .

5.2 Programski jezik za suradnju i natjecanje

Programski jezik za suradnju i natjecanje (engl. *Coopetition Language*, *CL*) je jezik koji omogućava izgradnju raspodijeljenih aplikacija primjenom hibridne metode kompozicije usluga. Hibridna metoda kompozicije usluga ostvaruje se primjenom skupa raspodijeljenih programa čija je logika opisana u jeziku *CL* i zasniva se na primjeni metoda orkestracije i koreografije. Slično kao i složene usluge ostvarene primjenom metode koreografije, raspodijeljene aplikacije ostvarene primjenom jezika *CL* koriste raspodijeljeno upravljanje tijekom izvođenja. Slično kao i složene usluge ostvarene primjenom orkestracije, raspodijeljeni programi ostvareni primjenom jezika *CL* imaju mogućnost čuvanja stanja, upravljanja tijekom izvođenja i pretvorbe *XML* dokumenata.

Osnovne značajke jezika *CL* su programiranje na veliko, sažeti skup naredbi, korištenje ponovne iskoristivosti i prenosivost raspodijeljenih aplikacija. Oblikovanje i izgradnja raspodijeljenih aplikacija ostvaruje se primjenom *programiranja na veliko* (engl. *programming in the large*) [24]. *Programiranje na veliko* omogućava izgradnju raspodijeljenih aplikacija velikih razmjera koje su ostvarene povezivanjem aplikacijskih usluga u kompaktne cjeline s novim složenim funkcionalnostima. Obzirom da je osnova *programiranja na veliko* ponovno korištenje gotovih programske elemenata, načelo

programiranja na veliko ubrzava i olakšava postupke izgradnje i razvoja raspodijeljenih aplikacija.

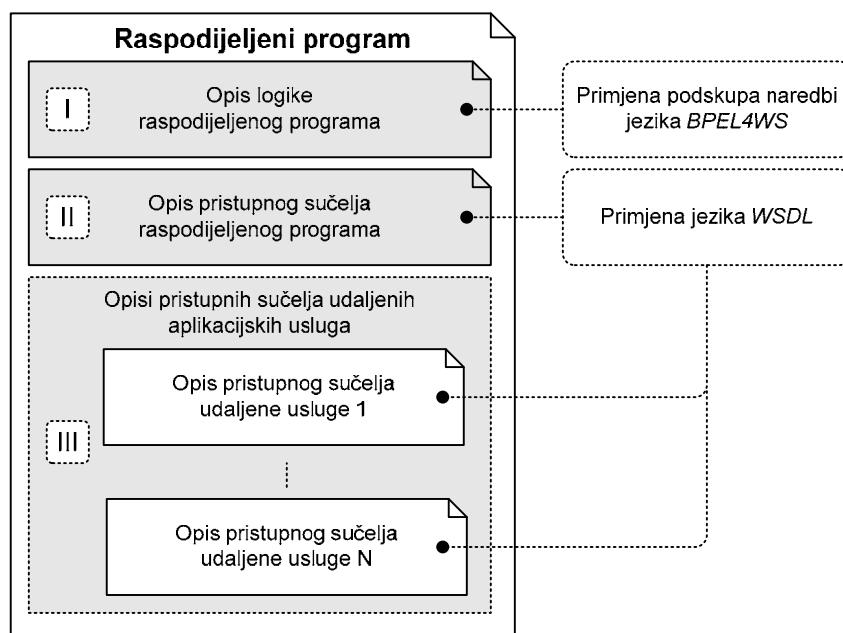
Programski jezik *CL* sadrži skup osnovnih naredbi koji uključuje naredbe poziva aplikacijskih usluga, upravljanja tijekom izvođenja i obrade podataka. Prednost primjene pojednostavljenog jezika za kompoziciju usluga je olakšana izgradnja i izvođenje raspodijeljenih aplikacija zasnovanih na uslugama. Izgradnja raspodijeljenih aplikacija pojednostavljena je obzirom da jezik *CL* ne sadrži složene naredbe za ostvarivanje istovremenog grananja tijeka izvođenja. Istovremeno izvođenje aplikacijskih usluga moguće je ostvariti korištenjem više raspodijeljenih programa koji ostvaruju međusobnu sinkronizaciju i komunikaciju primjenom koordinacijskih mehanizama. Izvođenje raspodijeljenih aplikacija ostvarenih primjenom jezika *CL* je olakšano je obzirom da za izvođenje aplikacija nije potrebno koristiti složene središnje poslužiteljske sustave za upravljanje poslovnim procesima. Raspodijeljena aplikacija izvodi se primjenom skupa jednostavnih i slabo povezanih raspodijeljenih interpretatora.

Primjena ponovne iskoristivosti omogućava jednostavniju i bržu izgradnju novih raspodijeljenih aplikacija. Ponovnu iskoristivost moguće je primijeniti za izgradnju aplikacijske i koordinacijske logike raspodijeljenih aplikacija. Za izgradnju aplikacijske logike moguće je koristiti naslijedene programske sustave čije su funkcionalnosti izložene kao usluge. Također moguće je ponovno koristiti cjelokupne prethodno ostvarene raspodijeljene aplikacije zasnovane na uslugama. Za izgradnju koordinacijske logike moguće je koristiti različite obrasce za sinkronizaciju i komunikaciju raspodijeljenih programa kao što su *slijedno izvođenje, ponavlјajuće izvođenje, usporedno grananje, sinkronizacija, uvjetno grananje, višestruko spajanje i diskriminatore*. Za izgradnju koordinacijske logike nove raspodijeljene aplikacije, postojeći se obrasci međusobno povezuju prema zahtjevima raspodijeljene aplikacije.

Jezik *CL* zasnovan je na jeziku *XML* što osigurava neovisnost raspodijeljenih programa o računalnim platformama, operacijskim sustavima i razvojnim okruženjima. Usklađenost jezika *CL* sa *Web Services* tehnologijom omogućava korištenje raznovrsnih programskih sustava čije su funkcionalnosti izložene kao usluge u globalnoj mreži Internet.

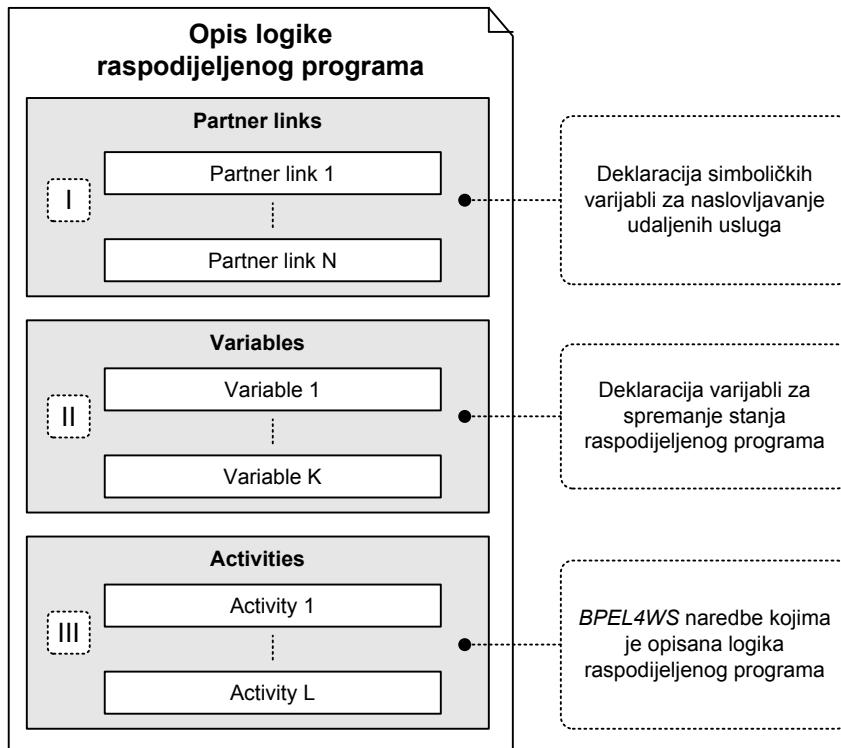
5.2.1 Struktura raspodijeljenog programa napisanog u programskom jeziku za suradnju i natjecanje

Programski jezik *CL* ostvaren je kombinacijom jezika *BPEL4WS* i *WSDL*, te specifikacija *WS-Addressing* i *WS-ResourceFramework*. U svrhu opisivanja procesne logike raspodijeljenih programa, koristi se podskup naredbi jezika *BPEL4WS*. Jezik *WSDL* koristi se za opisivanje pristupnog sučelja raspodijeljenog programa i opisivanje pristupnih sučelja udaljenih aplikacijskih usluga koje raspodijeljeni program koristi tijekom izvođenja. Specifikacija *WS-ResourceFramework* koristi se za opisivanje primjeraka raspodijeljenih programa, dok se specifikacija *WS-Addressing* koristi za naslovljavanje raspodijeljenih programa i udaljenih aplikacijskih usluga.



Slika 55: Globalna struktura raspodijeljenog programa napisanog u jeziku *CL*

Slika 55 prikazuje globalnu strukturu opisa raspodijeljenog programa ostvarenog primjenom jezika *CL*. Opis raspodijeljenog programa sadrži tri cjeline: opis procesne logike raspodijeljenog programa (I), opis pristupnog sučelja raspodijeljenog programa (II) i opisi pristupnih sučelja udaljenih aplikacijskih usluga (III).



Slika 56: Globalna struktura opisa logike raspodijeljenog programa napisanog u jeziku *CL*

Slika 56 prikazuje globalnu strukturu opisa logike raspodijeljenog programa napisanog u jeziku *CL*. Opis procesne logike raspodijeljenog programa sadrži naredbe tri bloka naredbi jezika *BPEL4WS*: *partnerLinks* (I), *variables* (II) i *activities* (III). Blok naredbi *partnerLinks* koristi se za naslovljavanje vanjskih usluga koje raspodijeljeni program koristi tijekom izvođenja. Blok naredbi *variables* omogućava deklaraciju varijabli u kojima raspodijeljeni program čuva stanje tijekom izvođenja. Blok naredbi *activities* sadrži skup naredbi koje opisuju logiku koju izvodi raspodijeljeni program tijekom izvođenja. Primjena podskupa naredbi programskog jezika *BPEL4WS* ima dvije osnovne prednosti. Prva prednost je usklađenost raspodijeljenih programa s *Web Services* tehnologijom i *WS-** skupom specifikacija. Druga prednost je mogućnost korištenja raznovrsnih javno dostupnih i široko prihvaćenih alata za izgradnju *BPEL4WS* procesa u svrhu izgradnje procesne logike raspodijeljenih programa.

Opis pristupnog sučelja raspodijeljenog programa ostvaren je primjenom jezika *WSDL*. Opis uključuje ime raspodijeljenog programa, popis operacija pristupnog sučelja, te definicije strukture i sadržaja *XML* poruka za svaku operaciju. Zahvaljujući jeziku *WSDL*, svaki raspodijeljeni program ostvaren je kao standardna usluga kojoj mogu pristupati svi korisnici usklađeni s *Web Services* tehnologijom.

Opisi pristupnih sučelja udaljenih aplikacijskih usluga uključuju definicije pristupnih sučelja svih udaljenih aplikacijskih usluga kojima raspodijeljeni program pristupa tijekom izvođenja. Za svaku udaljenu uslugu koristi se zasebni *WSDL* dokument postavljen. Opis uključuje ime usluge, popis operacija pristupnog sučelja, te definicije strukture i sadržaja *XML* poruka za svaku operaciju.

Tablica 2: *XML* struktura raspodijeljenog programa napisanog u jeziku *CL*

Struktura programa	Opis
<dpCode>	Oznaka početka programa
<process> <partnerLinks/> <variables/> activities+ </process>	Opis procesne logike raspodijeljenog programa
<processDefinitions> <definitions/> </processDefinitions>	Opis pristupnog sučelja raspodijeljenog programa
<partnerDefinitions> <definitions/> ... <definitions/> </partnerDefinitions>	Opis pristupnih sučelja udaljenih aplikacijskih usluga
</dpCode>	Oznaka kraja programa

Opis *XML* strukture podataka koja opisuje raspodijeljeni program napisan u jeziku *CL* prikazan je u tablici 2. Opis započinje s elementom **dpCode** koji predstavlja vršni element opisa. Element **dpCode** u tijelu sadrži elemente **process**, **processDefinitions** i **partnerDefinitions**. Element **process** sadrži opis procesne logike raspodijeljenog programa ostvaren primjenom podskupa naredbi jezik *BPEL4WS*. Element **processDefinitions** sadrži opis pristupnog sučelja raspodijeljenog programa izgrađenog primjenom jezika *WSDL*. Element **partnerDefinitions** sadrži opise pristupnih sučelja udaljenih usluga kojima raspodijeljeni program pristupa tijekom izvođenja. Svaki opis pristupnog sučelja udaljene usluge izgrađen je primjenom jezika *WSDL* i nalazi se u zasebnom elementu **definitions** u tijelu elementa **partnerDefinitions**.

Tablica 3: Podskup naredbi jezika *BPEL4WS* koje nasljeđuje jezik *CL*

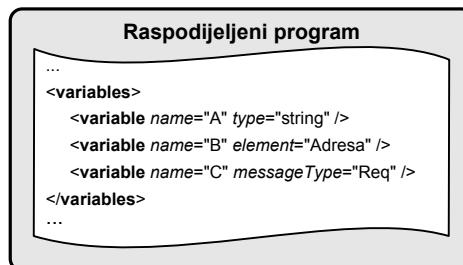
Naredbe jezika <i>CL</i>	Opis naredbe jezika <i>CL</i>
<variable/>	Naredba deklaracije varijabli u kojima raspodijeljeni program spremi stanje tijekom izvođenja
<partnerLink/>	Naredba deklaracije simboličkih imena za udaljene usluge koje raspodijeljeni program koristi tijekom izvođenja
<sequence> activities+ </sequence>	Naredba slijednog izvođenja skupa naredbi
<while/>	Naredba petlje s uvjetom ponavljanja
<invoke/>	Naredba sinkronog poziva udaljene usluge
<receive/>	Naredba primanja poruke zahtjeva tijekom poziva operacije raspodijeljenog programa
<reply/>	Naredba slanja poruke odgovora za primljenu poruku zahtjeva tijekom poziva operacije raspodijeljenog programa
<switch> <case1/> ... <caseN/> <otherwise/> </switch>	Naredba ostvarivanja uvjetnog grananja tijeka izvođenja raspodijeljenog programa
<pick> <onMessage1/> ... <onMessageN/> <onTimer/> </pick>	Naredba obrade poruka zahtjeva za dvije ili više operacija raspodijeljenog programa
<assign/>	Naredba upravljanja vrijednostima varijabli raspodijeljenog programa
<wait/>	Naredba čekanja
<terminate/>	Naredba zaustavljanja tijeka izvođenja raspodijeljenog programa
<empty/>	Naredba bez akcije

Tablica 3 sadrži opise naredbi jezika *BPEL4WS* koje je moguće koristiti za izgradnju opisa logike raspodijeljenih programa. Programske jezike *CL* i *BPEL4WS* su srodni, ali ne identični. Naredbe jezika *CL* grupirane su u četiri skupine: naredbe deklaracije varijabli i simboličkih imena usluga, naredbe komunikacije, naredbe upravljanja tijekom izvođenja i naredbe upravljanja podacima. Primjer raspodijeljenog programa napisanog u jeziku *CL* prikazan je u dodatku B.

5.2.2 Naredbe deklaracije varijabli i simboličkih imena usluga

Deklaracija varijabli koje koristi raspodijeljeni program ostvaruje se primjenom skupa naredbi *variable*. Slika 57 prikazuje primjer korištenja skupa naredbi *variable* za deklaraciju tri varijable raspodijeljenog programa. Prikazani primjer sadrži deklaraciju

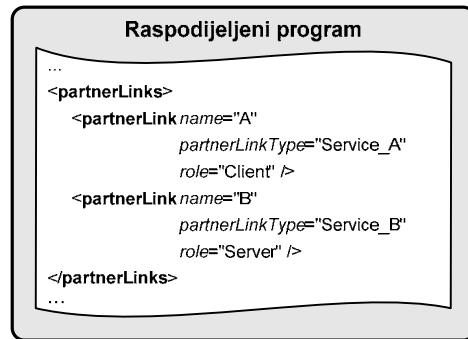
varijabli s imenima *A*, *B* i *C*. Varijabla *A* služi za spremanje znakovnih nizova, varijabla *B* služi za spremanje *XML* elemenata s imenom *Adresa*, dok varijabla *C* služi za spremanje *SOAP* poruka zahtjeva s imenom *Req*.



Slika 57: Primjer korištenja naredbe deklaracije varijabli

Naredba *variable* sadrži atribute *name*, *type* i *messageType*. Atribut *name* služi za definiranje imena varijable koja se deklarira. Uz atribut *name* moguće je u paru koristiti isključivo atribut *element*, *type* ili *messageType*. Atribut *element* služi za zadavanje *XML* elementa koji definira tip varijable. Atribut *type* služi za deklaraciju tipa varijable primjenom jednostavnih standardnih *XML Schema* tipova podataka, kao što su znakovni niz ili cijeli broj. Atribut *messageType* služi za deklaraciju tipa varijable primjenom definicije *SOAP* poruke dostupne u određenom *WSDL* dokumentu. Varijable deklarirane s *messageType* tipom služe za spremanje *SOAP* poruka zahtjeva ili odgovora koje raspodijeljeni program koristi tijekom izvođenja.

Osim deklaracije varijabli, jezik *CL* omogućava i deklaraciju simboličkih imena koja služe za naslovljavanje udaljenih usluga koje raspodijeljeni program koristi tijekom izvođenja. Deklaracija simboličkih imena ostvaruje se primjenom skupa naredbi *partnerLink*. Slika 58 prikazuje primjer deklaracije simboličkih imena *A* i *B*. Simboličko ime *A* služi za naslovljavanje aplikacijske usluge *A* primjenom komunikacijskih parametara *Service_A* i uloge korisnika koji su definirani u *WSDL* opisu sučelja raspodijeljenog programa. Simboličko ime *B* služi za naslovljavanje aplikacijske usluge *B* primjenom komunikacijskih parametara *Service_B* i uloge poslužitelja koji su definirani u *WSDL* opisu sučelja aplikacijske usluge.



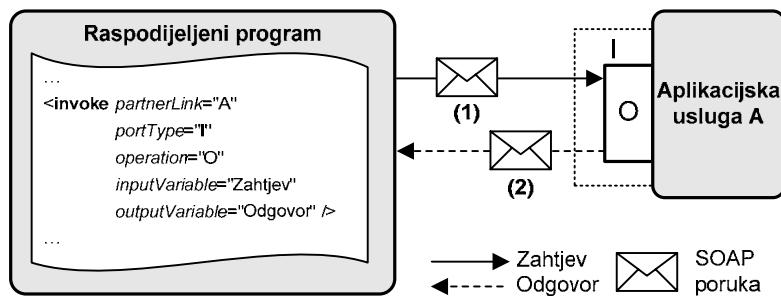
Slika 58: Primjer korištenja naredbe deklaracije simboličkih imena usluga

Naredba *partnerLink* sadrži atribute *name*, *partnerLinkType* i *role*. Atribut *name* služi za zadavanje simboličkog imena za udaljenu aplikacijsku uslugu. Atribut *partnerLinkType* služi za definiranje komunikacijskih postavki koje se koriste tijekom komunikacije s udaljenom aplikacijskom uslugom. Atribut *role* služi za definiranje uloge raspodijeljenog programa tijekom komunikacije s udaljenom aplikacijskom uslugom. Moguće uloge raspodijeljenog programa su poslužitelj, korisnik ili obje istovremeno.

5.2.3 Naredbe komunikacije

Programski jezik *CL* sadrži naredbe komunikacije koje omogućavaju razmjenu *SOAP* poruka između raspodijeljenih programa i udaljenih aplikacijskih usluga ili korisnika raspodijeljenih programa. Skup naredbi komunikacije uključuju naredbe *invoke*, *receive*, *reply* i *pick*.

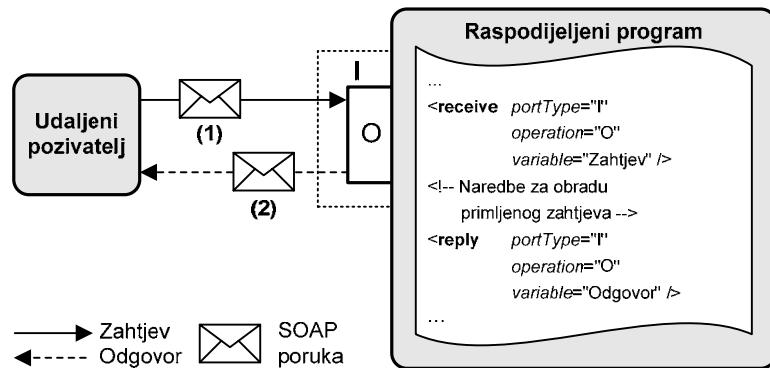
Naredbom *invoke* ostvaruju se pozivi operacija udaljenih aplikacijskih usluga. Tijekom izvođenja naredbe *invoke* raspodijeljeni program šalje *SOAP* poruku zahtjeva spremljenu u nekoj od varijabli raspodijeljenog programa i čeka primitak *SOAP* poruke odgovora. Nakon primitka *SOAP* poruke odgovora, raspodijeljeni program sprema primljenu poruku odgovora u neku od svojih varijabli. Slika 59 prikazuje primjer korištenja naredbe *invoke* u svrhu poziva operacije *O* koja se nalazi u pristupnom sučelju *I* udaljene aplikacijske usluge *A*. Tijekom izvođenja naredbe *invoke*, raspodijeljeni program usmjerava *SOAP* poruku zahtjeva spremljenu u varijabli *Zahtjev* (1). Udaljena aplikacijska usluga *A* prihvata upućenu *SOAP* poruku zahtjeva, ostvaruje obradu primljenog zahtjeva i proslijedi *SOAP* poruku odgovora raspodijeljenom programu (2). Raspodijeljeni program prihvata *SOAP* poruku odgovora i sprema primljenu poruku u varijablu *Odgovor*.



Slika 59: Primjer korištenja naredbe *invoke*

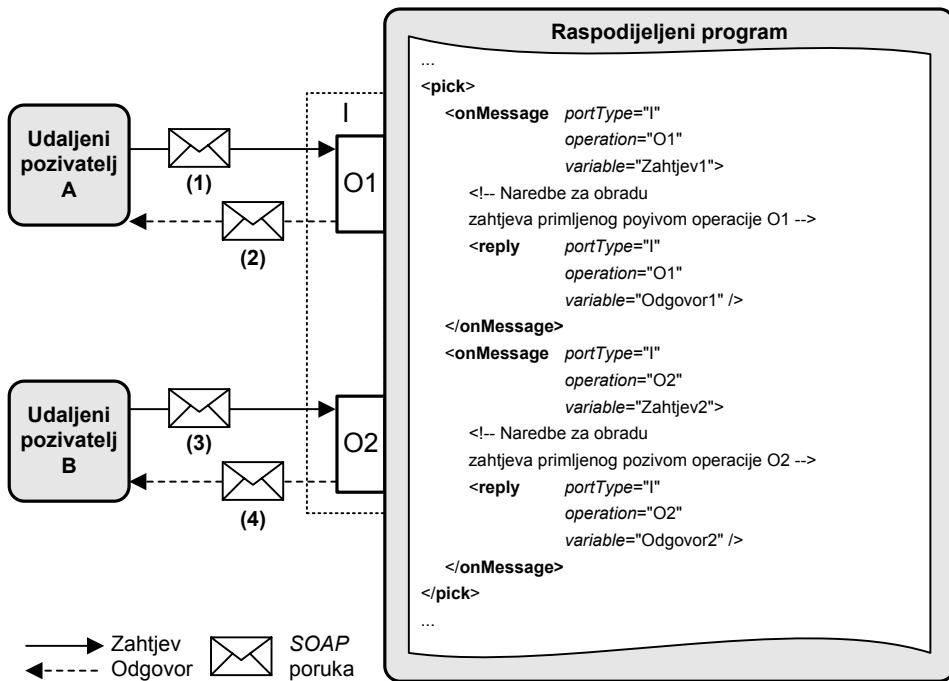
Naredba *invoke* sadrži atribute *partnerLink*, *portType*, *operation*, *inputVariable* i *outputVariable*. Atribut *partnerLink* služi za definiranje simboličkog imena udaljene aplikacijske usluge čiju operaciju raspodijeljeni program poziva. Atribut *portType* služi za definiranje imena pristupnog sučelja udaljene aplikacijske usluge u kojem se nalazi operacija koju raspodijeljeni program poziva. Atribut *operation* služi specificiranje imena operacije koju raspodijeljeni program poziva. Atribut *inputVariable* služi za definiranje imena varijable koja sadrži *SOAP* poruku zahtjeva koju raspodijeljeni program usmjerava aplikacijskoj usluzi tijekom poziva operacije. Atribut *outputVariable* služi za specificiranje imena varijable u koju se spremi *SOAP* poruka odgovora primljena od udaljene aplikacijske usluge nakon završetka izvođenja pozvane operacije.

Naredba *receive* omogućava primanje poziva određene operacije raspodijeljenog programa, a naredba *reply* omogućava slanje rezultata izvođenja te operacije. Slika 60 prikazuje primjer korištenja para naredbi *receive* i *reply* u svrhu prihvaćanja poziva operacije *O* koja se nalazi u pristupnom sučelju *I* raspodijeljenog programa. Izvođenjem naredbe *receive* raspodijeljeni program privremeno obustavlja izvođenje do trenutka poziva zadane operacije. Po primitku *SOAP* poruke zahtjeva upućene od strane udaljenog pozivatelja (1), raspodijeljeni program sprema primljenu *SOAP* poruku zahtjeva u varijablu *Zahtjev*. Nakon primitka *SOAP* poruke zahtjeva, raspodijeljeni program izvodi *CL* naredbe za obradu zahtjeva. Nakon završetka obrade, raspodijeljeni program izvodi naredbu *reply*. Naredba *reply* prosljeđuje *SOAP* poruku odgovora spremljenu u varijabli *Odgovor* udaljenom pozivatelju (2).

Slika 60: Primjer uporabe naredbi *receive* i *reply*

Naredbe *receive* i *reply* sadrže atribute *portType*, *operation* i *variable*. Atribut *portType* služi za definiranje imena pristupnog sučelja raspodijeljenog programa u kojem se nalazi operacija za koju je potrebno primiti poziv. Atribut *operation* služi za specificiranje imena operacije za koju je potrebno primiti poziv. U slučaju da se koristi naredba *receive*, atribut *variable* služi za definiranje imena varijable u koju se spremi *SOAP* poruka zahtjeva primljena tijekom poziva operacije. Ako se koristi naredba *reply*, atribut *variable* služi za definiranje imena varijable u koju je spremljena *SOAP* poruka odgovora koju je potrebno proslijediti kao rezultat izvođenja pozvane operacije udaljenom pozivatelju.

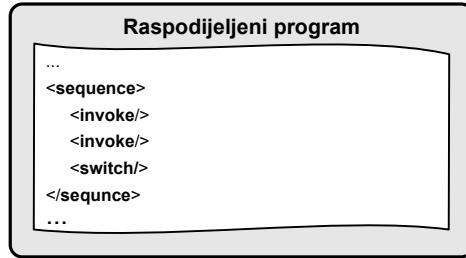
Naredba *pick* omogućava prihvatanje poziva jedne od konačnog skupa različitih operacija raspodijeljenog programa. Slika 61 prikazuje primjer uporabe naredbe *pick*. Prikazana naredba u svojem tijelu sadrži dva elementa *onMessage* pomoću kojih omogućavaju prihvatanje poziva operacija *O1* ili *O2*. Izvođenjem naredbe *pick*, raspodijeljeni program privremeno obustavlja izvođenje do trenutka poziva operacije *O1* ili *O2* koje se nalaze u pristupnom sučelju *I* raspodijeljenog programa. U slučaju poziva operacije *O1*, raspodijeljeni program izvodi prvu naredbu *onMessage*. Raspodijeljeni program sprema primljenu *SOAP* poruku zahtjeva u varijablu *Zahtjev1* i izvodi naredbe u tijelu prve naredbe *onMessage*. Po završetku obrade, raspodijeljeni program šalje *SOAP* poruku odgovora udaljenom pozivatelju *A* izvođenjem naredbe *reply*. Naredba *reply* šalje *SOAP* poruku odgovora spremljenu u varijabli *Odgovor1* udaljenom pozivatelju *A* (2). U slučaju poziva operacije *O2*, raspodijeljeni program na sličan način izvodi drugu naredbu *onMessage*.

Slika 61: Primjer uporabe naredbe *pick*

Naredba *pick* u svojem tijelu može sadržavati konačan broj elemenata *onMessage*. Element *onMessage* sadrži atribute *portType*, *operation* i *variable*. Atribut *portType* služi za definiranje imena pristupnog sučelja raspodijeljenog programa. Atribut *operation* služi za definiranje imena operacije za koju je potrebno prihvati poziv. Atribut *variable* služi za definiranje imena varijable u koju se spremi *SOAP* poruka zahtjeva primljena tijekom poziva zadane operacije raspodijeljenog programa. U tijelu svake naredbe *onMessage* moguće je navesti niz naredbi kojima se ostvaruje obrada pristigle *SOAP* poruke zahtjeva. Završna naredba unutar tijela svake naredbe *onMessage* je naredba *reply*, koja služi za proslijedivanje *SOAP* poruke odgovora udaljenom pozivatelju.

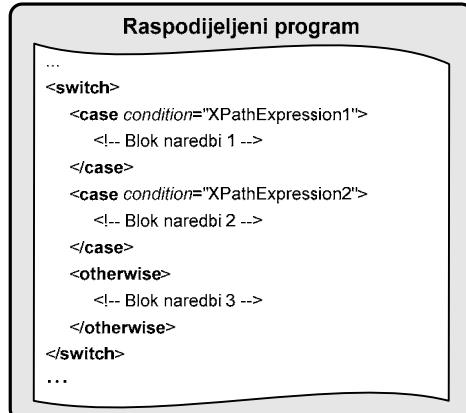
5.2.4 Naredbe upravljanja tijekom izvođenja

Programski jezik *CL* uključuje naredbe *sequence*, *switch*, *while*, *wait*, *terminate* i *empty* koje omogućavaju upravljanje tijekom izvođenja raspodijeljenih programa. Naredba *sequence* omogućava slijedno izvođenje skupa naredbi koje su zadane u njezinom tijelu. Slika 62 prikazuje primjer uporabe naredbe *sequence*. Prikazana naredba *sequence* u svojem tijelu sadrži dvije naredbe *invoke* i jednu naredbu *switch* koje se slijedno izvode redoslijedom kojim su navedene.



Slika 62: Primjer uporabe naredbe *sequence*

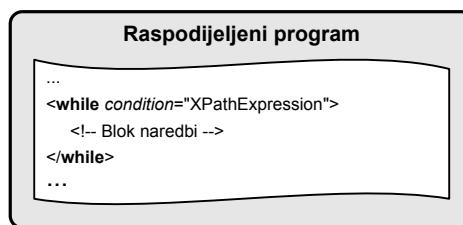
Naredba *switch* omogućava ostvarivanje uvjetnog grananja tijeka izvođenja na temelju sadržaja varijabli raspodijeljenog programa. Slika 63 prikazuje primjer korištenja naredbe *switch* koja sadrži dvije *case* naredbe i naredbu *otherwise*. U slučaju da se tijekom izvođenja naredbe *switch* booleov izraz *XPathExpression1* razrješava kao logički istinita tvrdnja, izvodi se blok naredbi u tijelu prve *case* naredbe. U slučaju da se Booleov izraz *XPathExpression1* razrješava kao logički neistinita tvrdnja i Booleov izraz *XPathExpression2* razrješava kao logički istinita tvrdnja, izvodi se blok naredbi u tijelu druge *case* naredbe. Konačno, u slučaju da se tijekom izvođenja naredbe *switch* oba izraza *XPathExpression1* i *XPathExpression2* razrješavaju kao logički neistiniti tvrdnje, izvodi se blok naredbi u tijelu naredbe *otherwise*.



Slika 63: Primjer uporabe naredbe *switch*

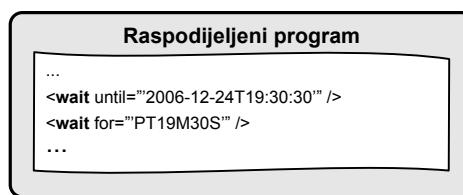
U tijelu naredbe *switch* nalazi se niz naredbi *case* i neobavezna naredba *otherwise*. Svaka naredba *case* sadrži atribut *condition* koji sadrži Booleov izraz napisan primjenom upitnog jezika XPath. Nadalje, svaka naredba *case* u svojem tijelu sadrži skup naredbi. Tijekom izvođenja naredbe *switch*, izvodi se prva od navedenih naredbi *case* čiji se Booleov izraz razrješava kao logički istinita tvrdnja. U slučaju da ne postoji naredba *case* čiji se Booleov uvjet razrješava kao logički istinita tvrdnja, a zadana je naredba *otherwise*, izvode se naredbe u tijelu naredbe *otherwise*.

Naredba *while* omogućava ostvarivanje petlje s ispitivanjem uvjeta ponavljanja na početku. Naredba sadrži atribut *condition* koji sadrži Booleov izraz napisan u jeziku XPath. Zadani izraz određuje uvjet ponavljanja bloka naredbi u tijelu naredbe *while*. Slika 64 prikazuje primjer uporabe naredbe *while*. Na početku izvođenja naredbe *while* ispituje se uvjet ponavljanja petlje zadan izrazom *XPathExpression*. U slučaju da se izraz razrješava kao logički neistinita tvrdnja, izvode se naredbe nakon naredbe *while*. U protivnom, ako se izraz razrješava kao logički istinita tvrdnja, izvodi se blok naredbi u njezinom tijelu. Nakon završetka izvođenja bloka naredbi ponovno se izračunava Booleov izraz *XPathExpression* i opisani postupak se ponavlja.



Slika 64: Primjer uporabe naredbe *while*

Naredba *wait* omogućava privremeno zaustavljanje tijeka izvođenja raspodijeljenog programa do isteka zadano vremenskog razdoblja ili dostizanja određenog trenutka u vremenu. Slika 65 prikazuje primjer uporabe oba oblika naredbe *wait*. Prva naredba *wait* ostvaruje zaustavljanje tijeka izvođenja raspodijeljenog programa do 24. prosinca 2006. godine u 19 sati, 30 minuta i 30 sekundi. Druga naredba *wait* ostvaruje zaustavljanje tijeka izvođenja raspodijeljenog programa za 19 minuta i 30 sekundi.



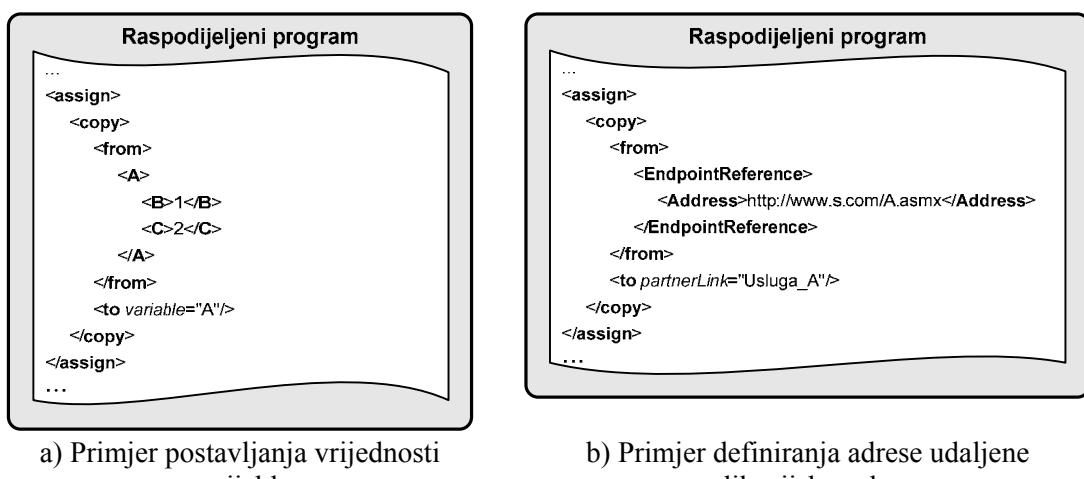
Slika 65: Primjer uporabe naredbe *wait*

Naredba sadrži atribute *until* i *for* koji se koriste međusobno isključivo. Atribut *until* služi za specificiranje trenutka u vremenu do kojeg će tijek izvođenja biti zaustavljen. Atribut *for* služi za definiranje vremenskog razdoblja za vrijeme kojeg je tijek izvođenja zaustavljen. Vremenski trenutak i razdoblje vremena zadaju se u obliku koji propisuje specifikacija za vremenske tipove podataka jezika XML [120].

Naredba *terminate* omogućava trajno zaustavljanje izvođenja raspodijeljenog programa. Naredba *empty* služi za ostvarivanje akcije bez učinaka tijekom izvođenja raspodijeljenog programa.

5.2.5 Naredbe upravljanja podacima

Naredbe za upravljanje podacima omogućavaju postavljanje i mijenjanje vrijednosti varijabli i definiranje odredišnih adresa udaljenih aplikacijskih usluga koje raspodijeljeni program poziva tijekom izvođenja. Upravljanje podacima u jeziku *CL* omogućava naredbu *assign*.

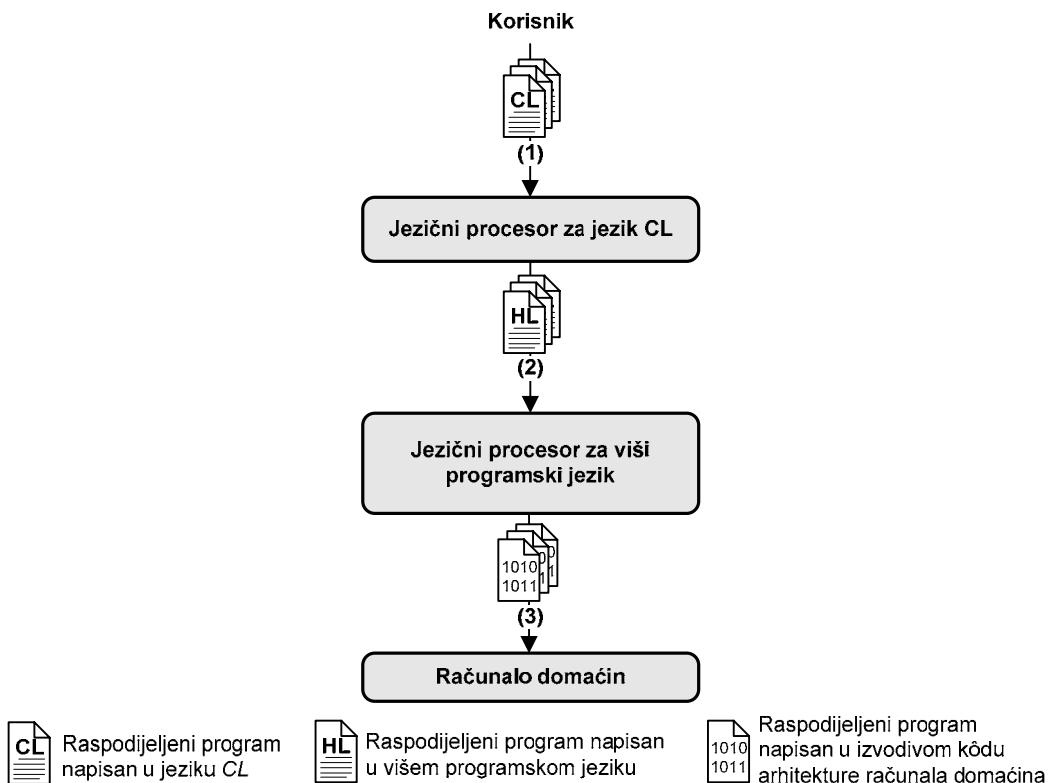


Slika 66: Primjeri korištenja naredbe *assign*

Slika 66 prikazuje primjere različitih načina uporabe naredbe *assign*. Slika 66.a) prikazuje primjer uporabe naredbe *assign* za postavljanje vrijednosti varijable *A*. Naredba *assign* u svom tijelu sadrži naredbu *copy* koja sadrži elemente *from* i *to*. Element *from* omogućava definiranje izvorišta podataka, dok element *to* služi za definiranje odredišta podataka. U prikazanom primjeru, izvorište podataka je *XML* dokument koji se postavlja kao vrijednost varijable *A*. Slika 66.b) prikazuje primjer uporabe naredbe *assign* u svrhu definiranja odredišne adrese udaljene aplikacijske usluge sa simboličkim imenom *Usluga_A*. Adresa odredišne usluge postavlja se tako da se za izvorište naredbe *copy* koristi *EndpointReference* struktura podataka koju definira specifikacija *WS-Addressing*. Struktura *EndpointReference* sadrži element *Address* koji sadrži fizičku adresu udaljene aplikacijske usluge. U prikazanom primjeru, za adresu udaljene usluge postavljena je fizička adresa *http://www.s.com/A.asmx*.

5.3 Višestupanjsko prevođenje programa

Raspodijeljene aplikacije zasnovane na uslugama razvijene uporabom jezika *CL* izvode se primjenom raspodijeljenog interpretatora programa koji je postavljen na skup radnih računala domaćina. Kako bi ostvario izvođenje raspodijeljenih aplikacija, raspodijeljeni interpretator primjenjuje postupak prevođenja raspodijeljenih programa napisanih u jeziku *CL* u izvodivi kôd arhitekture računala domaćina. Zbog visoke razine apstrakcije i složenosti semantičkih svojstava naredbi jezika *CL*, izravno prevođenje raspodijeljenih programa iz jezika *CL* u kôd arhitekture računala domaćina iznimno je složen proces. Kako bi se olakšao postupak izgradnje raspodijeljenog interpretiranja programa, primjenjuje se postupak *višestupanjskog prevođenja programa*. (engl. *multi-tier program translation*).



Slika 67: Postupak višestupanjskog prevođenje programa

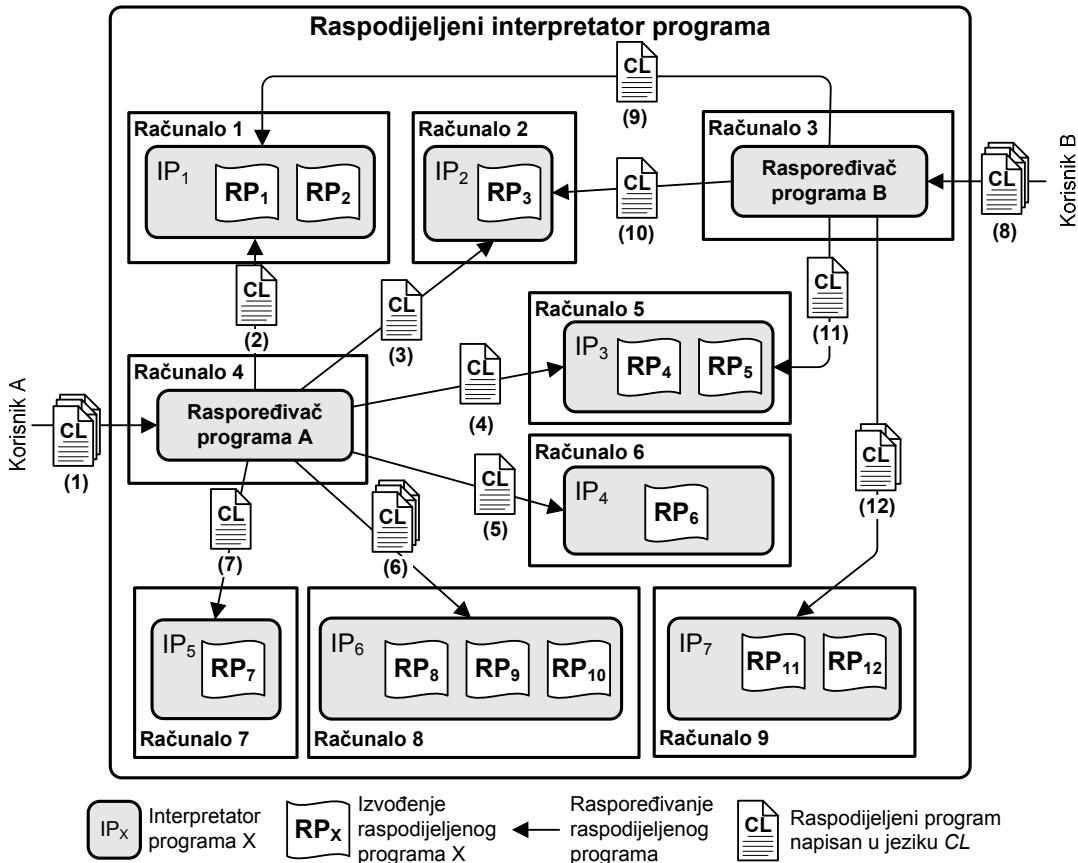
Slika 67 prikazuje postupak *višestupanjskog prevođenja programa* koji je ostvaren po uzoru na postupak prevođenja *spremi-i-pokreni* (engl. *just in time*) [121]. Postupak prevođenja započinje nakon što korisnik izgradi raspodijeljene programe u jeziku *CL*. Raspodijeljeni programi proslijedu se jezičnom procesoru za jezik *CL* (1). Jezični procesor za jezik *CL* prevodi primljene raspodijeljene programe u raspodijeljene programe napisane u višem programskom jeziku. Za odredišni viši programski jezik moguće je odabrati bilo koji

od viših programskih jezika, kao što su *C*, *C++*, *Java* ili *C#*. Nakon završetka postupka prevodenja, dobiveni raspodijeljeni programi napisani u višem programskom jeziku proslijedu se jezičnom procesoru za odabrani viši programski jezik (2). Jezični procesor za viši programski jezik prevodi raspodijeljene programe u izvodivi kôd arhitekture računala domaćina (3). Izvođenje primljenih raspodijeljenih programa ostvaruje se izvođenjem izgrađenog kôda na računalu domaćinu.

Primjena postupka *višestupanjskog prevodenja programa* olakšava i ubrzava izgradnju raspodijeljenog interpretatora programa. Za izgradnju raspodijeljenog interpretatora poželjno je odabratи viši programski jezik za koji su dostupne programske knjižnice koje ostvaruju potporu za *XML* i *Web Services* tehnologiju. Primjena postojećih *XML* i *Web Services* knjižnica značajno olakšava i ubrzava izgradnju postupka prevodenja raspodijeljenih programa napisanih u jeziku *CL*. Prevođenje raspodijeljenih programa provodi se izravnim preslikavanjem svake naredbe raspodijeljenog programa napisanog u jeziku *CL* u odgovarajući skup naredbi višeg programskog jezika. Osim potpore za *XML* i *Web Services* tehnologije, poželjno je odabratи široko prihvaćeni viši programski jezik za koji su dostupni jezični procesori za raznovrsne računalne platforme i operacijske sustave. Mogućnost primjene postojećih jezičnih procesora dodatno ubrzava postupak izgradnje raspodijeljenog interpretatora i omogućava njegovo korištenje na raznovrsnim računalnim platformama i operacijskim sustavima.

5.4 Elementi raspodijeljenog interpretatora programa

Raspodijeljeni interpretator programa ostvaruje raspodijeljeno okruženje za raspoređivanje, prevodenje i usporedno interpretiranje raspodijeljenih aplikacija zasnovanih na uslugama koje su ostvarene primjenom jezika *CL*. Gradivne elemente raspodijeljenog interpretatora programa čine *Raspoređivači programa* i *Interpretatori programa*. *Raspoređivači programa* ostvaruju postupak raspoređivanja (engl. *scheduling*) raspodijeljenih programa napisanih u jeziku *CL* s ciljem izvođenja raspodijeljenih aplikacija. *Interpretatori programa* izvode raspodijeljene programe napisane u programskom jeziku *CL* na računalima domaćinima. Raspodijeljeni interpretator programa može sadržavati proizvoljan broj *Raspoređivača programa* i *Interpretatora programa* postavljenih na skupu radnih računala.

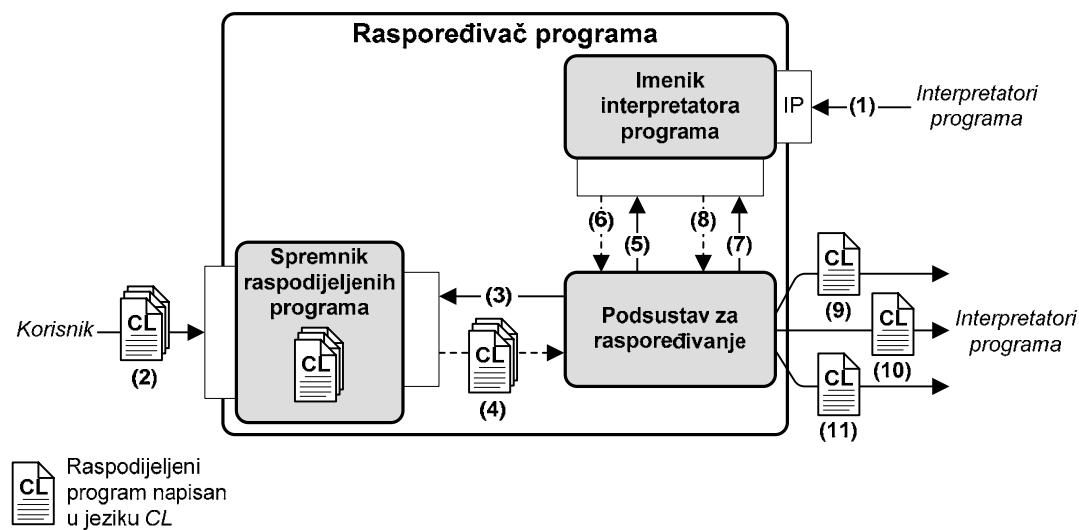


Slika 68: Primjer raspodijeljenog interpretatora programa

Slika 68 prikazuje primjer ustroja raspodijeljenog interpretatora programa postavljenog na devet radnih računala. Prikazani raspodijeljeni interpretator sadrži dva nezavisna *Raspoređivača programa*, postavljena na računalo 3 i računalo 4. Nadalje, raspodijeljeni interpretator sadrži sedam nezavisnih *Interpretatora programa* (IP_1 - IP_7) koji su postavljeni na sedam različitih računala. *Raspoređivač programa A* prihvata raspodijeljene programe napisane u programskom jeziku *CL* koji ostvaruju raspodijeljene aplikacije korisnika *A* (1). Nadalje, *Raspoređivač programa A* raspoređuje primljene raspodijeljene programe *Interpretatorima programa* IP_1 , IP_2 , IP_3 , IP_4 , IP_5 i IP_6 kako bi ostvario izvođenje primljenih raspodijeljenih programi (2-7). *Interpretatori programa* prihvataju raspoređene raspodijeljene programe napisane u jeziku *CL* i usporedno ih izvode na računalima domaćinima 1, 2, 5, 6, 7 i 8. *Raspoređivač programa B* prima raspodijeljene programe napisane u jeziku *CL* koji ostvaruju korisničke raspodijeljene aplikacije korisnika *B* (8). *Raspoređivač programa B* raspoređuje primljene raspodijeljene programe za izvođenje *interpretatorima programa* IP_1 , IP_2 , IP_3 i IP_7 (9-12). *Interpretatori programa* prihvataju raspoređene raspodijeljene programe napisane u jeziku *CL* i usporedno ih izvode na računalima domaćinima 1, 2, 5 i 9.

5.5 Arhitektura raspoređivača programa

Raspoređivač programa ostvaruje postupak za raspoređivanje raspodijeljenih programa napisanih u jeziku *CL* na dostupne *Interpretatore programa*. *Raspoređivač programa* može biti ostvaren primjenom proizvoljnog postupka za raspoređivanje raspodijeljenih programa [122] [123]. Odluke tijekom raspoređivanja mogu biti zasnovane na informacijama o trenutnom opterećenju radnih računala, količini radne memorije radnih računala, propusnosti mreže i ostalim informacijama dostupnim tijekom izvođenja raspodijeljenog interpretatora programa. Primjena odgovarajućeg postupka za raspoređivanje omogućava učinkovito korištenje računalnih sredstava radnih računala raspodijeljenog interpretatora i učinkovito izvođenje raspodijeljenih aplikacija.



Slika 69: Arhitektura podsustava *Raspoređivač programa*

Slika 69 prikazuje arhitekturu *Raspoređivač programa* koja sadrži *Spremnik raspodijeljenih programa*, *Imenik interpretatora programa* i *Podsustav za raspoređivanje*. Raspodijeljeni programi napisani u jeziku *CL* koji se raspoređuju i izvode primjenom dostupnih *Interpretatora programa* spremljeni su u *Spremnik raspodijeljenih programa*. Informacije o značajkama *Interpretatora programa* koje su potrebne za provođenje postupka raspoređivanja spremljene su u *Imeniku interpretatora programa*. Logiku za raspoređivanje raspodijeljenih programa ostvaruje *Podsustav za raspoređivanje*.

Arhitektura *Raspoređivač programa* omogućava ostvarivanje gurni (engl. *push*) načina raspoređivanja ili povuci (engl. *pull*) načina raspoređivanja. Gurni način raspoređivanja primjenjuje se u slučajevima kada se želi ostvariti strogo upravljanje nad postupkom izvođenja raspodijeljenih aplikacija. Primjenom gurni načina raspoređivanja

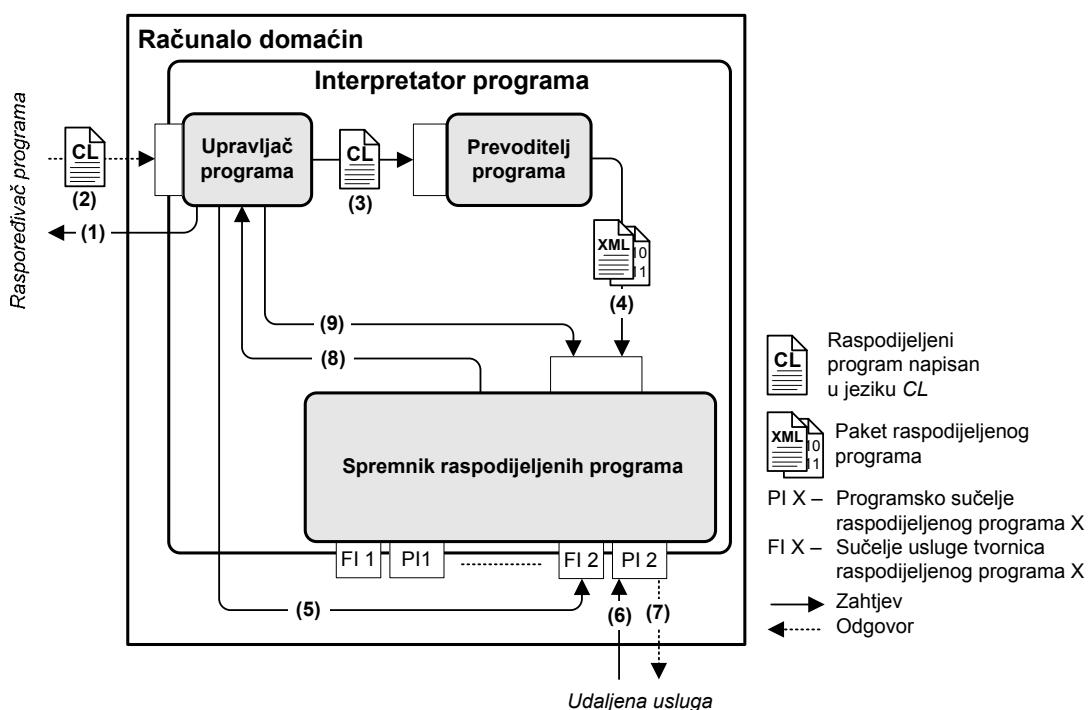
ostvaruje se raspodijeljeni interpretator programa s čvrsto povezanim *Interpretatorima programa*. Gurni način raspoređivanja provodi se tako da svaki *Interpretator programa* periodički osvježava vlastite značajke u *Imeniku interpretatora programa* (1). Zahvaljujući periodičkom osvježavanju značajki, *Rasporedišač programa* uvijek imao na raspolaganju najtočnije i najnovije značajke za svaki dostupni *Interpretator programa*. Nakon prispijeća raspodijeljenih programa u *Spremnik raspodijeljenih programa* (2), *Podsustav za raspoređivanje* dohvaća spremljene raspodijeljene programe (3, 4). Nadalje, *Podsustav za raspoređivanje* dohvaća trenutno važeće značajke za svaki prijavljeni *Interpretator programa* (5, 6) na osnovu kojih provodi postupak raspoređivanja raspodijeljenih programa (9, 10, 11).

Primjenom povuci načina raspoređivanja moguće je ostvariti raspodijeljene interpretatore programa koji sadrže veći broj slabo povezanih *Interpretatora programa* i imaju svojstvo razmjerog rasta. Povuci način raspoređivanja ostvaruje se tako da svaki *Interpretator programa* samostalno zahtjeva raspodijeljeni program koji je potrebno izvesti. U trenutku kada *Interpretator programa* raspolaže sa dovoljnom količinom sredstava na računalu domaćinu, interpretator ima mogućnost zatražiti raspodijeljeni program za izvođenje. Kako bi dohvatio raspodijeljeni program koji je potrebno izvesti, *Interpretator programa* prijavljuje vlastite značajke u *Imenik interpretatora programa* (1). Nakon prispijeća raspodijeljenih programa u *Spremnik raspodijeljenih programa* (2), *Podsustav za raspoređivanje* dohvaća spremljene raspodijeljene programe (3, 4). Nadalje, *Podsustav za raspoređivanje* dohvaća značajke za sve *Interpretatore programa* koji zahtijevaju primitak raspodijeljenog programa (5, 6). Primjenom dohvácenih značajki, interpretator programa odabire skup *Interpretatora programa* koji će izvoditi primljene raspodijeljene programe. Nakon odabira *Interpretatora programa*, *Podsustav za raspoređivanje* uklanja zapise za odabrane interpretatore iz *Imenika interpretatora programa* (7, 8) i provodi postupak raspoređivanja raspodijeljenih programa (9, 10, 11). Nakon što pojedini *Interpretator programa* izvede primljeni raspodijeljeni program, *Interpretator programa* ima mogućnost tražiti sljedeći raspodijeljeni programa ponavljanjem opisanog postupka.

5.6 Arhitektura interpretatora programa

Interpretator programa omogućava istovremeno izvođenje konačnog broja raspodijeljenih programa napisanih u jeziku *CL* na računalu domaćinu. Osnovni podsustavi arhitekture *Interpretatora programa* su *Upravljač programa*, *Prevoditelj programa* i *Spremnik raspodijeljenih programa*. Podsustav *Upravljač programa* prima raspodijeljene

programe napisane u jeziku *CL* od *Raspoređivača programa*, te upravlja postupkom prevodenja i izvođenja raspodijeljenih programa na računalu domaćinu. Podsustav *Prevoditelj programa* ostvaruje prevodenje raspodijeljenih programa napisanih u jeziku *CL* primjenom postupka *višestupanjskog prevodenja programa*. Rezultat postupka prevodenja je *paket raspodijeljenog programa* koji sadrži datoteku s izvršnim kôdom raspodijeljenog programa i *XML* datoteku s postavkama potrebnim za izvođenje raspodijeljenog programa. Izvršna datoteka sadrži kôd koji ostvaruje logiku raspodijeljenog programa i moguće ga je izvoditi na računalu domaćinu. *XML* datoteka s postavkama sadrži sve postavke koje su potrebne tijekom izvođenja raspodijeljenog programa na računalu domaćinu. Podsustav *Spremnik raspodijeljenih programa* omogućava istovremeno spremanje i izvođenje konačnog broja raspodijeljenih programa na računalu domaćinu.



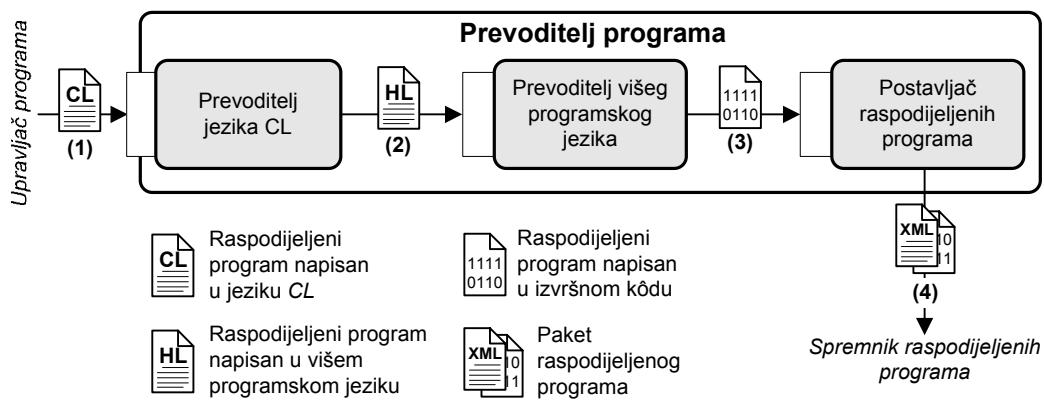
Slika 70: Arhitektura Interpretatora programa

Slika 70 prikazuje arhitekturu *Interpretatora programa*. Postupak interpretiranja raspodijeljenog programa započinje *Upravljač programu* koji od *Raspoređivača programa* dohvaća raspodijeljeni program napisan u jeziku *CL* namijenjen za izvođenje na računalu domaćinu (1, 2). Nakon primjera raspodijeljenog programa, *Upravljač programa* proslijedi primljeni raspodijeljeni program *Prevoditelju programa* (3). *Prevoditelj programa* prevodi primljeni raspodijeljeni program napisan u jeziku *CL*, gradi *paket raspodijeljenog programa* i proslijede izgrađeni *paket raspodijeljenog programa* spremniku *raspodijeljenih programa* (4). *Spremnik raspodijeljenih programa* sprema primljeni *paket raspodijeljenog programa*,

te izlaže sučelje tvornice i programsko sučelje raspodijeljenog programa. Nakon uspješnog spremanja raspodijeljenog programa, *upravljač programa* putem sučelja tvornice raspodijeljenog programa započinje izvođenje primjera raspodijeljenog programa (5). Tijekom izvođenja, primjerak raspodijeljenog programa ostvaruje komunikaciju s vanjskim udaljenim uslugama putem programskog sučelja (6, 7). Nakon završetka izvođenja, *Spremnik raspodijeljenih programa* dojavljuje završetak izvođenja primjera raspodijeljenog programa *Upravljaču programa* (8). *Upravljač programa* uklanja raspodijeljeni program iz *Spremnika raspodijeljenih programa* (9). Nakon uklanjanja raspodijeljenog programa, *Upravljač programa* dohvaća novi raspodijeljeni program od *Rasporedivača programa* (1, 2) i ponavlja opisani postupak.

5.6.1 Prevoditelj programa

Podsustav *Prevoditelj programa* ostvaruje postupak prevodenja raspodijeljenih programa napisanih u jeziku CL-u izvodivi kôd, te gradi *pakete raspodijeljenih programa* primjenom postupka *višestupanjskog prevodenja programa*.



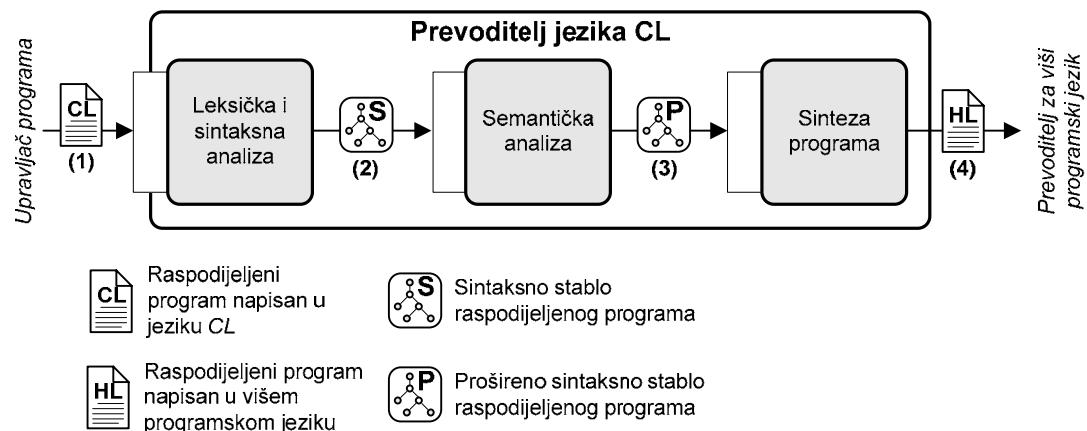
Slika 71: Arhitektura *Prevoditelja programa*

Slika 71 prikazuje arhitekturu *Prevoditelj programa* koja sadrži podsustave *Prevoditelj jezika CL*, *Prevoditelj višeg programskog jezika* i *Postavljajući raspodijeljenih programa*. *Prevoditelj jezika CL* prima raspodijeljeni program napisan u jeziku CL od *Upravljača programa* (1) i ostvaruje prevodenje primljenog raspodijeljenog programa u raspodijeljeni program napisan u višem programskom jeziku. Nakon završetka prevodenja, *Prevoditelj jezika CL* prosljeđuje izgrađeni raspodijeljeni program napisan u višem programskom jeziku *Prevoditelju višeg programskog jezika* (2). *Prevoditelj višeg programskog jezika* prevodi primljeni raspodijeljeni program iz višeg programskog jezika u izvodivi kôd arhitekture računala domaćina i prosljeđuje izgrađeni izvodivi kôd podsustavu *Postavljajući raspodijeljenih programa* (3). *Postavljajući raspodijeljenih programa* korištenjem

primljenog izvodivog kôda gradi *paket raspodijeljenog programa* i sprema izgrađeni paket u *Spremnik raspodijeljenih programa* (4).

Prevoditelj jezika CL

Podsustav *Prevoditelj jezika CL* ostvaruje postupke leksičke, sintaksne i semantičke analize raspodijeljenih programa napisanih u jeziku *CL*, te gradi raspodijeljene programe napisane u višem programskom jeziku. Slika 70 prikazuje arhitekturu *Prevoditelja jezika CL* koji sadrži podsustave *Leksička i sintaksna analiza*, *Semantička analiza* i *Sinteza programa*. Podsustavi *Leksička i sintaksna analiza* i *Semantička analiza* zajednički ostvaruju fazu analize raspodijeljenih programa napisanih u programskom jeziku *CL*. Podsustav *Sinteza programa* provodi postupak izgradnje raspodijeljenog programa napisanog u višem programskom jeziku.



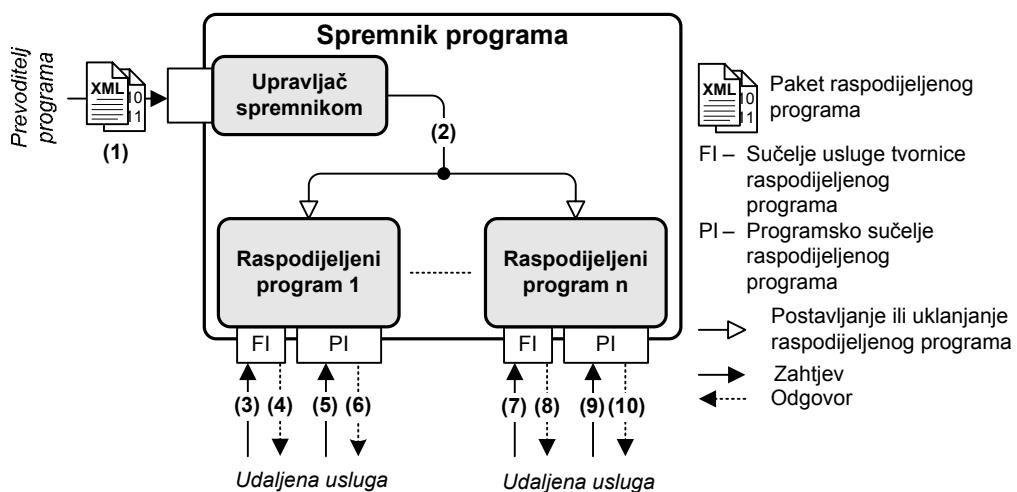
Slika 72: Arhitektura podsustava *Prevoditelj za jezik CL*

Postupak prevođenja započinje nakon što podsustav *Leksička i sintaksna analiza* primi raspodijeljeni program napisan u jeziku *CL* od podsustava *Upravljač programa* (1). Podsustav *Leksička i sintaksna analiza* prihvata raspodijeljeni program, provodi leksičku i sintaku analizu, gradi sintaksno stablo raspodijeljenog programa i proslijeđuje izgrađeno sintaksno stablo podsustavu *Semantička analiza* (2). Podsustav *Semantička analiza* prima sintaksno stablo raspodijeljenog programa, provodi postupak semantičke analize, gradi prošireno sintaksno stablo raspodijeljenog programa i proslijeđuje izgrađeno prošireno sintaksno stablo podsustavu *Sinteza programa* (3). Izgrađeno prošireno stablo je struktura podataka nastala proširivanjem sintaksnog stabla raspodijeljenog programa sa kazaljkama koje olakšavaju postupak pretraživanja stabla tijekom provođenja sinteze programa. Nakon primjeka proširenog sintaksnog stabla raspodijeljenog programa, podsustav *Sinteza programa* provodi postupak rekurzivnog spusta [7] i gradi raspodijeljeni program napisan u višem programskom jeziku. Nakon završetka postupka izgradnje, podsustav *Sinteza*

programa prosljeđuje izgrađeni raspodijeljeni program napisan u višem programskom jeziku podsustavu *Prevoditelj višeg programskog jezika* (4).

5.6.2 Spremnik programa

Podsustav *Spremnik programa* omogućava spremanje stanja raspodijeljenih programa i pristup raspodijeljenim programima koje izvodi *Interpreter programa* na računalu domaćinu. Podsustav *Spremnik programa* može istovremeno spremiti konačan broj međusobno nezavisnih raspodijeljenih programa.



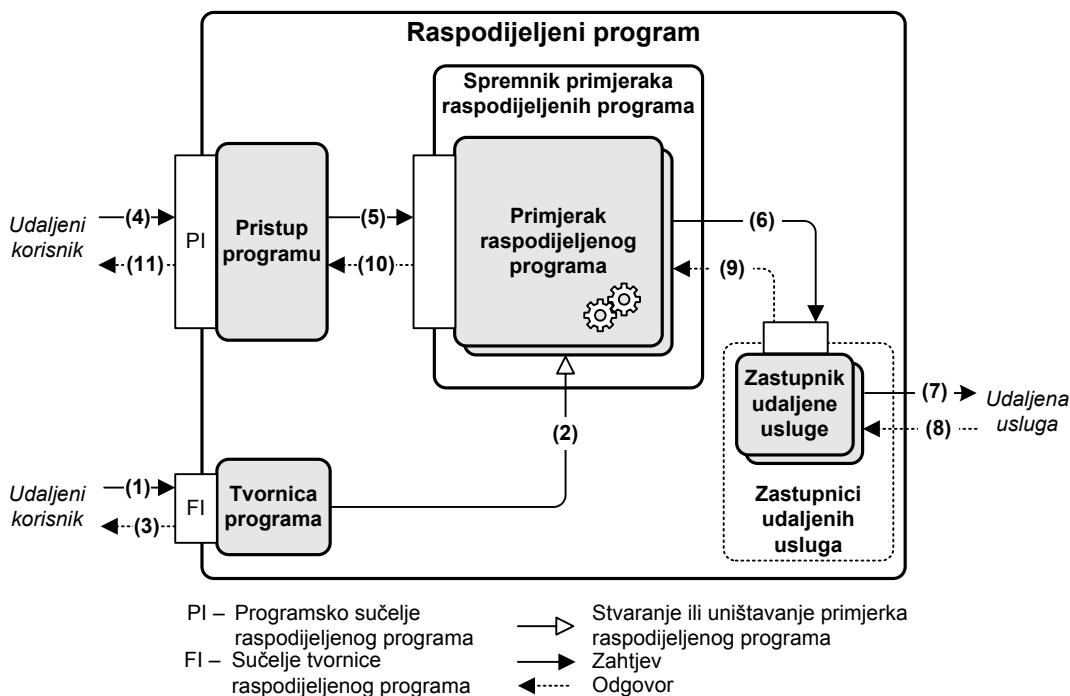
Slika 73: Arhitektura Spremnika programa

Slika 73 prikazuje arhitekturu *Spremnika programa* koja sadrži podsustave *Upravljač spremnikom* i skup spremljenih *Raspodijeljenih programa*. Podsustav *Upravljač spremnikom* omogućava spremanje i uklanjanje raspodijeljenih programa iz *Spremnika programa*. *Upravljač spremnikom* prima *paket raspodijeljenog programa* od podsustava *Prevoditelj programa* koji se sprema u *Spremnik programa* (1). *Upravljač spremnikom* zatim dohvata izvodivi kôd raspodijeljenog programa dostupan u primljenom *paketu raspodijeljenog programa* i postavlja ga u *Spremnik programa* (2). Nakon postavljanja izvodivog kôda raspodijeljenog programa u *Spremnik programa*, udaljene usluge imaju mogućnost koristiti raspodijeljeni program putem *sučelja tvornice* i *programskog sučelja* (3-10).

5.6.3 Raspodijeljeni program

Rezultat provođenja postupka sinteze je izvodivi *Raspodijeljeni program*. *Raspodijeljeni program* ostvaruje logiku za izvođenje konačnog broja primjeraka raspodijeljenih programa u odabranom višem programskom jeziku. Slika 74 prikazuje

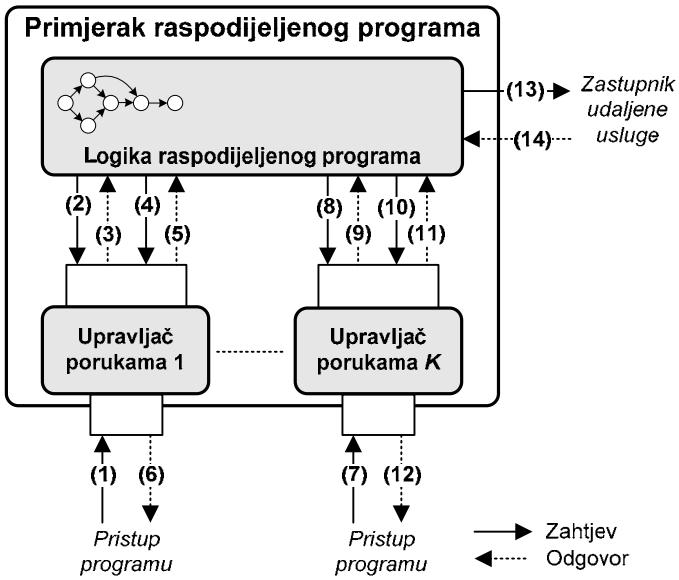
arhitekturu *Raspodijeljenog programa* koja sadrži podsustave *Pristup programu*, *Tvornica programa*, *Spremnik primjeraka raspodijeljenih programa* i skup *Zastupnika udaljenih usluga*. Podsustav *Pristupa programu* izlaže operacije koje sadrži programsko sučelje raspodijeljenog programa. Ovaj podsustav izravno koriste udaljeni korisnici tijekom komunikacije s aktivnim primjercima raspodijeljenog programa. Podsustav *Tvornice programa* izlaže operacije koje omogućavaju stvaranje i uništavanje primjeraka raspodijeljenih programa. Podsustav *Spremnika primjeraka raspodijeljenih programa* omogućava spremanje i izvođenje konačnog broja primjeraka raspodijeljenih programa. Primjeri raspodijeljenih programa spremjeni u spremniku međusobno su nezavisni i izvode logiku raspodijeljenog programa opisanu programom u jeziku *CL*. Skup *zastupnika udaljene usluge* omogućava komunikaciju s udaljenim uslugama koje primjeri raspodijeljenog programa pozivaju tijekom izvođenja.



Stvaranje primjerka raspodijeljenog programa ostvaruje se putem *Tvornice programa* (1). *Tvornica programa* sprema novi primjerak raspodijeljenog programa u *Spremnik primjeraka raspodijeljenih programa* (2) i prosljeđuje pozitivnu potvrdu udaljenom korisniku (3). Nakon spremanja u spremnik, novi primjerak raspodijeljenog programa započinje s izvođenjem logike raspodijeljenog programa. Tijekom izvođenja primjerka raspodijeljenog programa, udaljene usluge imaju mogućnost komunikacije s primjerkom raspodijeljenog programa putem podsustava *Pristup programu* (4). Svaki

zahtjev za pristup primjerku raspodijeljenog programa, uz aplikativne podatke, sadrži i oznaku koja jedinstveno određuje odredišni primjerak raspodijeljenog programa kojem se pristupa. Pristupna usluga koristi oznaku primjerka raspodijeljenog programa kako bi odabrala odgovarajući primjerak raspodijeljenog programa u *Spremniku raspodijeljenih programa* i prosljeđuje mu pristigli zahtjev (5). Odredišni primjerak raspodijeljenog programa tijekom obrade pristiglog zahtjeva ima mogućnost poziva udaljenih usluga uporabom *Zastupnika udaljenih usluga* (6). Nakon primitka zahtjeva za poziv udaljene usluge, *Zastupnik udaljene usluge* šalje poruku zahtjeva naslovljenoj udaljenoj usluzi (7), prima poruku odgovora (8) i prosljeđuje primljenu poruku odgovora primjerku raspodijeljenog programa (9). Nakon primitka poruke odgovora od *Zastupnika udaljene usluge*, primjerak raspodijeljenog programa nastavlja izvođenje, obrađuje primljenu poruku i šalje rezultate obrade podsustavu *Pristup programu* (10). Podsustav *Pristup programu* gradi poruku odgovora koja sadrži rezultate obrade primljene od primjerka raspodijeljenog programa i šalje izgrađenu poruku udaljenom korisniku (11).

Primjerak raspodijeljenog programa ostvaruje potporu za izvođenje logike primjerka raspodijeljenog programa. Slika 75 prikazuje arhitekturu primjerka raspodijeljenog programa koji sadrži podsustave *Logika raspodijeljenog programa* i skup podsustava *Upravljač porukama*. Podsustav *Logika raspodijeljenog programa* ostvaruje i izvodi logiku primjerka raspodijeljenog programa. Za svaku operaciju raspodijeljenog programa koristi se zaseban podsustav *upravljača porukama*. Zadaća podsustava *Upravljača porukama* je primanje, spremanje i prosljeđivanje poruka zahtjeva i odgovora koje primjerak raspodijeljenog programa razmjenjuje tijekom poziva operacija raspodijeljenog programa.



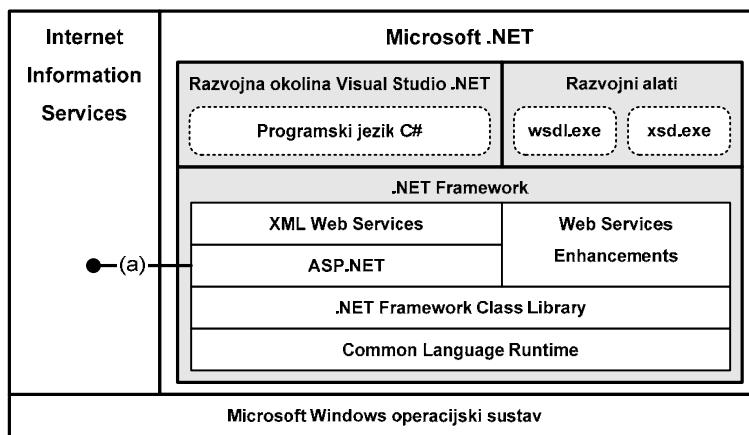
Slika 75: Arhitektura podsustava *Primjerak raspodijeljenog programa*

U slučaju poziva operacije raspodijeljenog programa za određeni *Primjerak raspodijeljenog programa*, podsustav *Upravljač poruka* prihvata i sprema poruku zahtjeva pristiglu od podsustava *Pristup programu* (1, 7). Tijekom izvođenja *Primjerka raspodijeljenog programa*, podsustav *Logika raspodijeljenog programa* prema potrebi dohvaća poruku zahtjeva koju je spremio *Upravljač porukama* (2, 3, 8, 9), obrađuje dohvaćenu poruku zahtjeva i prosljeđuje poruku odgovora *Upravljaču porukama* (4, 5, 10, 11). Nakon primitka poruke odgovora, *Upravljač porukama* prosljeđuje primljenu poruku odgovora podsustavu *Pristup programu* (6, 12). Tijekom izvođenja, podsustav *Logike raspodijeljenog programa* ima mogućnost korištenja podsustava *Zastupnika udaljene usluge*, radi pozivanja udaljenih usluga (13, 14).

6. poglavlje

Programsko ostvarenje raspodijeljenog interpretatora programa

Raspodijeljeni interpretator programa ostvaren je primjenom *Microsoft .NET* razvojnog okruženja. Osnovni elementi *Microsoft .NET* [16] razvojnog okruženja i programski sustavi koji su korišteni za izgradnju raspodijeljenog interpretatora prikazani su na slici 76.



Slika 76: Osnovni elementi razvojnog okruženja korištenog za izgradnju raspodijeljenog interpretatora

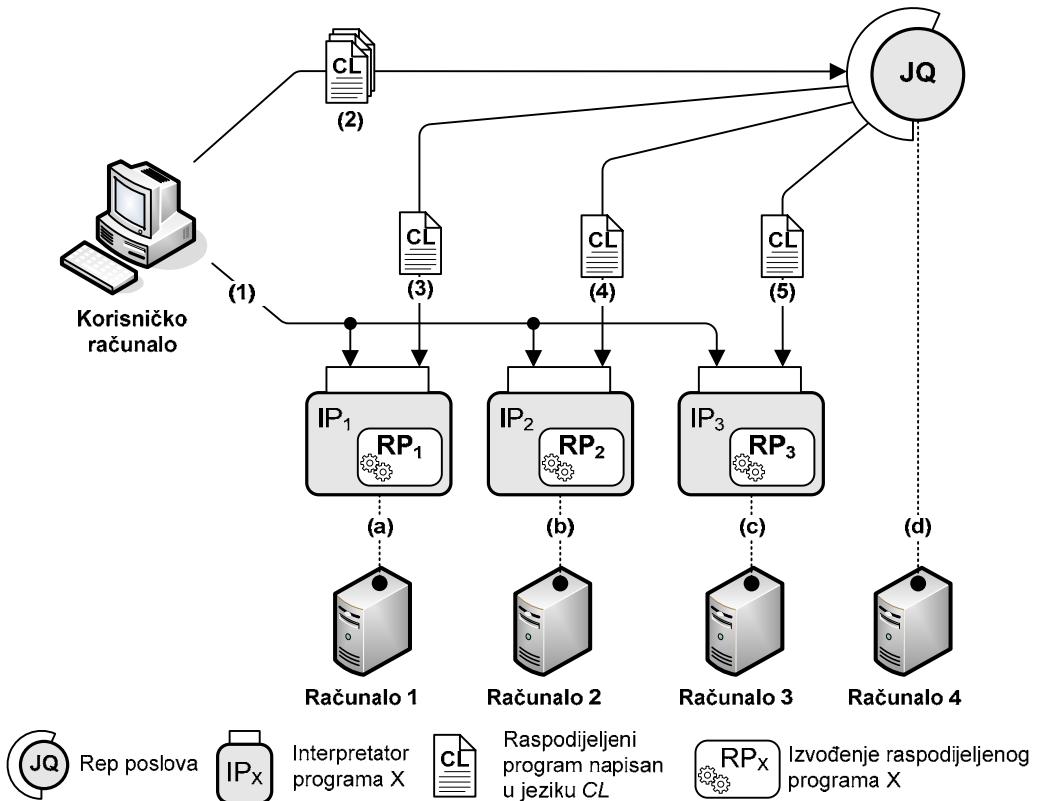
Za izgradnju raspodijeljenog interpretatora korišten je sustav *Internet Information Services* (IIS) i razvojno okruženje *Microsoft .NET*. Sustav *Internet Information Services* [126] omogućava postavljanje, izvođenje i upravljanje poslužiteljima usluga u globalnoj mreži Internet, kao što su FTP poslužitelji, mrežni poslužitelji i poslužitelji mrežnih aplikacija. Razvojno okruženje *Microsoft .NET* omogućava učinkovitu i brzu izgradnju aplikacija. Osnovni elementi razvojnog okruženja *Microsoft .NET* koji su korišteni za uzgradnju raspodijeljenog interpretatora su .NET Framework sustav, razvojna okolina *Visual Studio .NET* i skup dodatnih razvojnih alata. Sustav .NET Framework sadrži podsustave *Common Langauge Runtime* i *.NET Framework Class Library* koji čine potporu za izgradnju i izvođenje aplikacija. Podsustav *Common Langauge Runtime* ostvaruje prividni stroj (engl. *virtual machine*) koji omogućava izvođenje aplikacija napisanih u jeziku *CIL* (*Common Intermediate Langauge, CIL*). Jezik *CIL* zasnovan je na strojno nezavisnom kôdu koji je

moguće izvoditi na bilo kojoj računalnoj platformi za koju je ostvaren *Common Langauge Runtime* podsustav. Podsustav *.NET Framework Class Library* sadrži skup programskih knjižnica ostvarenih u jeziku *CIL* koje omogućavaju brzu i učinkovitu izgradnju aplikacija primjenom *Microsoft .NET* razvojnog okruženja. Podsustavi *Common Langauge Runtime* i *.NET Framework Class Library* čine osnovu za korištenje podsustava *ASP.NET* [124] i *XML Web Services*, te programske knjižnice *Web Services Enhancements (WSE)* [125]. Podsustav *ASP.NET* omogućava izgradnju i izvođenje mrežnih aplikacija ostvarenih primjenom *.NET Framework* okruženja. Mrežne aplikacije koje izvodi *ASP.NET* podsustav dostupne su na korištenje u globalnoj mreži Internet putem *Internet Information Services* sustava (a). Osnovne funkcionalnosti podsustava *ASP.NET* proširuje podsustav *XML Web Services* koji omogućava razvoj, postavljanje, izvođenje i upravljanje uslugama zasnovanim na *Web Services* tehnologiji. Programska knjižnica *WSE* ostvaruje napredne mehanizme *Web Services* tehnologije koji su definirani u *WS-** proširenom skupu specifikacija. Tijekom izgradnje raspodijeljenog interpretatora, primjenom razreda *WSE* programske knjižnice ostvarena je potpora za *WS-Addressing* specifikaciju i potpora za obradu *SOAP* poruka zahtjeva i odgovora. Programska logika raspodijeljenog interpretatora ostvarena je primjenom razreda napisanih u programskom jeziku *C#* korištenjem razvojne okoline *Visual Studio .NET*. Nadalje, tijekom razvoja korištene su i programske datoteke napisane u jeziku *C#* koje su automatski izgrađene primjenom alata *wsdl.exe* i *xsd.exe*. Alat *wsdl.exe* omogućava automatsku izgradnju datoteka napisanih u jeziku *C#* koje ostvaruju zastupnike za usluge zasnovane na *Web Services* tehnologiji. Alata *xsd.exe* omogućava prevodenje dokumenata koji sadrže *XML Schema* definicije u razrede ostvarene u jeziku *C#*. Izgrađeni razredi omogućavaju automatsko ostvarivanje leksičke i sintaksne analize *XML* dokumenata koji su opisani zadanim *XML Schema* definicijama.

6.1 Programska arhitektura raspodijeljenog interpretatora

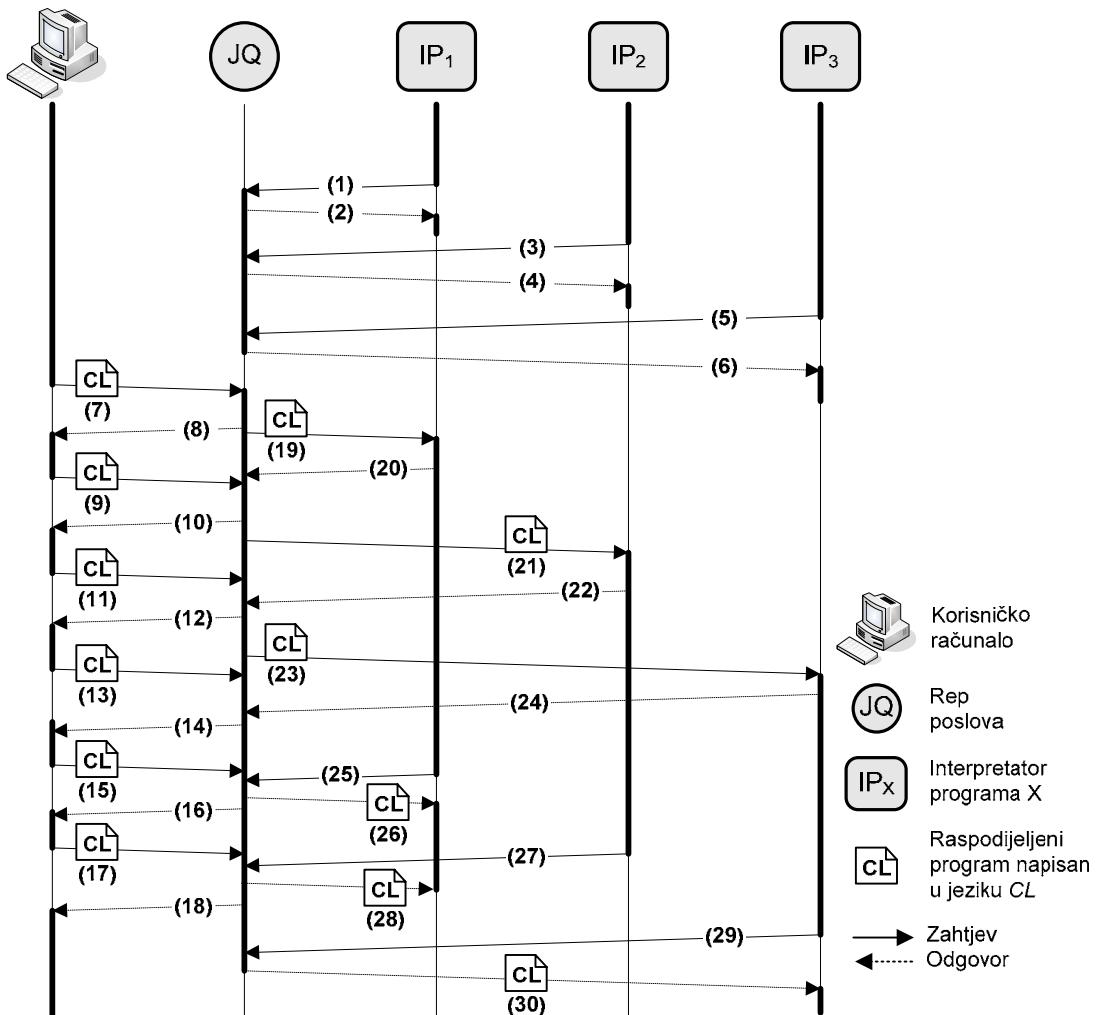
Programska arhitektura raspodijeljenog interpretatora ostvarena je na načelima arhitekture zasnovane na uslugama. Raspodijeljeni interpretator ostvaren je kao skup slabo povezanih podsustava *Interpretatora programa* i *Repa poslova* (engl. *job queue*) koji su izgradieni primjenom *Web Services* tehnologije. Ostvareni podsustavni *Interpretatori programa* međusobno se natječu za izvođenje raspodijeljenih programa napisanih u jeziku *CL* koji su spremljeni u *Repu poslova*. Primjenom skupa ostvarenih podsustava

Interpretatora programa i *Repa poslova* moguće je postaviti ostvareni raspodijeljeni interpretator programa na proizvoljan broj radnih računala.



Slika 77: Programsко ostvarenje raspodijeljenog interpretatora

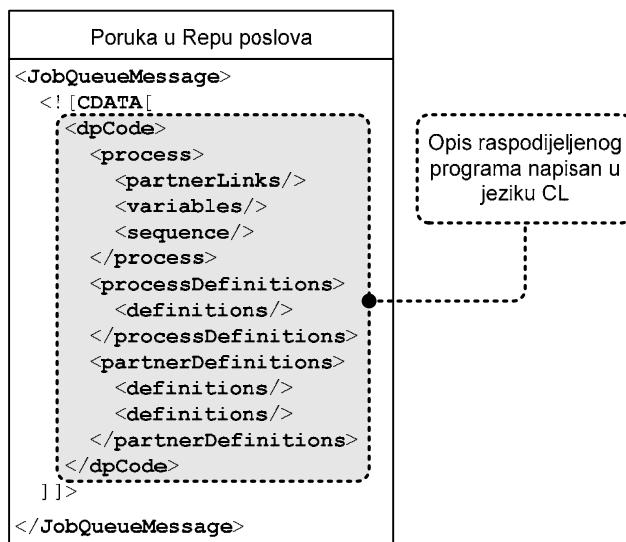
Slika 77 prikazuje ostvarene podsustave raspodijeljenog interpretatora. Prikazani raspodijeljeni interpretator sadrži tri *Interpretatora programa* (IP_1 , IP_2 , IP_3) koji se izvode kao usluge na računalima 1, 2 i 3 (a, b, c). Osim *Interpretatora programa* IP_1 , IP_2 i IP_3 , na računalo 4 postavljen je *Rep poslova* (d) koji se koristi za ostvarivanje postupka raspoređivanja raspodijeljenih programa napisanih u jeziku *CL*. Podsustav *Rep poslova* ostvaren je primjenom koordinacijskog mehanizma poštanski pretinc. Kako bi izvršio raspodijeljenu aplikaciju, korisnik pokreće *Interpretatore programa* (1) i sprema raspodijeljene programe napisane u jeziku *CL* koji ostvaruju raspodijeljenu aplikaciju u *Rep poslova* (2). Podsustavi *Interpretatori programa* IP_1 , IP_2 i IP_3 dohvataju raspodijeljene programe iz *Repa poslova* i izvršavaju ih na računalu domaćinu. Podsustav *Rep poslova* ostvaruje strategiju raspoređivanja raspodijeljenih programa prema redoslijedu dolaska (engl. *first-come-first-served*) [122].



Slika 78: Ostvareni postupak raspoređivanja raspodijeljenih programa

Slika 78 prikazuje primjer raspoređivanja raspodijeljenih programa primjenom strategije raspoređivanja prema redoslijedu dolaska. Postupak raspoređivanja započinje nakon što korisnik pokrene Interpretatore programa IP_1 , IP_2 i IP_3 . Tijekom pokretanja, svaki Interpretator programa prima adresu računala na kojem se nalazi poštanski pretinac i ime poštanskog pretinca koji ostvaruje Rep poslova. Nakon pokretanja, Interpretator programa IP_1 pokušava dohvatiti poruku iz Repa poslova primjenom modela za dohvaćanje poruka iz poštanskog pretinca zasnovanog na dojavi (1). Obzirom da je Rep poslova prazan, Rep poslova sprema adresu Interpretatora programa IP_1 i šalje mu negativnu potvrdu (2). Na opisani način Interpretatori programa IP_1 i IP_2 također pokušavaju dohvatiti poruku iz Repa poslova (3-6). Kada korisnik započne spremanje raspodijeljenih programa u Rep poslova (7-18), Rep poslova sprema primljene raspodijeljene programe Interpretatorima programa IP_1 , IP_2 i IP_3 prema redoslijedu prisjeća njihovih zahtjeva (19-30). Na početku Rep poslova koristi prethodno spremljene adrese Interpretatora programa kako bi im poslao pristigle raspodijeljene programe. Podsustav Interpretator programa IP_1 prima raspodijeljeni

program nakon što *Rep poslova* pozove njegovu operaciju za dojavu (19). Nakon primitka raspodijeljenog programa, *Interpretator program IP₁* potvrđuje primitak raspodijeljenog programa (20) i započinje njegovo izvođenje. Na opisani način *Interpretatori programa IP₂* i *IP₃* također primaju i izvode raspodijeljene programe koje im šalje *Rep poslova* (21-24). Nakon što *Interpretator programa IP₁* završi s izvođenjem primljenog raspodijeljenog programa, interpretator ponovno pokušava dohvati novi raspodijeljeni program iz *Repa poslova* (25). Obzirom da u trenutku poziva u *Repu poslova* postoje raspodijeljeni programi koje je potrebno izvesti, *Rep poslova* uklanja raspodijeljeni program koji je najduže spremlijen u repu i šalje ga *Interpretatoru programa IP₁* (26). Na opisani način, *Interpretatori programa IP₂* i *IP₃* također dohvaćaju raspodijeljene programe spremeljene u *Repu poslova* i izvode ih (27-30). *Interpretatori programa IP₁, IP₂* i *IP₃* ponavljaju opisani postupak dohvaćanja i izvođenja raspodijeljenih programa sve dok ih korisnik ne zaustavi pozivom odgovarajuće operacije njihovog pristupnog sučelja.



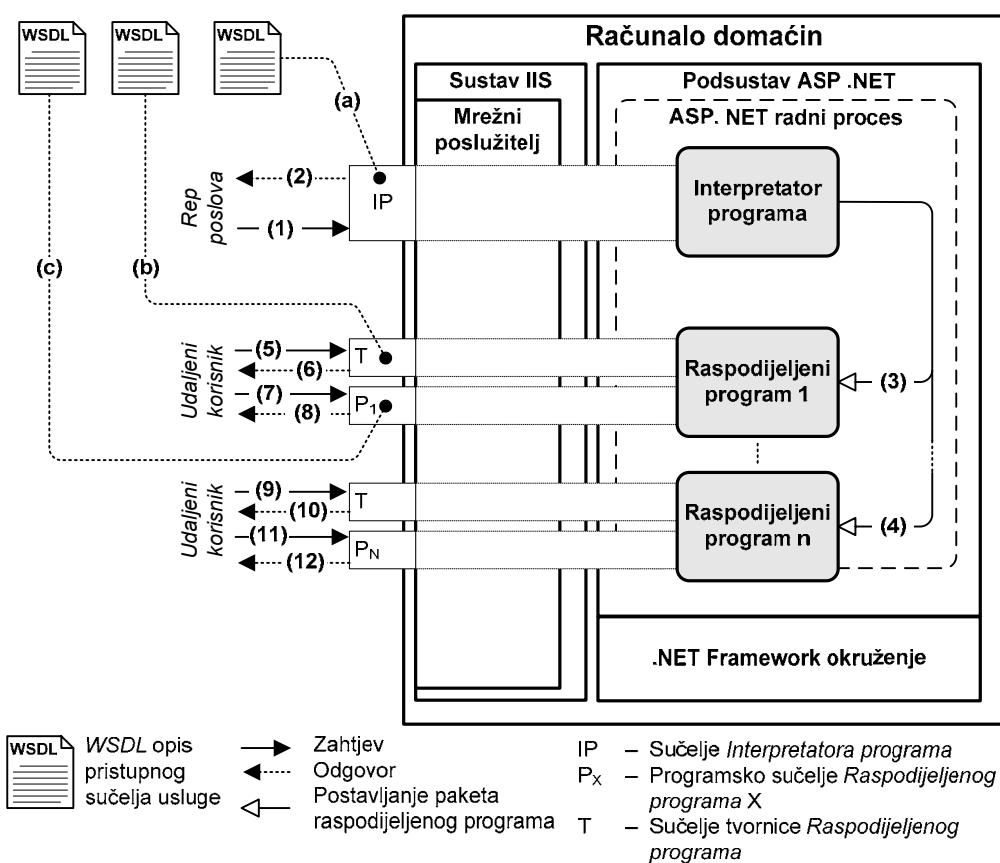
Slika 79: Primjer strukture poruke u *Repu poslova*

Svi raspodijeljeni programi napisani u jeziku *CL* spremaju se u *Rep poslova* u tekstualnom obliku. Primjer strukture poruke s raspodijeljenim programom koja je spremljena u *Repu poslova* prikazna je na slici 79. Osnovni *XML* element poruke je **JobQueueMessage** koji u svojem tijelu sadrži element **CDATA**. Element **CDATA** je standardni *XML* element koji u svojem tijelu omogućava spremanje proizvoljnih znakovnih nizova koji se ne tumače tijekom obrade *XML* dokumenta. Element **CDATA** koristi se za spremanje novog *XML* dokumenta koji sadrži opis raspodijeljenog programa napisan u programskom jeziku *CL*. Obzirom da se opis raspodijeljenog programa napisanog u jeziku

CL nalazi u tijelu elementa **CDATA**, struktura i kontekst opisa ostaje u potpunosti sačuvan tijekom prijenosa programa putem mreže.

6.1.1 Interpretator programa

Podsustav *Interpretator programa* i svi *Raspodijeljeni programi* koji se izvode na računalu domaćinu ostvareni su kao usluge zasnovane na *Web Services* tehnologiji primjenom *XML Web Services* podsustava *.NET Framework* razvojnog okruženja. Logika *Interpretatora programa* i *Raspodijeljenih programa* ostvarena je primjenom jezika *C#* i prevedena je u *CIL* kôd koji izvodi *.NET Framework okruženje* na računalu domaćinu.



Slika 80: Radno okruženje ostvarenog podsustava *Interpretator programa*

Slika 80 prikazuje radno okruženje ostvarenog podsustava *Interpretator programa*. Logiku ostvarenog podsustava *Interpretator programa* izvodi radni proces sustava *ASP.NET* (engl. *ASP.NET worker process*). Pristupno sučelje *Interpretatora programa* (*IP*) opisano je primjenom jezika *WSDL* (a) i dostupno je na korištenje putem mrežnog poslužitelja koji izvodi *Internet Information Services* sustav. Pristupno sučelje *Interpretatora programa* omogućava pokretanje i zaustavljanje interpretatora, te primanje raspodijeljenih programa iz *Repa poslova* koje je potrebno izvesti na računalu domaćinu (1, 2). Osim usluge

Interpretatora programa, na računalu domaćinu izvode se i usluge svih *Raspodijeljenih programa* koje postavlja ostvareni *Interpretator programa* (3, 4). Postavljeni *Raspodijeljeni programi* sadrže *sučelje tvornice* (T) i *programske sučelje* (P_X) koja su opisana primjenom jezika *WSDL* (b, c). Sučelja postavljenih *Raspodijeljenih programa* omogućavaju udaljenim korisnicima stvaranje, uništavanje i korištenje primjeraka postavljenih *Raspodijeljenih programa* (5-12). Primjena jezika *WSDL* omogućava raznovrsnim korisnicima u globalnoj mreži Internet korištenje ostvarenog *Interpretatora programa* i postavljenih *Raspodijeljenih programa*. *WSDL* opisi pristupnih sučelja sadrže sve informacije potrebe za ostvarivanje poziva operacija *Interpretatora programa* i *Raspodijeljenih programa* neovisno o računalnoj platformi, operacijskom sustavu ili razvojnoj okolini korisnika.

Tablica 4: Opis operacija pristupnog sučelja *IP* podsustava *Interpretator programa*

Ime operacije	Opis operacije
<code>bool Start(JobQueueEPR, Count)</code>	Operacija kojom se pokreće <i>Interpretator programa</i> . Parametar <i>JobQueueEPR</i> sadrži adresu računala na kojem se nalazi poštanski pretinac i ime primjerka poštanskog pretinca kojim je ostvaren <i>Rep poslova</i> . Parametar <i>Count</i> određuje maksimalni broj raspodijeljenih programa koje je dozvoljeno istovremeno izvoditi na računalu domaćinu. Ako je pokretanje uspješno obavljeno, operacija vraća pozitivnu potvrdu. U protivnome, operacija vraća negativnu potvrdu.
<code>bool Stop()</code>	Operacija kojom se zaustavlja <i>Interpretator programa</i> . Ako je zaustavljanje uspješno obavljeno, operacija vraća pozitivnu potvrdu. U protivnome, operacija vraća negativnu potvrdu.
<code>bool MailBoxNotify(Message, mbxEPR)</code>	Operacija kojom <i>Rep poslova</i> vraća raspodijeljeni program napisan u jeziku <i>CL Interpretatoru programa</i> na izvođenje. Parametar <i>Message</i> je poruka primljena iz <i>Repa poslova</i> koja sadrži raspodijeljeni program napisan u jeziku <i>CL</i> . Ako <i>Interpretator programa</i> očekuje primitak poruke, operacija vraća pozitivnu potvrdu. U protivnome, operacija vraća negativnu potvrdu.
<code>bool DPEnd(DPEPR)</code>	Operacija kojom <i>Raspodijeljeni programi</i> postavljeni na računalu domaćinu dojavljuju <i>Interpretatoru programa</i> završetak izvođenja. Parametar <i>DPEPR</i> sadrži ime primjerka <i>Raspodijeljenog programa</i> koji je završio izvođenje. Ako <i>Interpretator programa</i> očekuje dojavu o završetku izvođenja od primjerka <i>Raspodijeljenog programa</i> , operacija vraća pozitivnu potvrdu. U protivnome, operacija vraća negativnu potvrdu.

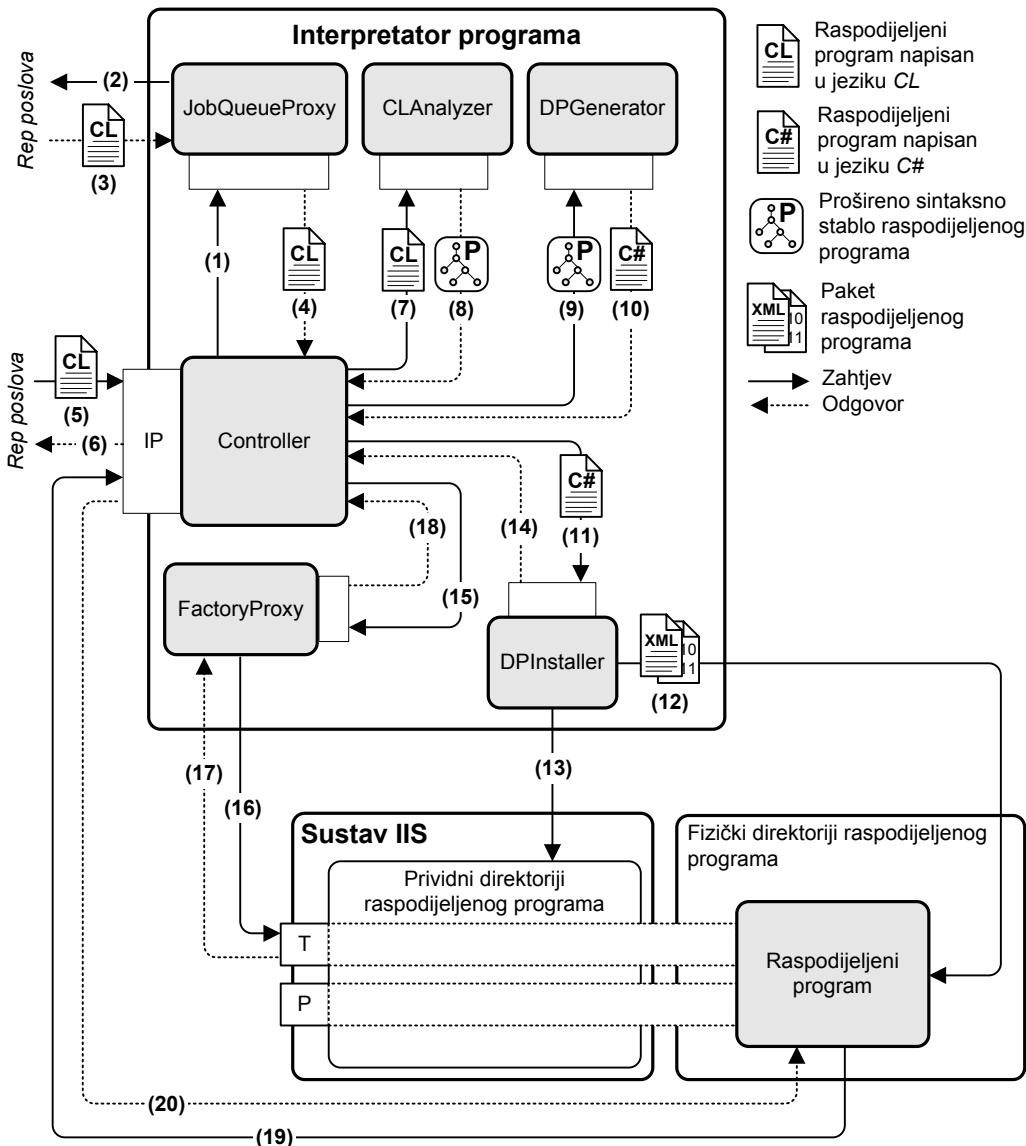
Tablica 4 sadrži opis operacija pristupnog sučelja *IP* ostvarenog podsustava *Interpretator programa*. Logika operacija koje ostvareni podsustav *Raspodijeljeni programi* sadrži u *sučelju tvornice T* ne ovisi o logici raspodijeljenog programa. Svi *Raspodijeljeni*

programi koji se izvode na računalu domaćinu sadrže isti skup operacija u *sučelju tvornice T* koje su opisane su u tablici 5. Logika operacija *programskog sučelja P_X* podsustava *Raspodijeljeni program* ovisi o logici opisanoj primjenom jezika *CL*. Opis *programskog sučelja P_X* ostvaren je u *WSDL* dokumentu koji je uključen u opisu raspodijeljenog programa napisanom u jeziku *CL*.

Tablica 5: Opis operacija *sučelja tvornice T* podsustava *Raspodijeljeni program*

Ime operacije	Opis operacije
<code>bool Create(Name, rEPR, Notify)</code>	Operacija kojom udaljeni korisnici stvaraju primjerke <i>Raspodijeljenih programa</i> postavljenih na računalu domaćinu. Parametar <i>Name</i> služi za definiranje jedinstvenog imena primjerka <i>Raspodijeljenog programa</i> . Parametar <i>rEPR</i> služi za definiranje adrese udaljene usluge koja zahtjeva stvaranje primjerka <i>Raspodijeljenog programa</i> . Parametar <i>Notify</i> je zastavica kojom pozivatelj izražava želju za primitkom dojave nakon što stvoreni primjerak <i>Raspodijeljenog programa</i> završi izvođenje. Dojavu o završetku izvođenja ostvaruje primjerak <i>Raspodijeljenog programa</i> pozivom operacije za dojavu na adresi <i>rEPR</i> . Ako je pokretanje uspješno ostvareno, operacija vraća pozitivnu potvrdu. U protivnome, operacija vraća negativnu potvrdu.
<code>bool Terminate(Name)</code>	Operacija kojom udaljni korisnici zaustavaljaju izvođenje primjerka <i>Raspodijeljenog programa</i> koji se trenutno izvodi. Parametar <i>Name</i> definira ime primjerka <i>Raspodijeljenog programa</i> čije je izvođenje potrebno zaustaviti. Ako je zaustavljanje uspješno završeno, operacija vraća pozitivnu potvrdu. U protivnome, operacija vraća negativnu potvrdu.

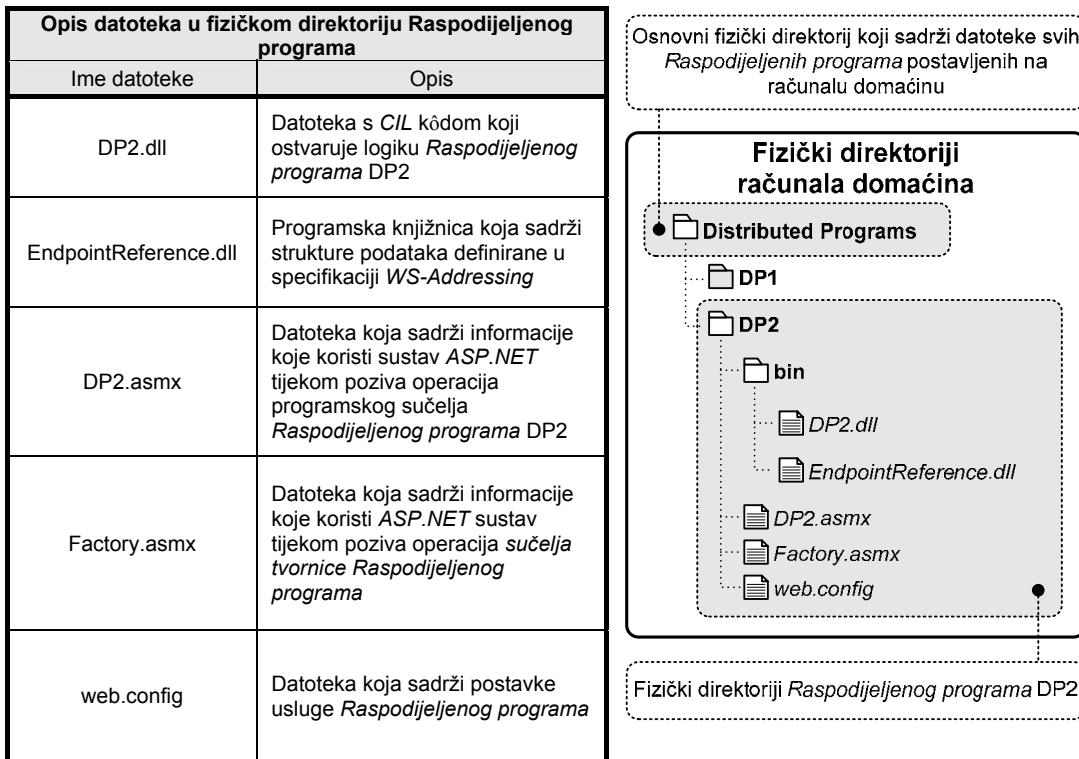
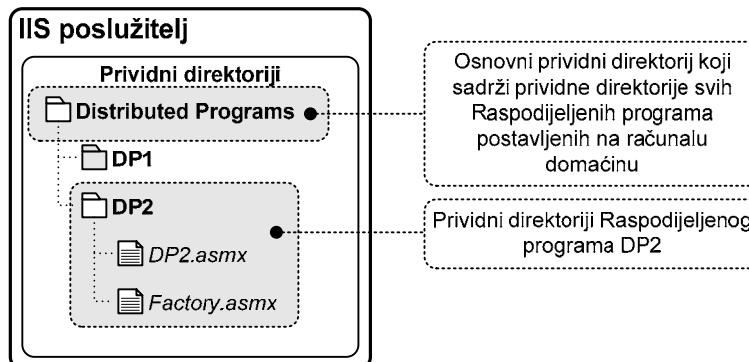
Logika ostvarenog podsustava *Interpretator programa* izgrađena je primjenom tri skupine razreda. Prvu skupinu razreda čine razredi koji su ostvareni izravno u programskom jeziku *C#*. Drugu skupinu razreda čine razredi koji su automatski izgrađeni primjenom dodatnih razvojnih alata. Posljednju skupinu razreda čine razredi dostupni u standardnoj knjižnici *.NET Framework* razvojne okoline.



Slika 81: Programsко ostvarenje podsustava *Interpretator programa*

Slika 81 prikazuje osnovne razrede koji ostvaruju logiku ostvarenog podsustava *Interpretator programa*. Logika podsustava *Interpretator programa* izgrađena je korištenjem razreda *Controller*, *JobQueueProxy*, *FactoryProxy*, *CLAnalyzer*, *DPGenerator* i *DPIInstaller*. Razred *Controller* ostvaruje operacije pristupnog sučelja *IP* podsustava *Interpretator programa* opisane u tablici 4, te ostvaruje upravljačku logiku podsustava *Interpretator programa*. Tijekom izvođenja *Interpretatora programa*, razred *Controller* dohvaća raspodijeljene programe spremljene u *Repu poslova* korištenjem razreda *JobQueueProxy* (1). Razred *JobQueueProxy* je zastupnik koji omogućava komunikaciju sa poštanskim pretinacom koji ostvaruje *Rep poslova*. Razred *JobQueueProxy* sadrži sve operacije pristupnog sučelja poštanskog pretinca. Po pozivu operacije za dohvaćanje poruke iz *Repa poslova*, razred *JobQueueProxy* proslijeđuje zahtjev *Repu poslova* (2) i prima odgovor (3).

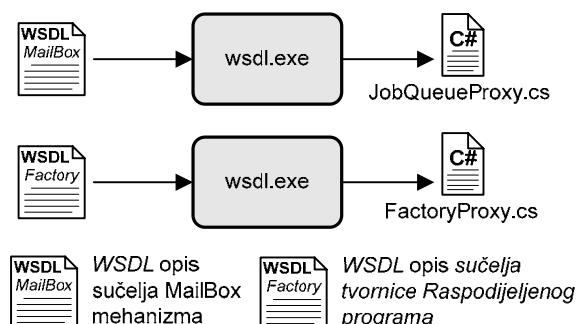
Ako se u trenutku poziva u *Repu poslova* nalaze raspodijeljni programi, razred *Controller* kao odgovor prima raspodijeljeni program koji je najduže spremljen u *Repu poslova* (4). Ako se u trenutku poziva u *Repu poslova* ne nalaze raspodijeljeni programi, razred *Controller* prima negativnu potvrdu i čeka primitak raspodijeljenog programa iz *Repa poslova* putem pristupnog sučelja *IP* (5, 6). Nakon primitka raspodijeljenog programa, razred *Controller* šalje primljeni program razredu *CLAnalyzer* (7). Razred *CLAnalyzer* provodi leksičku, sintaksnu i semantičku analizu programa, te gradi prošireno sintaksno stablo raspodijeljenog programa. Nakon primitka proširenog sintaksnog stabla (8), razred *Controller* šalje izgrađeno stablo razredu *DPGenerator* (9). Razred *DPGenerator* na temelju primljenog proširenog stabla graditi raspodijeljni program napisan u jeziku *C#*. Nakon primitka izgrađenog raspodijeljenog programa (10), razred *Controller* šalje program razredu *DPIInstaller* (11). Razred *DPIInstaller* prevodi primljeni raspodijeljeni program u jezik *CIL*, stvara datoteke potrebne za izvođenje usluge *Raspodijeljenog programa* i spremi ih u odgovarajući fizički direktorij na računalu domaćinu (12). Nakon izgradnje fizičkog direktorija, razred *CPInstaller* prijavljuje novi prividni direktorij u mrežnom poslužitelju sustava *Internet Information Services* (13) i šalje pozitivnu potvrdu razredu *Controller* (14). Prijavom novog prividnog direktorija omogućen je pristup sučeljima postavljenog *Raspodijeljenog programa*. Nakon postavljanja *Raspodijeljenog programa*, razred *Controller* koristi razred *FactoryProxy* kako bi započeo izvođenje novog primjerka *Raspodijeljenog programa* (15-18). Nakon završetka izvođenja, primjerak *Raspodijeljenog programa* šalje dojavu razredu *Controller* (19, 20) koji zatim uklanja postavljeni *Raspodijeljeni program* sa računala domaćina i ponavlja opisani postupak.

Slika 82: Prikaz strukture i opis datoteka u fizičkom direktoriju *Raspodijeljenog programa*Slika 83: Prikaz strukture i sadržaja prividnih direktorija *Raspodijeljenog programa*

Slika 82 prikazuje primjer strukture i sadržaja fizičkog direktorija *Raspodijeljenog programa* postavljenog na računalu domaćinu. Slika 83 prikazuje primjer strukture prividnog direktorija prijavljenog mrežnom poslužitelju sustava *Internet Information Services* nakon postavljanja *Raspodijeljenog programa* na računalu domaćinu. Mrežni poslužitelj sadrži osnovni prividni direktorij *DistributedPrograms* koji sadrži zasebni prividni direktorij za svaki *Raspodijeljeni program* koji je trenutno postavljen na računalu domaćinu. Prividni direktorij *Raspodijeljenog programa* dozvoljava javni pristup *.asmx* datotekama *programskega sučelja i sučelja tvornice*.

Razredi *JobQueueProxy* i *FactoryProxy*

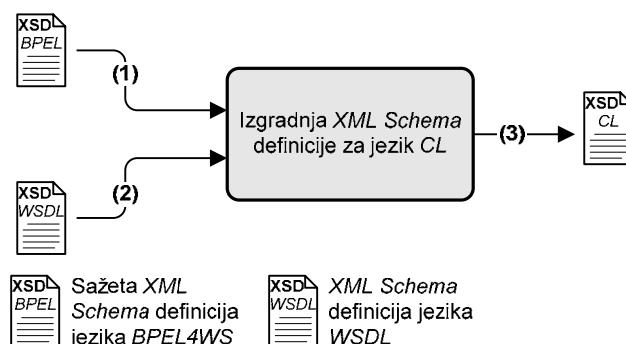
Razred *JobQueueProxy* ostvaruje zastupnika poštanskog pretinca koji ostvaruje *Rep poslova*. Razred *JobQueueProxy* sadrži operacije za stvaranje i uništavanje primjeraka poštanskog pretinca, te operacije za spremanje i dohvaćanje poruka iz poštanskog pretinca. Razred *FactoryProxy* ostvaruje zastupnika za *sučelja tvornice* svih podsustava *Raspodijeljeni programi* koji su postavljeni na računalu domaćinu. Razred *FactoryProxy* sadrži operacije *Create* i *Terminate* koje su opisane u tablici 5. Oba razreda *JobQueueProxy* i *FactoryProxy* izgrađena su automatski primjenom alata *wsdl.exe*. Slika 84 prikazuje postupak izgradnje razreda *JobQueueProxy* i *FactoryProxy*. Alat *wsdl.exe* kao ulaz prima *WSDL* opis sučelja usluge poštanskog pretinca ili *sučelja tvornice*, a za izlaz daje datoteku koja sadrži programsko ostvarenje zastupnika napisano primjenom jezika *C#*.



Slika 84: Postupak izgradnje razreda *JobQueueProxy* i *FactoryProxy*

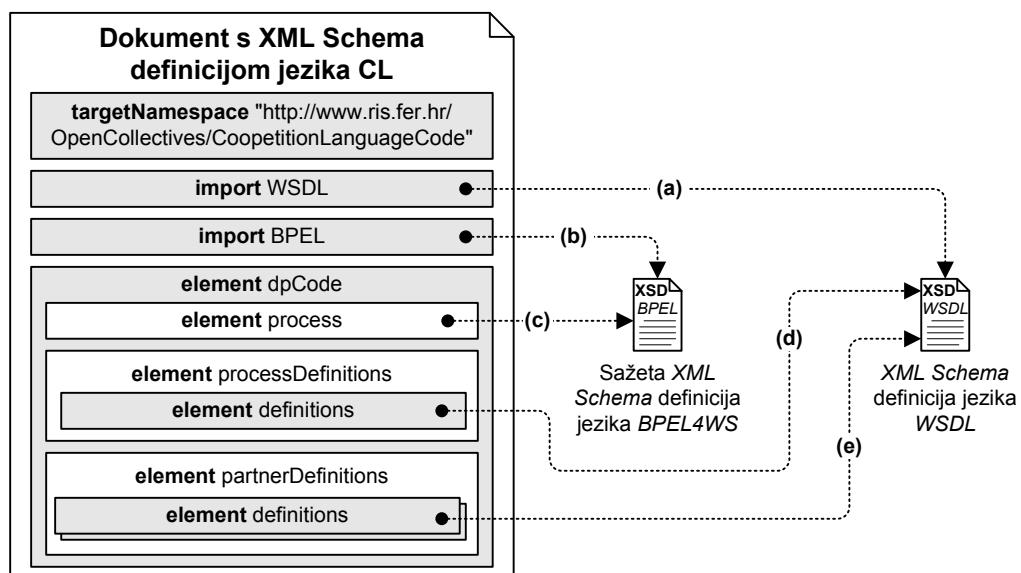
Razred *CLAnalyzer*

Razred *CLAnalyzer* ostvaruje leksičku, sintaksnu i semantičku analizu raspodijeljenih programa napisanih u jeziku *CL*, te gradi prošireno sintaksno stablo raspodijeljenog programa. Leksička i sintaksna analiza raspodijeljenih programa ostvarena je primjenom dokumenta s *XML Schema* definicijom jezika *CL*. Dokument s *XML Schema* definicijom jezika *CL* definira leksičke i sintaksne značajke *XML* dokumenata koji predstavljaju valjani opis raspodijeljenih programa napisanih u jeziku *CL*.



Slika 85: Postupak izgradnje dokumenta s *XML Schema* definicijom jezika *CL*

Slika 85 prikazuje postupak izgradnje dokumenta s *XML Schema* definicijom jezika *CL*. Postupak izgradnje *XML Schema* definicije jezika *CL* nije moguće automatizirati primjenom alata .NET Framework okruženja već razvijatelj gradi dokument samostalno na temelju leksičkih i sintaksnih pravila jezika *CL*. Obzirom da jezik *CL* koristi konstrukte jezika *BPEL4WS* i *WSDL*, *XML Schema* definiciju jezika *CL* potrebno je graditi primjenom dokumenata s *XML Schema* definicijama jezika *BPEL4WS* (1) i *WSDL* (2). Dokumenti s *XML Schema* definicijama jezika *WSDL* i *BPEL4WS* javno su dostupni u globalnoj mreži Internet. Obzirom da jezik *CL* koristi podskup naredbi jezika *BPEL4WS*, za izgradnju *XML Schema* definicije jezika *CL* ne koristi se cijelovita već sažeta *XML Schema* definicija jezika *BPEL4WS*. Sažeta *XML Schema* definicija jezika *BPEL4WS* sadrži definicije samo za naredbe jezika *BPEL4WS* koje koristi jezik *CL*. Dokument s *XML Schema* definicijom jezika *CL* gradi se uparivanjem sažete *XML Schema* definicije jezika *BPEL4WS* i *XML Schema* definicije jezika *WSDL* (3).

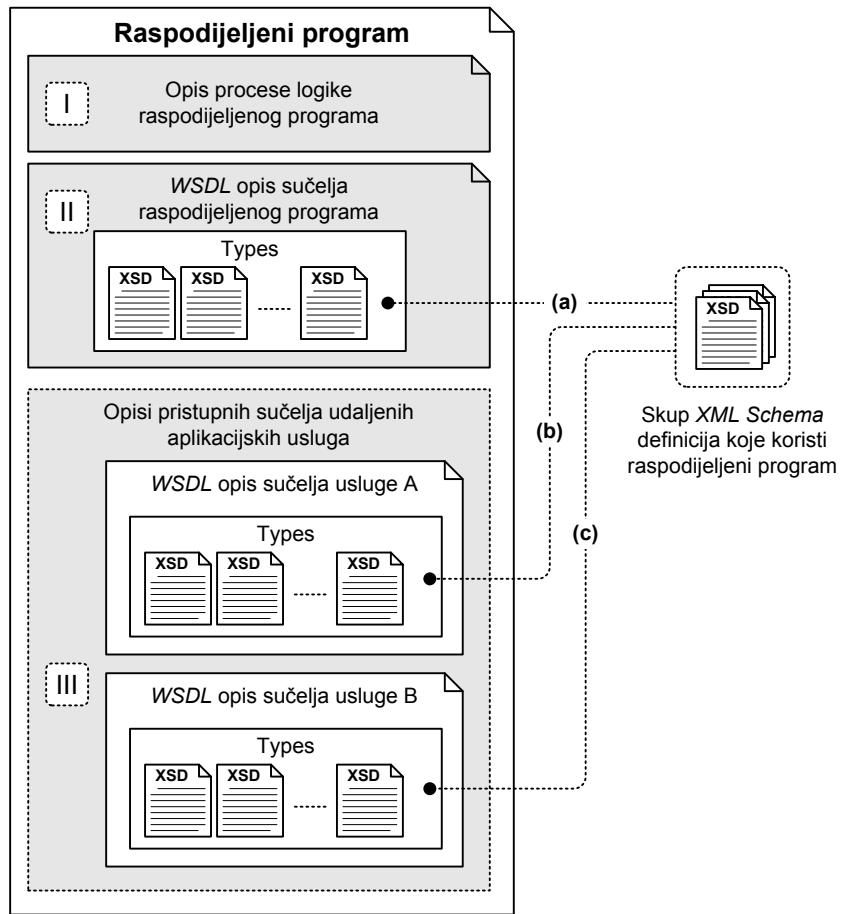


Slika 86: Globalna struktura dokumenta s *XML Schema* definicijom jezika *CL*

Slika 86 prikazuje globalnu strukturu dokumenta s *XML Schema* definicijom jezika *CL*. Na početku prikazanog dokumenta nalazi se naredba **targetNamespace** koja definira *XML* prostor imena u kojem se nalaze svi raspodijeljeni programi napisani u jeziku *CL*. Raspodijeljeni programi napisani u jeziku *CL* koriste prostor imena <http://www.ris.fer.hr/OpenCollectives/CoopetitionLanguageCode>. Nakon definicije *XML* prostora imena slijede dvije naredbe **import**. Naredbe **import** definiraju kazaljke na vanjske dokumente koji sadrže *XML Schema* definicije koje koristi prikazani dokument. Dokument s *XML Schema* definicijom jezika *CL* koristi dokument s *XML Schema* definicijom jezika *WSDL* (a) i

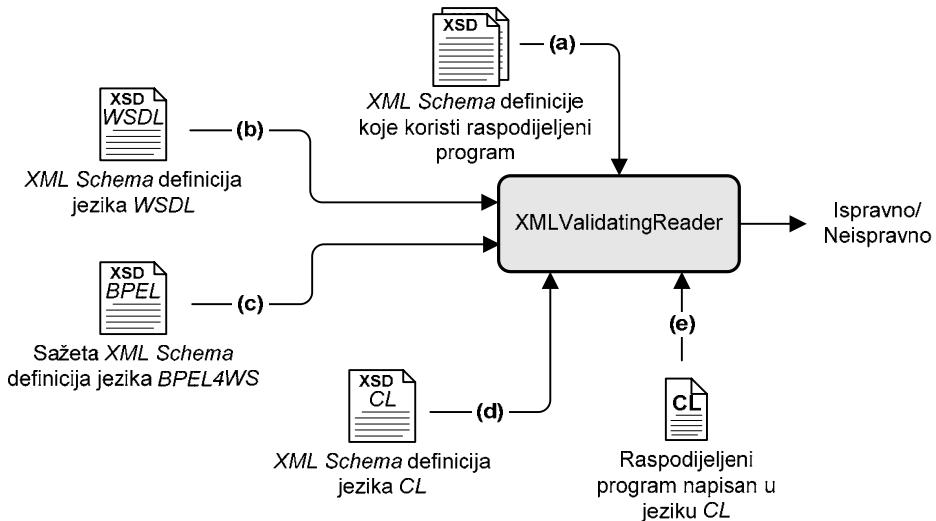
dokument s sažetom *XML Schema* definicijom jezika *BPEL4WS* (b). Nakon deklaracija kazaljki na vanjske dokumente, slijede deklaracije *XML* elemenata koji određuju strukturu raspodijeljenih programa napisanih u jeziku *CL*. Opis svakog raspodijeljenog programa napisan u jeziku *CL* započinje sa vršnim *XML* elementom **dpCode**. Element **dpCode** sadrži element **process** koji sadrži definiciju procesne logike raspodijeljenog programa ostvarenu primjenom podskupa naredbi jezika *BPEL4WS*. Element **process** definiran je u dokumentu s sažetom *XML Schema* definicijom jezika *BPEL4WS* (c). Nakon elementa **process** slijedi element **processDefinitions** koji sadrži *WSDL* opis pristupnog sučelja raspodijeljenog programa. Element **processDefinitions** u svojem tijelu sadrži element **definitions** koji je definiran u dokumentu s *XML Schema* definicijom jezika *WSDL* (d). Element **partnerDefinitions** je zadnji element u opisu raspodijeljenog programa i sadrži konačan broj *WSDL* opisa pristupnih sučelja usluga koje raspodijeljeni program poziva tijekom izvođenja. Element **partnerDefinitions** u svojem tijelu sadrži konačan broj elemenata **definitions** koji su definirani u dokumentu s *XML Schema* definicijom jezika *WSDL* (e). Potpuna struktura dokumenta s *XML Schema* definicijom jezika *CL* prikazana je dodatku A.

Izgrađeni dokument s *XML Schema* definicijom jezika *CL* korišten za ostvarivanje leksičke i sintaksne analize raspodijeljenih programa napisanih u jeziku *CL*. Tijekom leksičke i sintaksne analize koristi se također i dokument sa sažetom *XML Schema* definicijom jezika *BPEL4WS* i dokument s *XML Schema* definicijom jezika *WSDL*. Raspodijeljeni program u opisu procesne logike sadrži naredbe za obradu *SOAP* poruka koje tijekom izvođenja razmjenjuje s različitim udaljenim uslugama. Struktura tih *SOAP* poruka definirana je *WSDL* opisima udaljenih usluga koje raspodijeljeni program poziva tijekom izvođenja. Kako bi se uspješno provela leksička i sintaksne analiza raspodijeljenog programa, potrebno je stoga koristiti i sve *XML Schema* definicije koje se nalaze u *WSDL* opisima udaljenih usluga koje raspodijeljeni program poziva tijekom izvođenja.



Slika 87: Primjer postupka prikupljanja *XML Schema* definicija

Slika 87 prikazuje primjer provođenja postupka prikupljanja *XML Schema* definicija potrebnih za ostvarivanje leksičke i sintaksne analize raspodijeljenog programa. Na slici je prikazan primjer raspodijeljenog programa koji sadrži opis procesne logike (I), *WSDL* opis pristupnog sučelja raspodijeljenog programa (II) i *WSDL* opise pristupnih sučelja usluge A i usluge B (III) koje program poziva tijekom izvođenja. Kako bi se uspješno provela leksička i sintaksna analiza procesne logike prikazanog raspodijeljenog programa (I), potrebno je prikupiti *XML Schema* definicije koje se nalaze u *WSDL* opisu pristupnog sučelja raspodijeljenog programa (a). Također je potrebno prikupiti sve *XML Schema* definicije koje se nalaze u *WSDL* opisima usluge A (b) i usluge B (c). Izgrađeni skup *XML Schema* definicija koristi razred *CLAnalyzer* kako bi ostvario leksičku i sintaksnu analizu procesne logike raspodijeljenog programa napisanog u jeziku *CL*.

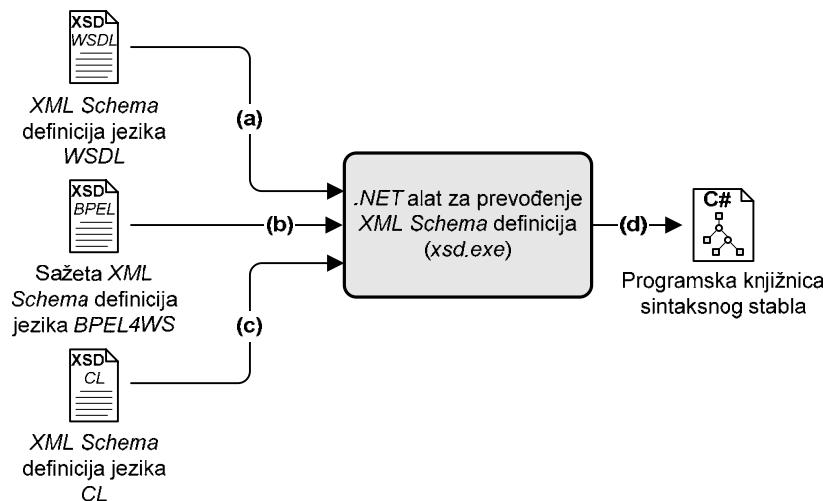


Slika 88: Leksička i sintaksna analiza raspodijeljenog programa napisanog u jeziku *CL*

Slika 88 prikazuje postupak leksičke i sintaksne analize raspodijeljenog programa napisanog u jeziku *CL*. Leksičku i sintaksnu analizu raspodijeljenih programa napisanih u jeziku *CL* provodi razred *CLAnalyzer* primjenom razreda *XMLValidatingReader* koji je dostupan je u prostoru imena *System.XML* standardne programske knjižnice .NET razvojne okoline. Razred *XMLValidatingReader* omogućava ispitivanje ispravnosti strukture i sadržaja određenog *XML* dokumenta primjenom zadanih *XML Schema* definicija. Tijekom leksičke i sintaksne analize raspodijeljenog programa, razred *XMLValidatingReader* za ulaz prima skup *XML Schema* definicija koje koristi raspodijeljeni program (a), *XML Schema* definiciju jezika *WSDL* (b), sažetu *XML Schema* definiciju jezika *BPEL4WS* (c), *XML Schema* definiciju jezika *CL* (d) i raspodijeljeni program napisan u jeziku *CL* čiju je ispravnost potrebno provjeriti (e). Kao rezultat postupka leksičke i sintaksne analize, razred *XMLValidatingReader* vraća pozitivnu potvrdu u slučaju da raspodijeljeni program ne sadrži leksičke i sintaksne pogreške. Ako raspodijeljeni program sadrži leksičke ili sintaksne pogreške, razred *XMLValidatingReader* vraća redak i stupac u kojem se nalazi prva leksička ili sintaksna pogreška i opis pogreške.

Nakon uspješno provedene leksičke i sintaksne analize, razred *CLAnalyzer* gradi sintaksno stablo raspodijeljenog programa. Osnovu za izgradnju sintaksnog stabla raspodijeljenog programa čini programska knjižnica sintaksnog stabla. Programska knjižnica sintaksnog stabla sadrži skup razreda koji služe za izgradnju sintaksnog stabla raspodijeljenih programa. Knjižnica je automatski izgrađena na temelju *XML Schema* definicije jezika *CL*, *XML Schema* definicije jezika *WSDL* i sažete *XML Schema* definicije jezika *BPEL4WS* primjenom alata *xsd.exe*. Alat *xsd.exe* dostupan je u .NET razvojnom

okruženju i omogućava automatsku pretvorbu zadanog dokumenta s *XML Schema* definicijom u razrede napisane u jeziku *C#*. Izgrađene razrede moguće je koristiti u svrhu automatske izgradnje sintaksnog stabla za *XML* dokumente koje opisuje zadana *XML Schema* definicija.



Slika 89: Postupak izgradnje programske knjižnice sintaksnog stabla

Slika 89 prikazuje postupak izgradnje programske knjižnice sintaksnog stabla. Kao ulaz za alat *xsd.exe* zadaju se *XML Schema* definicija jezika *WSDL* (a), sažeta *XML Schema* definicija jezika *BPEL4WS* (b) i *XML Schema* definicija jezika *CL* (c). Programski alat *xsd.exe* gradi programski knjižnicu sintaksnog stabla napisanu u jeziku *C#* (d). Razredi u izgrađenoj programskoj knjižnici koriste se za izgradnju sintaksnog stabla raspodijeljenog programa. Svaki razred predstavlja čvor u sintaksnom stablu raspodijeljenog programa i sadrži članove koji pokazuju na ostale razrede koji ostvaruju njegovu djecu u sintaksnom stablu. Alat *xsd.exe* koristi se na sljedeći način:

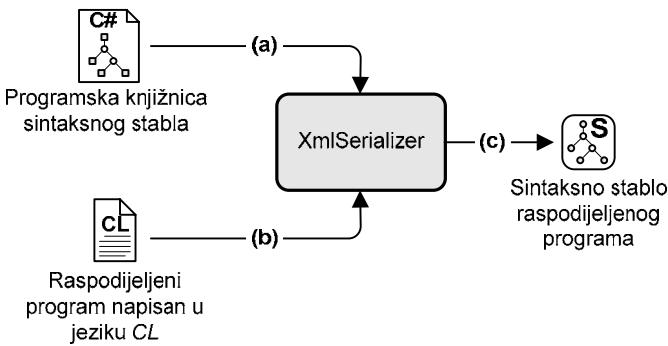
```
xsd.exe bpel.xsd wsdl.xsd cl.xsd /classes
```

Datoteka *bpel.xsd* sadrži sažetu *XML Schema* definiciju jezika *BPEL4WS*, datoteka *wsdl.xsd* sadrži *XML Schema* definiciju jezika *WSDL* i datoteka *cl.xsd* sadrži *XML Schema* definiciju jezika *CL*. Parametar */classes* definira *C#* razrede kao željeni rezultat izvođenja alata *xsd.exe*. Nakon izvođenja alat *xsd.exe* kao rezultat daje datoteku *bpel_wsdl_cl.cs* koja sadrži razrede programske knjižnice sintaksnog stabla ostvarene u jeziku *C#*.

```
public class tInvoke : tActivity {  
  
    /// <remarks/>  
    [System.Xml.Serialization.XmlAttributeAttribute(  
        DataType="NCName")]  
    public string partnerLink;  
  
    /// <remarks/>  
    [System.Xml.Serialization.XmlAttributeAttribute()  
    ]  
    public System.Xml.XmlQualifiedName portType;  
  
    /// <remarks/>  
    [System.Xml.Serialization.XmlAttributeAttribute(  
        DataType="NCName")]  
    public string operation;  
  
    /// <remarks/>  
    [System.Xml.Serialization.XmlAttributeAttribute(  
        DataType="NCName")]  
    public string inputVariable;  
  
    /// <remarks/>  
    [System.Xml.Serialization.XmlAttributeAttribute(  
        DataType="NCName")]  
    public string outputVariable;  
}
```

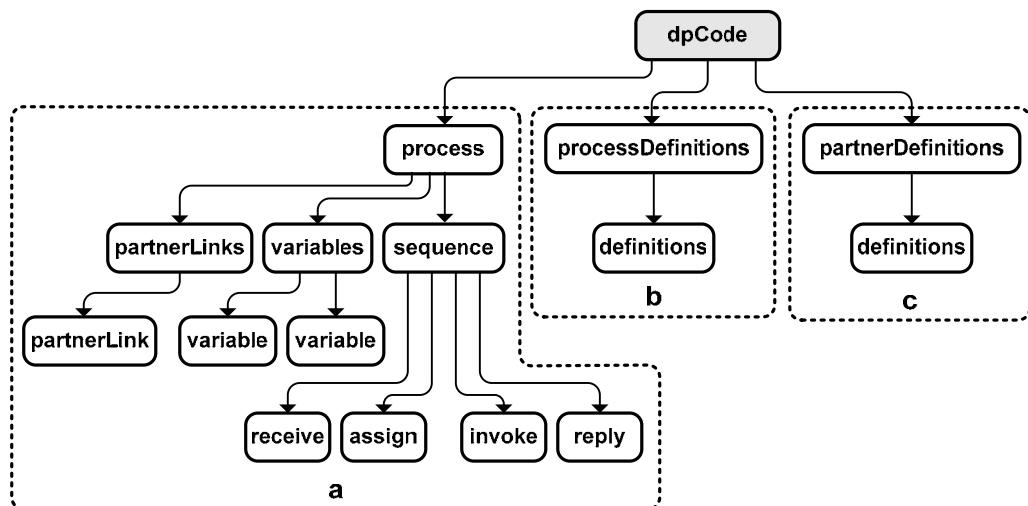
Slika 90: Isječak programske knjižnice sintaksnog stabla

Slika 90 prikazuje isječak programske knjižnice sintaksnog stabla koja je automatski izgrađena primjenom alata *xsd.exe*. Prikazani isječak sadrži deklaraciju razreda **tInvoke** koji opisuje čvor sintaksnog stabla raspodijeljenog programa za *CL* naredbu *invoke*. Razred **tInvoke** sadrži podatkovne članove **partnerLink**, **portType**, **operation**, **inputVariable** i **outputVariable** koji sadrže konkretnе vrijednosti parametara *CL* naredbe *invoke*. Svakom podatkovnom članu pridružen je atribut jezika *C#* naveden u uglatim zagradama povrh deklaracije podatkovnog člana. Atributi jezika *C#* definiraju pravila preslikavanja sadržaja podatkovnog člana koji je spremljen u memoriji u *XML* zapis, te pravila za preslikavanje *XML* zapisa u sadržaj podatkovnog člana u memoriji. Razred *CLAnalyzer* koristi izgrađenu programsku knjižnicu za izgradnju sintaksnog stabla raspodijeljenog programa.



Slika 91: Postupak izgradnje sintaksnog stabla raspodijeljenog programa

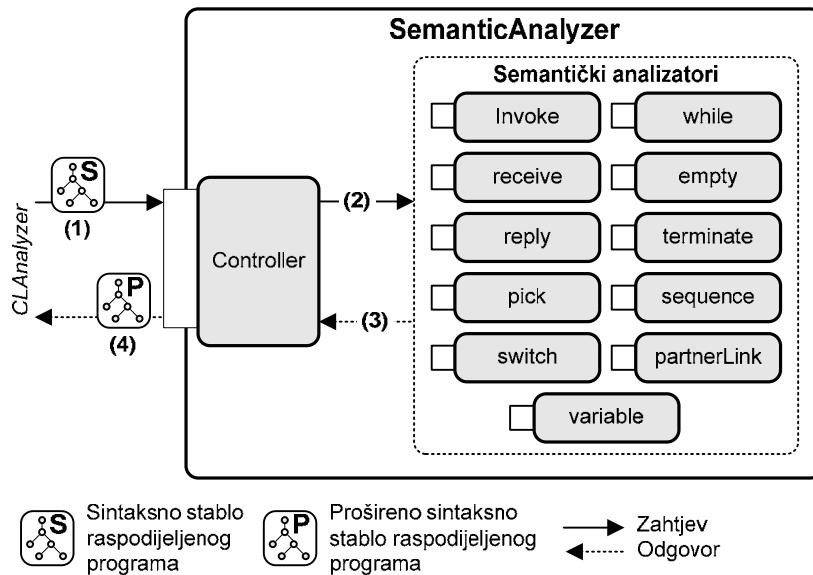
Slika 91 prikazuje postupak izgradnje sintaksnog stabla raspodijeljenog programa koji provodi razred *CLAnalyzer*. Izgradnja sintaksnog stabla provodi se primjenom razreda *XmlSerializer* koji je dostupan u prostoru imena *System.Xml* standardne knjižnice .NET razvojne okoline. Razred *XmlSerializer* kao ulaz prima definicije razreda u programskoj knjižnici sintaksnog stabla (a) i raspodijeljeni program napisan u jeziku CL (b). Kao rezultat izvođenja, razred *XmlSerializer* daje sintaksno stablo za primljeni raspodijeljeni program koje je izgrađeno primjenom razreda programske knjižnice sintaksnog stabla (c).



Slika 92: Primjer sintaksnog stabla raspodijeljenog programa

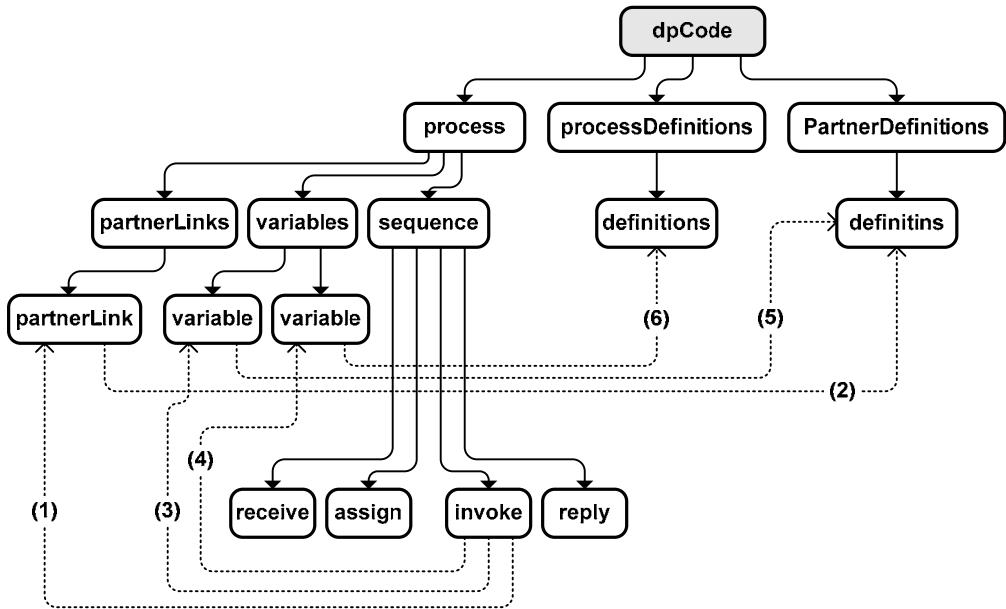
Slika 92 prikazuje primjer sintaksnog stabla raspodijeljenog programa koji u procesnoj logici sadrži CL naredbe *receive*, *assign*, *invoke* i *reply*, te tijekom izvođenja poziva samo jednu udaljenu uslugu. Sintaksno stablo raspodijeljenog programa sadrži korijenski čvor *dpCode* s tri čvora djeteta *process* (a), *processDefinitions* (b) i *partnerDefinitions* (c). Čvor *process* sadrži podstablo s čvorovima koji predstavljaju naredbe procesne logike raspodijeljenog programa. Čvor *processDefinitions* sadrži podstablo s *WSDL* opisom pristupnog sučelja raspodijeljenog programa. Čvor *partnerDefinitions* sadrži podstablo s *WSDL* opisom pristupnog sučelja usluge koju raspodijeljeni program koristi

tijekom izvođenja. Izgrađeno sintaksno stablo primjenjuje se za provođenje semantičke analize raspodijeljenog programa. Semantičku analizu raspodijeljenog programa ostvaruje razred *SemanticAnalyzer* koji koristi razred *CLAnalyzer*.



Slika 93: Programsко ostvarenje razreda *SemanticAnalyzer*

Slika 93 prikazuje elemente programskog ostvarenja razreda *SemanticAnalyzer* koji sadrži razrede *Controller* i skup razreda koji ostvaruju semantičku analizu za po jednu naredbu jezika *CL*. Razred *Controller* prima sintaksno stablo raspodijeljenog programa od razreda *CLAnalyzer* (1). Razred *Controller* provodi postupak semantičke analize primjenom metode rekurzivnog spusta [7]. Tijekom rekurzivnog spusta, ostvaruje se poziv odgovarajućeg razreda semantičkog analizatora (2) za svaku naredbu koja se pojavljuje u procesnoj logici raspodijeljenog programa. Svaki semantički analizator ispituje semantičkih pravila za pojedinu naredbu jezika *CL*. U slučaju da ne postoji semantička pogreška, semantički analizator vraća pozitivnu potvrdu (3). Razred *Controller* prihvata pozitivnu potvrdu i postupno gradi prošireno sintaksno stablo. Nakon završetka izgradnje proširenog sintaksnog stabla, razred *Controller* proslijediće izgrađeno prošireno sintaksno stablo razredu *CLAnalyzer* (4).

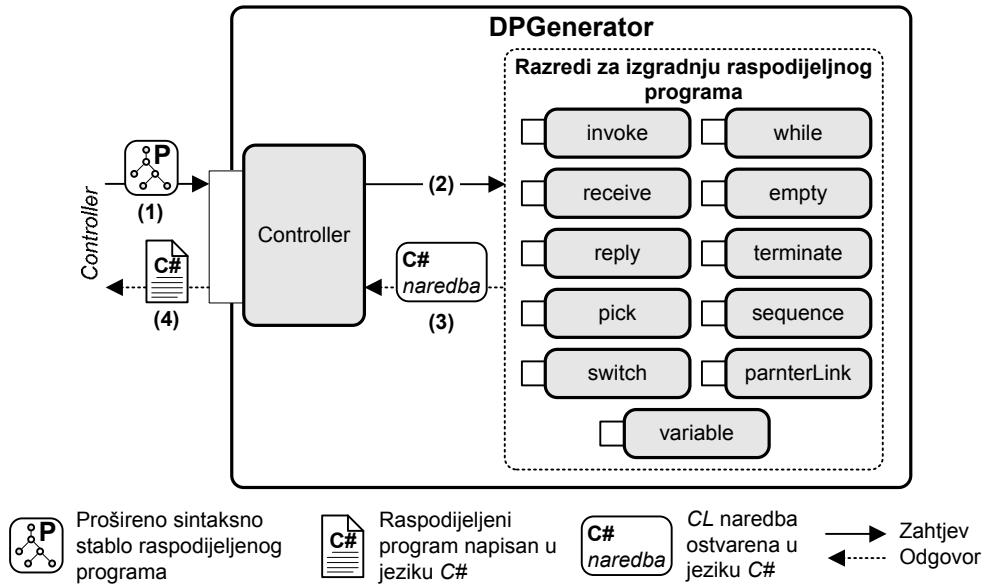


Slika 94: Primjer proširenog sintaksnog stabla raspodijeljenog programa

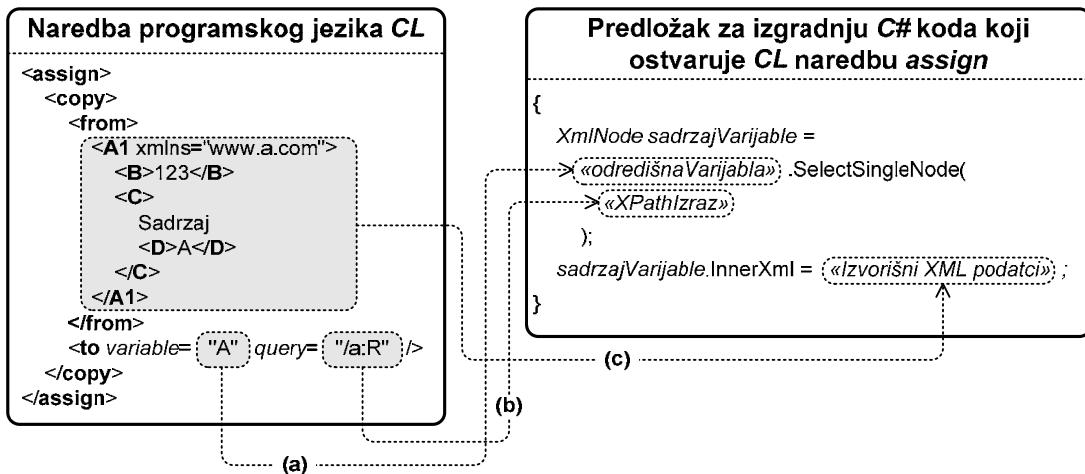
Slika 94 prikazuje primjer djelomično proširenog sintaksnog stabla raspodijeljenog programa nastalog proširivanjem sintaksnog stabla prikazanog na slici 92. Čvor naredbe *invoke* proširen je dodatnim kazaljkama na čvorove sintaksnog stabla. Dodatne kazaljke omogućavaju lakše i učinkovitije pretraživanje sintaksnog stabla tijekom generiranja C# kôda. Čvor naredbe *invoke* sadrži kazaljku na čvor koji sadrži deklaraciju simboličkog imena usluge koju raspodijeljeni program poziva (1). Čvor deklaracije simboličkog imena usluge sadrži kazaljku na *WSDL* dokument koji opisuje pristupno sučelje usluge za koju je deklarirano simboličko ime (2). Čvor naredbe *invoke* također sadrži kazaljke na deklaracije ulaznih i izlaznih varijabli koje koristi naredba *invoke* (3, 4). Nadalje, čvorovi za svaku deklariranu varijablu sadrže kazaljke na *WSDL* dokumente koji sadrže definicije strukture i sadržaja *SOAP* poruka koje spremaju deklarirane varijable (5, 6). Prošireno sintaksno stablo raspodijeljenog programa primjenjuje se tijekom generiranja C# kôda koji ostvaruje razred *DPGenerator*.

Razred DPGenerator

Razred *DPGenerator* na temelju proširenog sintaksnog stabla raspodijeljenog programa gradi raspodijeljeni program napisan u jeziku C#. Postupak izgradnje raspodijeljenog programa ostvaren je primjenom rekurzivnog spustava duž proširenog sintaksnog stabla raspodijeljenog programa.

**Slika 95:** Programsко ostvarenje razreda *DPGenerator*

Slika 95 prikazuje programsko ostvarenje razreda *DPGenerator*. Razred *DPGenerator* sadrži razred *Controller* i po jedan razred za svaku naredbu jezika *CL* koji ostvaruje prevođenje *CL* naredbe u jezik *C#*. Razred *Controller* proslijeđuje prošireno sintaksno stablo razredu *Controller* (1). Nakon primitka proširenog sintaksnog stabla, započinje izgradnja raspodijeljenog programa napisanog u jeziku *C#* primjenom postupka rekurzivnog spusta. Tijekom rekurzivnog spusta ostvaruje se poziv odgovarajućeg razreda koji gradi *C#* kôd za svaku naredbu koja se pojavljuje u procesnoj logici raspodijeljenog programa. Svaki od razreda za izgradnju raspodijeljenog programa ostvaruje logiku za prevodenje jedne naredbe jezika *CL* u skup *C#* naredbi primjenom predložaka za *CL* naredbe. Predložak *CL* naredbe sadrži ostvarenje logike *CL* naredbe u jeziku *C#* s praznim poljima u koja se postavljaju konkretnе vrijednosti parametara zadane *CL* naredbe. Razred *Controller* prima ostvarenje svake *CL* naredbe u *C#* jeziku (3) i postupno gradi cjeloviti raspodijeljeni program napisan u jeziku *C#*. Nakon završetka izgradnje, razred *Controller* proslijeđuje raspodijeljeni program napisan u jeziku *C#* razredu *Controller* koji je dio ostvarenog podsustava *Interpretator programa* (5).

**Slika 96:** Primjer korištenja predloška za izgradnju C# kôda za CL naredbu *assign*

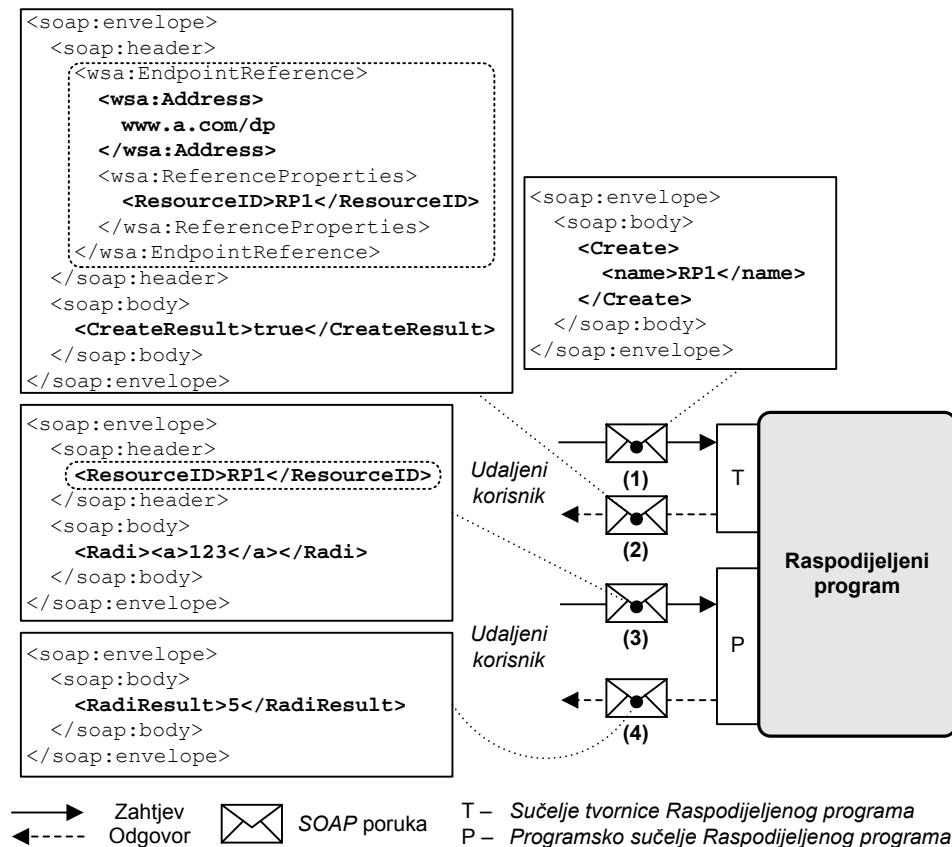
Slika 96 prikazuje primjer korištenja predloška za *CL* naredbu *assign* napisanog u jeziku *C#*. Predložak za *CL* naredbu *assign* sadrži tri nadomjesna polja u koja se tijekom prevodenja postavljaju konkretnе vrijednosti parametra *CL* naredbe *assign*. Prvi parametar predloška je ime odredišne varijable u koju se spremi izvořišni *XML* dokument zadan u *CL* naredbi *assign* (a). Drugi parametar je upit napisan u jeziku *XPath* koji određuje *XML* element u varijabli u koje će biti spremljen izvořišni *XML* dokument. Obzirom da .NET razvojna okolina sadrži programsku knjižnicu koja ostvaruje izravnu potporu za jezik *XPath*, upit se izravno preslikava na odgovarajuće mjesto u predlošku (b). Posljednji parametar je izvořišni *XML* dokument koji se izravno preslikava u polje koje služi za definiranje sadržaja varijable raspodijeljenog programa (c). Konačni rezultat provodenja primjenom predloška prikazan je na slici 97.

```
{
  XmlNode sadrzajVarijable =
    A.SelectSingleNode(
      "/a:R"
    );
  sadrzajVarijable.InnerXml = @@
  <A1 xmlns="www.a.com">
    <B>123</B>
    <C>
      Sadrzaj
      <D>A</D>
    </C>
  </A1>;
}
```

Slika 97: Primjer izgrađenog C# kôda za CL naredbu *assign*

6.1.2 Raspodijeljeni program

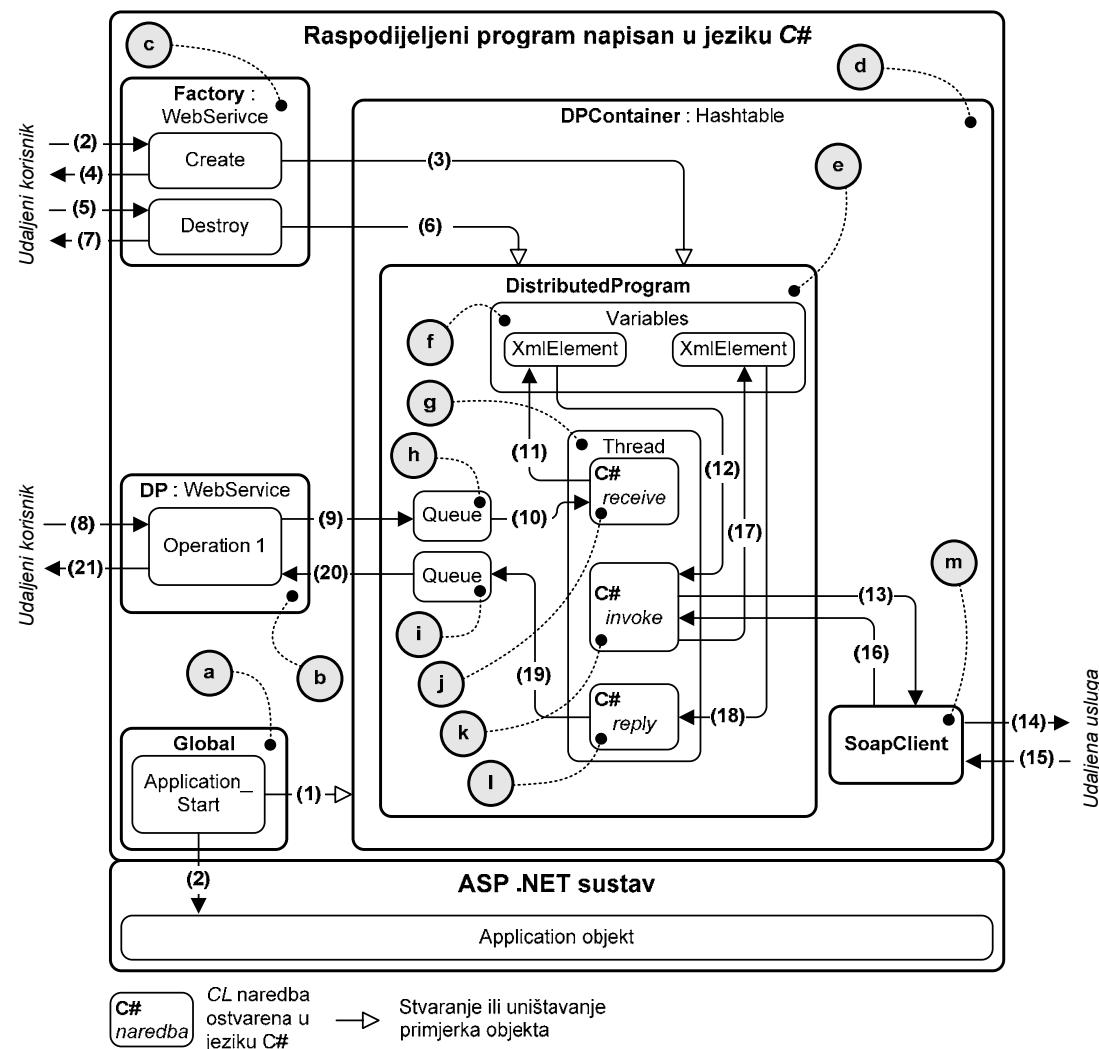
Podsustav *Raspodijeljeni program* ostvaren je u jeziku C# kao usluga s čuvanjem stanja prema specifikaciji *WS-ResourceFramework*. *Raspodijeljeni program* istovremeno može izvoditi konačan broj nezavisnih primjeraka raspodijeljenih programa od kojih je svaki ostvaren kao zasebni *WS-Resource*.



Slika 98: Primjer korištenja ostvarenog podsustava *Raspodijeljeni program*

Slika 98 opisuje primjer postupaka stvaranja i korištenja primjerka *Raspodijeljenog programa* putem *sučelja tvornice* (*T*) i *programskog sučelja* (*P*). Kako bi stvorio novi primjerak *Raspodijeljenog programa*, udaljeni korisnik poziva operaciju *Create*, dostupnu u *sučelju tvornice*, slanjem odgovarajuće *SOAP* poruke zahtjeva *Raspodijeljenom programu* (1). *SOAP* poruka zahtjeva sadrži željeno ime novog primjerka *Raspodijeljenog programa* (*RP1*). Nakon primitka *SOAP* poruke zahtjeva, *Raspodijeljeni program* stvara novi primjerak programa sa zadanim imenom. Nakon stvaranja novog primjerka programa, *Raspodijeljeni program* udaljenom korisniku šalje *SOAP* poruku odgovora koja u zaglavlju sadrži element *EndpointReference* definiran specifikacijom *WS-Addressing* (2). Element *EndpointReference* sadrži adresu računala domaćina (*www.a.com/dp*) na kojem se izvodi stvoreni primjerak programa i ime stvorenog primjerka *Raspodijeljenog programa* (*RP1*). Informacije dostupne

u elementu *EndpointReference* udaljeni korisnici koriste za naslovljavanje stvorenog primjerka *Raspodijeljenog programa* putem *programskog sučelja*. Kako bi ostvario poziv operacije za stvoreni primjerak *Raspodijeljenog programa*, udaljni korisnik šalje *SOAP* poruku zahtjeva koja u zaglavlju sadrži element *ResourceID* (3). Element *ResourceID* sadrži ime primjerka *Raspodijeljenog programa* kojem je namijenjena *SOAP* poruka zahtjeva. *SOAP* poruka zahtjeva u svojem tijelu također sadrži aplikacijske podatke koje će obraditi naslovljeni primjerka *Raspodijeljenog programa*. Prikazana *SOAP* poruka zahtjeva u svojem tijelu sadrži *XML* podatke: <Radi><a>123</Radi>. *Raspodijeljeni program* prosljeđuje primljenu *SOAP* poruku zahtjeva naslovljenom primjerku programa. Naslovljeni primjerak programa potom obrađuje primljenu *SOAP* poruku zahtjeva i udaljenom korisniku prosljeđuje *SOAP* poruku odgovora koja sadrži rezultate obrade (4). Prikazana *SOAP* poruka odgovora sadrži *XML* podatke: <RadiResult>5</RadiResult>.



Slika 99: Programsко ostvarenje logike podsustava *Raspodijeljeni program*

Slika 99 prikazuje programsko ostvarenje logike podsustava *Raspodijeljeni program* napisane u jeziku C#. Logika podsustava *Raspodijeljeni program* ostvarena je primjenom razreda *Global* (a), *DP* (b), *Factory* (c) i *DPContainer* (d). Razred *Global* sadrži skup standardnih metoda koje poziva *ASP.NET* sustav tijekom postavljanja, pokretanja i uklanjanja usluge *Raspodijeljenog programa* na računalu domaćinu. Razred *DP* izgrađen je nasljeđivanjem .NET razreda *WebService* i ostvaruje *programsko sučelje* podsustava *Raspodijeljeni program*. Razred *Factory* također je izgrađen nasljeđivanjem .NET razreda *WebService* i ostvaruje *sučelje tvornice* podsustava *Raspodijeljeni program*. Razred *DPContainer* izgrađen je primjenom .NET razreda *Hashtable* i ostvaruje funkcionalnosti potrebne za spremanje i izvođenje konačnog broja primjeraka *Raspodijeljenih programa* na računalu domaćinu.

Nakon postavljanja *Raspodijeljenog programa* na računalo domaćin, *ASP.NET* sustav poziva metodu *Application_Start* razreda *Global* (a) koja izvodi početne akcije potrebne za početak uspješnog izvođenja *Raspodijeljenog programa*. Metoda *Application_Start* stvara primjerak objekta razreda *DPContainer* (1) i spremi ga u *Application* objekt *ASP.NET* sustava (2). *Application* objekt je postojani objekt koji služi za čuvanje korisničkih podataka tijekom izvođenja usluge. Podaci spremljeni u *Application* objektu ostaju sačuvani i nakon proizvoljnog broja poziva koje prima usluga tijekom rada. Svi aktivni primjeri *Raspodijeljenih programa* koji se izvode na računalu domaćinu spremljeni su u *Application* objektu. Nakon završetka opisanih pripremnih akcija, *Raspodijeljeni program* je postavljen na računalu domaćinu i dostupan za korištenje.

Udaljeni korisnici stvaraju i uništavaju primjerke *Raspodijeljenog programa* pozivanjem operacija *Create* i *Destroy* koje sadrži razred *Factory* (c). Obje operacije izložene su kao javno dostupne operacije ostvarene primjenom C# atributa *WebMethod*. Tijekom poziva operacije *Create* (2), novi primjerak *Raspodijeljenog programa* sa zadanim imenom spremi se u razred *DPContainer* (3). Operacija *Create* zatim šalje udaljenom korisniku pozitivnu potvrdu (4). Uništavanje postojećeg primjerka *Raspodijeljenog programa* ostvaruje se na sličan način korištenjem operacije *Destroy* (5-7).

Primjeri *Raspodijeljenog programa* ostvareni su uporabom zasebnog razreda *DistributedProgram* (e). Razred *DistributedProgram* koristi razrede *Variables* (f), *Thread* (g) i razred *Queue* (h, i). Razred *Variables* sadrži po jedan objekt razreda *XmlElement* za svaku varijablu koju koristi *Raspodijeljeni program*. Razred *Thread* ostvaruje dretvu koja izvodi naredbe *Raspodijeljenog programa* ostvarene u jeziku C# (j, k, l). Razred *Queue*

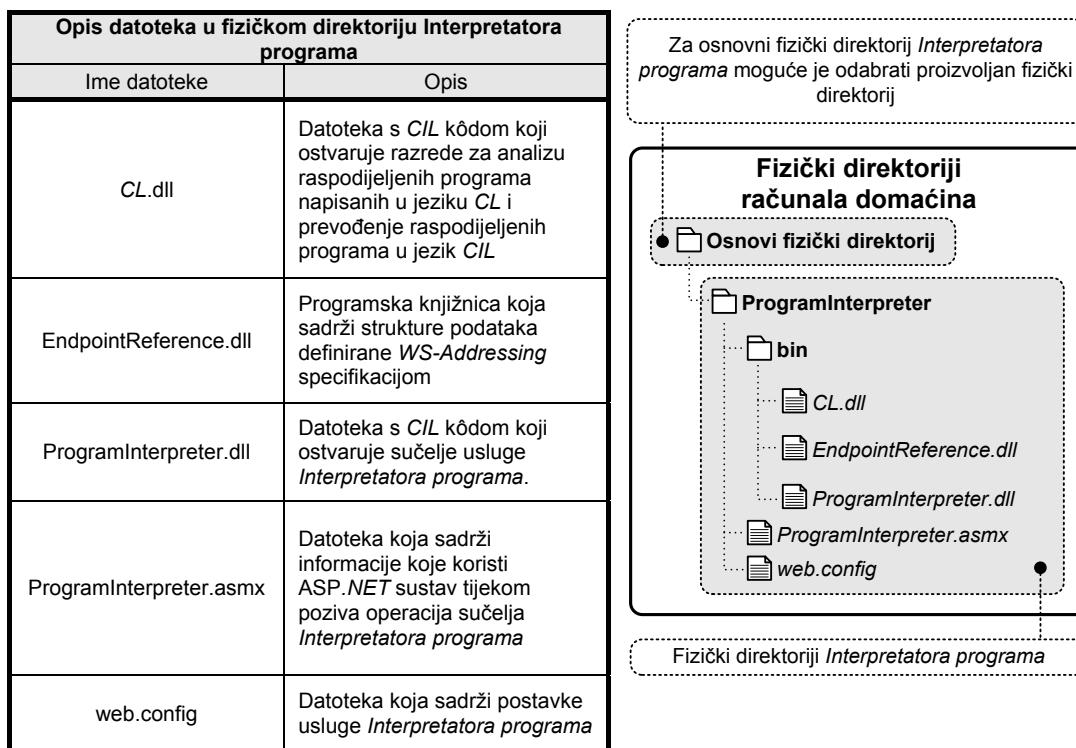
koristi se za ostvarivanje repa za *SOAP* poruke koje primjerak *Raspodijeljenog programa* razmjenjuje s udaljenim korisnicima putem *programskog sučelja*. Svakoj operaciji *Raspodijeljenog programa* pridružen je jedan rep za *SOAP* poruke zahtjeva (h) i jedan rep za *SOAP* poruke odgovora (i).

Nakon stvaranja primjera *Raspodijeljenog programa*, razred *Thread* započinje slijedno izvođenje naredbi *Raspodijeljenog programa*. U primjeru prikazanom na slici 99 *Raspodijeljeni program* sadrži naredbe *receive* (j), *invoke* (k) i *reply* (l). Prvo se izvodi naredba *receive* kojom primjerak *Raspodijeljenog programa* ostvaruje čekanje poziva operacije *Operation1*. Udaljeni korisnik ostvara poziv operacije *Operation1* usmjeravanjem *SOAP* poruke zahtjeva (8) koja u zaglavlju sadrži ime odredišnog primjera *Raspodijeljenog programa*. Razred *DP* prihvata *SOAP* poruku zahtjeva, dohvaća ime naslovjenog primjera programa i prosljeđuje primljenu poruku u rep za *SOAP* poruke zahtjeva (9). Naredba *receive* dohvaća *SOAP* poruku zahtjeva iz repa (10) i sprema primljenu poruku u varijablu za spremanje poruke zahtjeva (11). Nakon završetka izvođenja naredbe *receive*, izvodi se naredba *invoke*. Naredba *invoke* dohvaća vrijednost variabile koja sadrži spremljenu *SOAP* poruku zahtjeva (12) i prosljeđuje primljenu poruku razredu *SoapClient* (13). Razred *SoapClient* (m) dostupan je u *Web Services Enhancements* programskoj knjižnici i koristi se za ostvarivanje zastupnika za poziv udaljene usluge koje *Raspodijeljeni program* koristi tijekom izvođenja. Razred *SoapClient* prosljeđuje primljenu *SOAP* poruku zahtjeva udaljenoj usluzi (14), prihvata *SOAP* poruku odgovora (15) i prosljeđuje primljenu *SOAP* poruku odgovora naredbi *invoke* (16). Nakon primitka *SOAP* poruke odgovora, naredba *invoke* sprema primljenu poruku u varijablu za spremanje poruke odgovora (17) i završava s izvođenjem. Nakon završetka izvođenja naredbe *invoke*, izvodi se naredba *reply* koja dohvaća spremljenu *SOAP* poruku odgovora (18) i prosljeđuje *SOAP* poruku odgovora u odgovarajući rep poruka (19). Operacija razreda *DP* dohvaća *SOAP* poruku odgovora spremljenu u repu poruka (20) i prosljeđuje primljenu *SOAP* poruku odgovora udaljenom korisniku (21). Nakon završetka izvođenja naredbe *reply*, završava izvođenje primjera programa. Po završetku izvođenja, razred *DPContainer* uklanja objekt razreda *DistributedProgram* koji odgovara primjerku *Raspodijeljenog programa*.

6.2 Postavljanje i korištenje raspodijeljenog interpretatora programa

Za postavljanje i korištenje raspodijeljenog interpretatora programa na radnim računalima potrebno je koristiti *Windows 2000*, *Windows XP* ili *Windows 2003* operacijski

sustav, *Internet Information Services* sustav inačice 5.0, *.NET Framework* sustav inačice 1.1 i programsku knjižnicu *Web Services Enhancements* inačice 2.0. Postavljanje ostvarenog *Interpretatora programa* ostvaruje se izgradnjom odgovarajuće strukture fizičkih direktorija na računalu domaćinu.



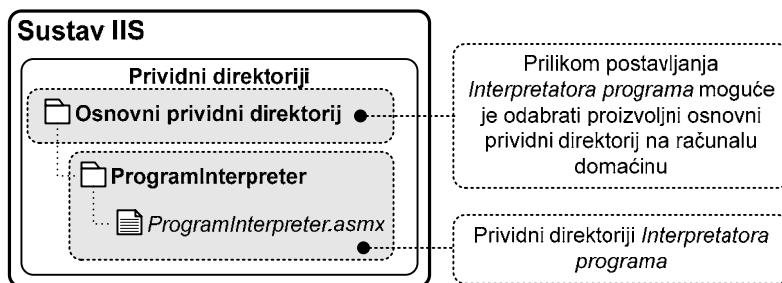
Slika 100: Prikaz strukture i opis datoteka u fizičkom direktoriju *Interpretatora programa*

Slika 100 prikazuju strukturu i sadržaj fizičkog direktorija *Interpretatora programa* na računalu domaćinu. Nakon postavljanja *Interpretatora programa* u odabrani osnovni fizički direktorij na računalu domaćinu, potrebno je definirati odgovarajuće postavke u *web.config* datoteci *Interpretatora programa*.

```
<configuration>
  <appSettings>
    <add key="DPInstallationBaseDirectory"
         value="C:/ProgramInterpreter/DistributedPrograms" />
    <add key="DPBaseVirtualDirectory"
         value="ProgramInterpreter/DistributedPrograms" />
    <add key="EndpointReferenceLib"
         value="C:/ProgramInterpreter/bin/Debug/EndpointReference.dll" />
    <add key="WSE20Lib"
         value="C:/ProgramInterpreter/bin/Debug/Microsoft.Web.Services2.dll" />
  </appSettings>
</configuration>
```

Slika 101: Sadržaj datoteke s postavkama ostvarenog *Interpretatora programa*

Slika 101 prikazuje primjer sadržaja datoteke *web.config* koja sadrži postavke ostvarenog *Interpretatora programa* zapisane primjenom jezika *XML*. Datoteka *web.config* sadrži osnovni element **configuration** koji sadrži element **appSettings**. Element **appSettings** sadrži skup elementa **add** koji određuju postavke ostvarenog *Interpretatora programa*. Svaki **add** element sadrži atribute **key** i **value**. Atribut **key** određuje ime postavke, dok atribut **value** određuje njezinu vrijednost. Postavka *DPIInstallationBaseDirectory* određuje osnovni fizički direktorij koji *Interpretator programa* koristi za spremanje datoteka raspodijeljenih programa koji se izvode na računalu domaćinu. U prikazanom primjeru zadan je osnovni fizički direktorij *C:/ProgramInterpreter/DistributedPrograms*. Postavka *DPBaseVirtualDirectory* određuje osnovni prividni direktorij koji *Interpretator programa* pridružuje prividnim direktorijima raspodijeljenih programa postavljenim na računalu domaćinu. U prikazanom primjeru zadan je osnovni prividni direktorij *ProgramInterpreter/DistributedPrograms*. Postavka *EndpointReferenceLib* određuje put do datoteke koja ostvaruje knjižnicu sa strukturama podataka definiranim u specifikaciji *WS-Addressing*. U prikazanom primjeru zadan je put *C:/ProgramInterpreter/bin/Debug/EndpointReference.dll*. Postavka *WSE20Lib* određuje put do datoteke koja ostvaruje razrede knjižnice *Web Services Enhancements*. U prikazanom primjeru zadan je put *C:/ProgramInterpreter/bin/Debug/Microsoft.Web.Services2.dll*.



Slika 102: Struktura i sadržaj prividnog direktorija ostvarenog *Interpretatora programa*

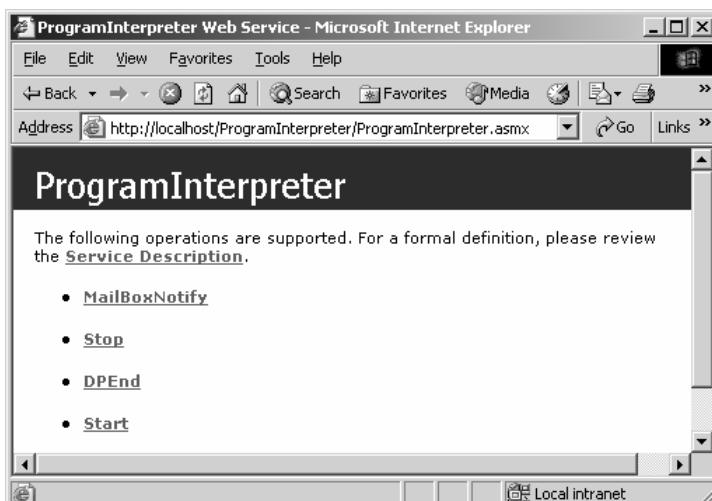
Nakon izgradnje fizičkog direktorija i definiranja postavki ostvarenog *Interpretatora programa* potrebno je prijaviti uslugu *Interpretatora programa* mrežnom poslužitelju sustava *Internet Information Services*. Slika 102 prikazuje strukturu prividnog direktorija *Interpretatora programa*. Prividni direktorij na računalu domaćinu moguće je stvoriti primjenom alata *mkwebdir.vbs* koji je dio potpore za održavanje *Internet Information Services* sustava. Prividni direktorij za ostvareni *Interpretator programa* gradi se primjenom *mkwebdir.vbs* na sljedeći način:

```
mkwebdir.vbs -w "Default Web Site" -v "ProgramInterpreter",
"C:\ProgramInterpreter"
```

Izvođenjem navedene naredbe na mrežnom poslužitelju računala domaćina stvara se prividni direktorij *ProgramInterpreter* koji pokazuje na fizički direktorij *C:\Program Interpreter*. Nakon stvaranja prividnog direktorija potrebno je primjenom alata *adsutil.vbs* i *chaccess.vbs* omogućiti javni pristup upravo izgrađenom prividnom direktoriju *Interpretatora programa* na računalu domaćinu. Alati se koriste na sljedeći način:

```
adsutil.vbs APPCREATEINPROC W3SVC/1/Root/ProgramInterpreter  
chaccess.vbs -a W3SVC/1/Root/ProgramInterpreter +execute
```

Nakon obavljenog opisanog postupka, postavljeni *Interpretator programa* na računalu domaćinu moguće je pristupiti putem adrese *http://localhost/ProgramInterpreter/ProgramInterpreter.asmx*. Slika 103 prikazuje sučelje postavljene usluge *Interpretatora programa* dohvaćeno primjenom preglednika Internet stranica.



Slika 103: Prikaz sučelja ostvarenog *Interpretatora programa*

Ostvareni *Interpretator programa* moguće je primjenom opisanog postupka postaviti na proizvoljan broj računala. Nakon što su postavljeni na željena radna računala, operacije postavljenih *Interpretatora programa* moguće je programski pozivati primjenom razreda zastupnika *Interpretatora programa*. Razred zastupnika *Interpretatora programa* moguće je automatski izgraditi primjenom alata *wsdl.exe* koji je dostupan u *.NET Framework* razvojnom okuženju. Datoteku s razredom zastupnika *Interpretatora programa* ostvarenim u jeziku *C#* moguće je automatski izgraditi primjenom alata *wsdl.exe* na sljedeći način:

```
wsdl.exe http://localhost/ProgramInterpreter/ProgramInterprete  
r.asmx?wsdl
```

Alat *wsdl.exe* kao rezultat izvođenja daje datoteku *ProgramInterpreter.cs* koja sadrži ostvarenje zastupnika *Interpretatora programa* u programskom jeziku *C#*.

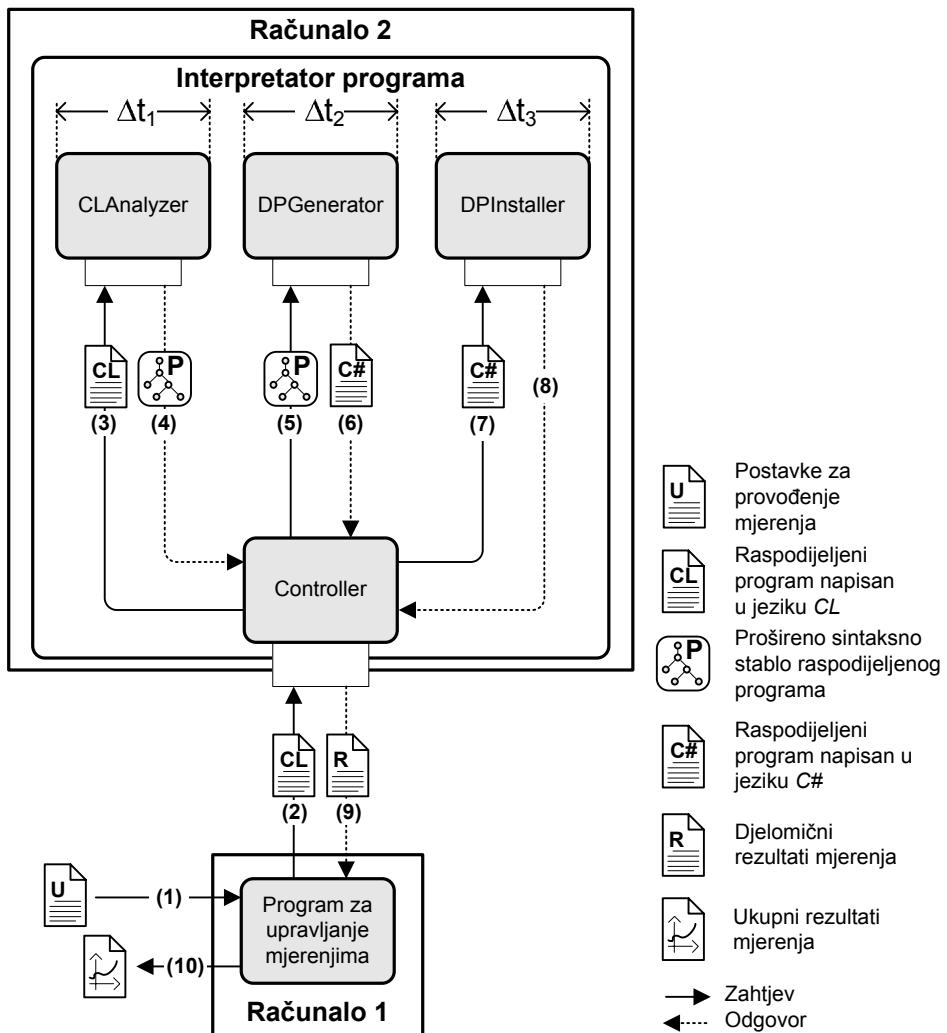
7. poglavlje

Mjerenje svojstava interpretatora programa

Svojstva *Interpretatora programa* određena su mjerenjem vremena prevodenja ispitnih raspodijeljenih programa napisanih u jeziku *CL*. Ispitni raspodijeljeni programi generirani su nasumično primjenom parametara *broj naredbi* i *broj aplikacijskih usluga*. Parametar *broj naredbi* određuje broj nasumično generiranih naredbi koje sadrži ispitni raspodijeljeni program. Parametar *broj aplikacijskih usluga* određuje broj nasumično generiranih *WSDL* opisa pristupnih sučelja aplikacijskih usluga koje raspodijeljeni program koristi tijekom izvođenja. Cilj provedenih mjerena je utvrditi utjecaj broja *CL* naredbi i broja aplikacijskih usluga na vrijeme potrebno za prevodenje raspodijeljenih programa iz jezika *CL* u jezik *C#*. Nadalje, cilj mjerena je također utvrditi utjecaj navedenih parametara na vrijeme potrebno za prevodenje raspodijeljenih programa iz jezika *C#* u jezik *CIL*.

7.1 Okolina za provođenje mjerena

Okolina za provođenje mjerena omogućava određivanje učinkovitosti rada ostvarenog *Interpretatora programa*. Osnovni element okoline je *Program za upravljanje mjerjenjima*. Namjena *Programa za upravljanje mjerjenjima* je generiranje ispitnih raspodijeljenih programa napisanih u jeziku *CL* prema zadanim parametrima i provođenje mjerena učinkovitosti rada *Interpretatora programa*. Ostvareni *Interpretator programa* proširen je dodanim naredbama za mjerene trajanje postupka prevodenja raspodijeljenih programa iz jezika *CL* u jezik *C#*. Mjerene trajanje postupka prevodenja raspodijeljenih programa iz jezika *CL* u jezik *C#* ostvareno je zbrajanjem rezultata mjerena trajanja postupka analize jezika *CL* i trajanja postupka sinteze raspodijeljenih programa napisanih u jeziku *C#*. Nadalje, ostvareni *Interpretator programa* također je proširen naredbama za mjerene trajanje postupka prevodenja raspodijeljenih programa iz jezika *C#* u jezik *CIL*.



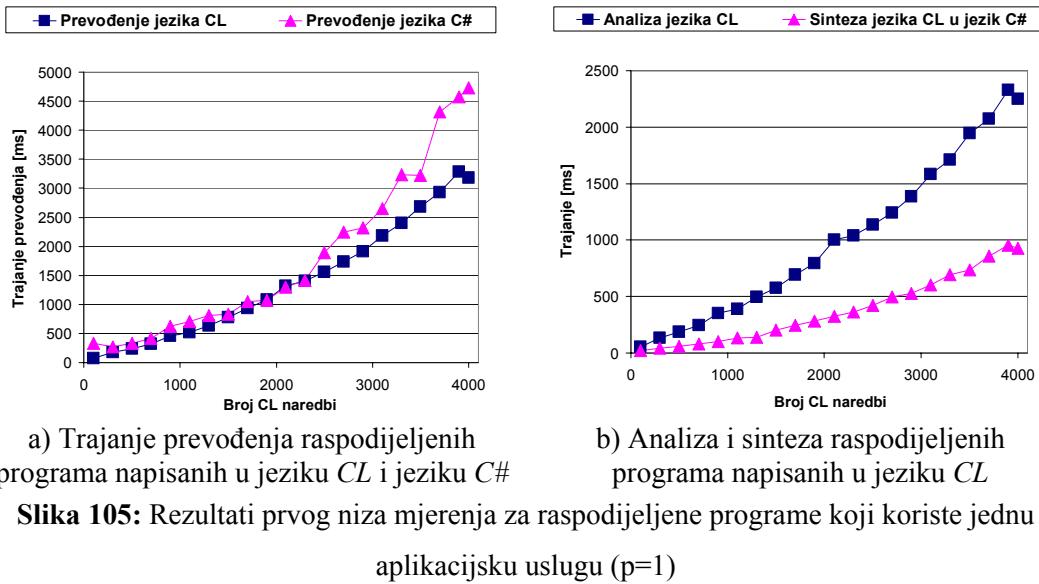
Slika 104: Okolina za provođenje mjerena

Slika 104 prikazuje okolinu za provođenje mjerena koja je postavljena na dva računala. Postupak mjerena započinje slanjem dokumenta s postavkama za provođenje mjerena *Programu za upravljanje mjerjenjima* (1). Postavke za provođenje mjerena sastoje se od parametara *broj naredbi* (n) i *broj aplikacijskih usluga* (p). Korištenjem zadanih parametara, *Program za upravljanje mjerjenjima* nasumično generira skup od ukupno $1 + \lceil n/100 \rceil$ ispitnih raspodijeljenih programa napisanih u jeziku CL. Generirani raspodijeljeni programi redom sadrže 1, 100, 200, 300, ..., $n-300$, $n-200$, $n-100$ i n naredbi. Svaki od nasumično generiranih ispitnih raspodijeljenih programa poziva p udaljenih aplikacijskih usluga. Nakon generiranja skupa ispitnih raspodijeljenih programa, *Program za upravljanje mjerjenjima* započinje zasebni ciklus mjerena za svaki generirani ispitni raspodijeljeni program. Na početku ciklusa mjerena, *Program za upravljanje mjerjenjima* proslijeđuje ispitni raspodijeljeni program koji sadrži 1 naredbu Interpretatoru programa (2). Podsustav Controller prihvata ispitni raspodijeljeni program i proslijeđuje ga podsustavu CLAnalyzer (3).

na analizu i izgradnju proširenog sintaksnog stabla. Podsustav *Controller* tijekom postupka analize mjeri vrijeme (Δt_1) potrebno za analizu raspodijeljenog programa napisanog u jeziku *CL* i izgradnju proširenog sintaksnog stabla raspodijeljenog programa. Nakon primitka proširenog sintaksnog stabla (4), podsustav *Controller* proslijedije izgrađeno sintaksno stablo podsustavu *DPGenerator* (5) na prevodenje u raspodijeljeni program napisan u jeziku *C#*. Podsustav *Controller* tijekom postupka prevodenja mjeri vrijeme (Δt_2) potrebno za izgradnju raspodijeljenog programa napisanog u jeziku *C#*. Nakon primitka raspodijeljenog programa napisanog u jeziku *C#* (6), podsustav *Controller* proslijediye primljeni raspodijeljeni program podsustavu *DPIInstaller* (7) koji prevodi raspodijeljeni program u *CIL* kôd. Podsustav *Controller* tijekom prevodenja mjeri vrijeme (Δt_3) potrebno za prevodenje raspodijeljenog programa iz jezika *C#* u jezik *CIL*. Nakon završetka postupka prevodenja (8), podsustav *Controller* proslijediye rezultate sa mjerenjima *Programu za upravljanje mjerenjima* (9) čime završava ciklus mjerjenja za jedan raspodijeljeni program. *Program za upravljanje mjerenjima* slijedno provodi opisani postupak za ostale raspodijeljene programe iz skupa generiranih ispitnih raspodijeljenih programa. Nakon završetka mjerjenja za sve ispitne raspodijeljene programe, *Program za upravljanje mjerenjima* objedinjuje primljene rezultate u jedinstveni dokument s ukupnim rezultatima mjerjenja (10).

7.2 Rezultati mjerjenja

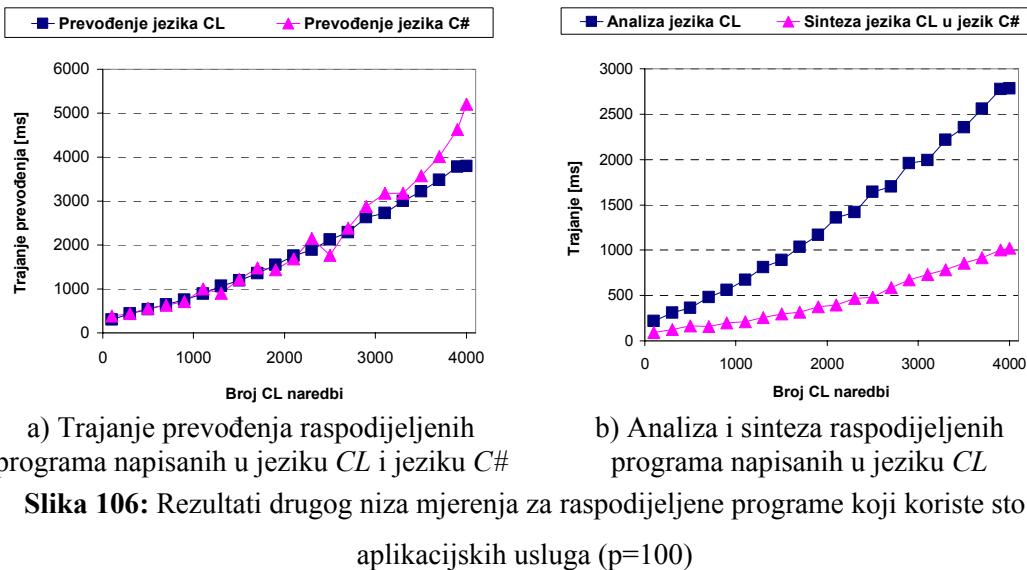
Mjerena su provedena primjenom osobnog računala s procesorom Intel Pentium 4 2.4 GHz i 512 MB radne memorije, te primjenom osobnog računala s procesorom AMD Athlon XP takta 1.83 GHz i 512 MB radne memorije. Prvo osobno računalo korišteno je za izvođenje *Programa za upravljanje mjerenjima*, dok je drugo osobno računalo korišteno za izvođenje ostvarenog *Interpretatora programa*. Kako bi se utvrdila učinkovitost rada ostvarenog *Interpretatora programa*, provedena su dva niza mjerjenja s različitim postavkama. Prvi niz mjerjenja proveden je za nasumično generirane raspodijeljene programe koji sadrže od 1 do 4000 *CL* naredbi i koriste jednu aplikacijsku uslugu. Mjerena su ponovljena pet puta, kako bi se umanjila pogreška zbog nesigurnosti tijekom provođenja mjerjenja, a konačni rezultat dobiven je izračunom srednje vrijednosti.



Slika 105: Rezultati prvog niza mjerena za raspolijeljene programe koji koriste jednu aplikacijsku uslugu ($p=1$)

Slika 105 prikazuje grafove s rezultatima prvog niza mjerena. Slika 105.a) prikazuje graf trajanja postupka prevodenja raspolijeljenih programa iz jezika *CL* u jezik *C#* i graf trajanja postupka prevodenja raspolijeljenih programa iz jezika *C#* u jezik *CIL* u ovisnosti o broju *CL* naredbi. Za raspolijeljene programe napisane u jeziku *CL* s manje od 2500 naredbi vrijeme potrebno za prevodenje u jezik *C#* neznatno je kraće od vremena potrebnog za prevodenje raspolijeljenih programa u jezik *CIL*. Za raspolijeljene programe s više od 2500 *CL* naredbi, trajanje postupka prevodenja raspolijeljenih programa u jezik *CIL* značajnije se povećava u odnosu na trajanje postupka prevodenja raspolijeljenih programa u jezik *C#*. Slika 105.b) prikazuje graf trajanja postupka analize raspolijeljenih programa napisanih u jeziku *CL* i graf trajanja postupka sinteze raspolijeljenih programa u jezik *C#* u ovisnosti o broju *CL* naredbi. U cijelom mjernom području postupak analize raspolijeljenih programa zahtijeva više vremena od postupka sinteze raspolijeljenih programa.

Drugi niz mjerena proveden je za nasumično generirane raspolijeljene programe koji sadrže od 1 do 4000 *CL* naredbi i koriste 100 aplikacijskih usluga. Broj aplikacijskih usluga povećan je u odnosu na prvi niz mjerena kako bi se odredio utjecaj povećanog broja aplikacijskih usluga na učinkovitost rada ostvarenog *Interpretatora programa*. Mjerenja su ponovljena pet puta, kako bi se umanjila pogreška zbog nesigurnosti tijekom provođenja mjerena, a konačni rezultat dobiven je izračunom srednje vrijednosti.



Slika 106: Rezultati drugog niza mjerena za raspolijeljene programe koji koriste sto aplikacijskih usluga ($p=100$)

Slika 106 prikazuje grafove s rezultatima drugog niza mjerena. Slika 106.a) prikazuje graf trajanja postupka prevodenja raspolijeljenih programa iz jezika *CL* u jezik *C#* i graf trajanja postupka prevođenja raspolijeljenih programa iz jezika *C#* u jezik *CIL* u ovisnosti o broju *CL* naredbi. U cijelom mjernom području, vrijeme potrebno za prevodenje raspolijeljenih programa iz jezika *CL* u jezik *C#* približno je jednako vremenu potrebnom za prevodenje raspolijeljenih programa iz jezika *C#* u jezik *CIL*. Slika 106.b) prikazuje graf trajanja postupaka analize raspolijeljenih programa napisanih u jeziku *CL* i trajanje postupka sinteze raspolijeljenih programa u jezik *C#* u ovisnosti o broju *CL* naredbi. U cijelom mjernom području postupak analize raspolijeljenih programa zahtjeva više vremena od postupka sinteze raspolijeljenih programa.

Na osnovi rezultata mjerena prikazanih na slika 105 i 106 vrijedi da je vrijeme potrebno za prevodenje raspolijeljenih programa iz jezika *CL* u jezik *C#* približno je jednako vremenu potrebnom za prevodenje raspolijeljenih programa iz jezika *C#* u jezik *CIL*. Postupak analize raspolijeljenih programa značajno je zahtjevniji od postupka sinteze raspolijeljenih programa u jezik *C#*. Postupak analize složeniji je od postupka sinteze zbog toga što analiza zahtjeva izvodenje složenog postupka parsiranja *XML* dokumenata u svrhu provjere ispravnosti raspolijeljenih programa napisanih u jeziku *CL*. Nadalje, postupak analize gradi sintaksno stablo i ostvaruje rekurzivni obilazak sintaksnog stabla u svrhu izgradnje proširenog sintaksnog stabla. Postupak sinteze raspolijeljenih programa napisanih u jeziku *C#* manje je zahtjevan u odnosu na postupak analize raspolijeljenih programa napisanih u jeziku *CL*. Postupak sinteze manje je zahtjevan u odnosu na postupak analize obzirom da ostvaruje samo jedan rekurzivni obilazak sintaksnog stabla raspolijeljenog

programa pri čemu se koriste kazaljke proširenog sintaksnog stabla radi bržeg pretraživnja stabla. Nadalje, postupak sinteze također gradi *C#* kôd primjenom gotovih predložaka napisanih u jeziku *C#*, te ne provodi složene analize *XML* dokumenata već koristi isključivo strukture podataka u memoriji računala.

Međusobnom usporedbom rezultata za dva različita niza provedenih mjerenja moguće je dodatno zaključiti da povećanje broja aplikacijskih usluga koje koristi raspodijeljeni program povećava ukupno trajanje postupka analize jezika *CL* u odnosu na trajanje postupka sinteze. Povećanje trajanja postupka analize uzrokovano je povećanim brojem *WSDL* dokumenata koje je potrebno analizirati tijekom analize raspodijeljenih programa napisanih u jeziku *CL*.

8. poglavje

Zaključak

Primjena načela računarstva zasnovanog na uslugama omogućava ubrzano i učinkovito oblikovanje i izgradnju raspodijeljenih aplikacija sa svojstvima razmjernog rasta i otpornosti na pogreške. Raspodijeljene aplikacije zasnovane na uslugama grade se međusobnim povezivanjem usluga koje su dostupne na globalnoj mreži Internet. Postupak povezivanja skupa usluga naziva se kompozicija usluga. Primjena kompozicije usluga omogućava izgradnju novih usluga koje ostvaruju skup složenih funkcionalnosti raspodijeljene aplikacije.

Ostvarivanje učinkovitih raspodijeljenih aplikacija zasnovanih na uslugama nalaže korištenje učinkovitog postupka kompozicije usluga. Učinkovitu kompoziciju usluga moguće je ostvariti osiguravanjem svojstava kvalitete, ispravnosti, sigurnosti i razmjernog rasta složene usluge. Svojstvo razmjernog rasta podrazumijeva učinkovito izvođenje složenih usluga i uslijed značajnog povećanja broja njihovih korisnika. Osnovni preduvjeti za ostvarivanje razmjernog rasta složenih usluga su učinkovito postavljanje, izvođenje i upravljanje tijekom izvođenja složenih usluga.

Razmjerni rast raspodijeljenih aplikacija moguće je ostvariti primjenom hibridne metode za izgradnju aplikacija koja je zasnovana na primjeni orkestracije i koreografije usluga. Raspodijeljene aplikacije izgrađene primjenom hibridne metode sadrže skup raspodijeljenih programa, aplikacijskih usluga i koordinacijskih mehanizama. Raspodijeljeni programi pozivaju aplikacijske usluge, te se međusobno natječu i surađuju primjenom koordinacijskih mehanizama kako bi ostvarili funkcionalnosti raspodijeljene aplikacije. Hibridna metoda za izgradnju raspodijeljenih aplikacija zasnovana je na primjeni raspodijeljenog upravljanja tijekom izvođenja aplikacija. Raspodijeljeno upravljanje tijekom izvođenja aplikacija omogućava izgradnju i izvođenje raspodijeljenih aplikacija sa svojstvima razmjernog rasta i otpornosti na pogreške.

U magistarskom radu definiran je jezik *CL* (*Coopetition Langauge*) za izgradnju raspodijeljenih aplikacija zasnovanih na uslugama. Nadalje, definirana je arhitektura i opisano programsko ostvarenje raspodijeljenog interpretatora programa koji omogućava postavljanje i izvođenje raspodijeljenih aplikacija zasnovanih na uslugama. Jezik *CL*

zasnovan je na primjeni jezika *WSDL* i podskupa naredbi jezika *BPEL4WS* koji omogućava opisivanje logike raspodijeljenih programa. Opisi raspodijeljenih programa napisani u jeziku *CL* su kompaktni, prenosivi i sadrže sve informacije koje su potrebne za njihovo izvođenje. Izvođenje raspodijeljnih aplikacija napisanih u jeziku *CL* ostvaruje se primjenom raspodijeljenog interpretatora programa. Raspodijeljeni interpretator ostvaren je kao skup jednostavnih i slabo povezanih interpretatora programa postavljenih na skupu radnih računala u globalnoj mreži Internet.

Korištenjem stečenih iskustava tijekom oblikovanja arhitekture i izgradnje raspodijeljenog interpretatora programa, budući rad u području raspodijeljenog interpretiranja programa bit će usredotočen na proširivanje modela za kompoziciju usluga. Cilj istraživanja bit će oblikovanje, izgradnja i ispitivanje mehanizama za ostvarivanje samoupravljivosti u programskim arhitekturama zasnovanim na uslugama. Poseban značaj u dalnjem radu bit će posvećen oblikovanju i razvoju odgovarajuće programske potpore koja omogućava krajnjem korisniku jednostavnu i učinkovitu izgradnju raspodijeljenih aplikacija zasnovanih na uslugama u globalnoj mreži Internet.

9. poglavlje

Literatura

- [1] H. El-Rewini, W. Halang: "**The Engineering of Complex Distributed Computer Systems**", *IEEE Concurrency*, Volume 5, Number 4, October/December 1997, pp. 30-31.
- [2] P. M. Sigh, M. A. Vouk: "**Network Computing**", *Encyclopedia of Electrical and Electronics Engineering*, Volume 14, *John Wiley & Sons*, 1999, pp. 114-132.
- [3] P. E. Ceruzzi: "**A History of Modern Computing**", *The MIT Press*, May 2003.
- [4] A. Sinha: "**Client-Server Computing**", *Communications of the ACM*, Volume 35, Number 7, July 1992, pp. 77-98.
- [5] S. Androulatsos-Theotokis, D. Spenellis: "**Survey of Peer-to-Peer Content Distribution Technologies**", *ACM Computing Surveys*, Volume 36, Number 4, December 2004, pp. 335-371.
- [6] J. O. Kephart, D. M. Chess: "**The Vision of Autonomic Computing**", *Computer*, Volume 36, Number 4, January 2003, pp. 41-50.
- [7] S. Srbljić: "**Jezični procesori 2: Analiza i sinteza ciljnog programa**", *Element*, 1. izdanje, siječanj 2002.
- [8] P. Wegner: "**Concepts and Paradigms of Object-Oriented Programming**", *ACM SIGPLAN OOPS Messenger*, Volume 1, Number 1, August 1990, pp. 7-87.
- [9] J. Hopkins: "**Component Primer**", *Communications of the ACM*, Volume 43, Number 10, October 2000, pp. 27-30.
- [10] P. Brereton, D. Budgen: "**Component-Based Systems: A Classification of Issues**", *Computer*, Volume 33, Number 11, November 2000, pp. 54-62.
- [11] M.P. Papazoglou, D. Georgakopoulos: "**Service Oriented Computing**", *Communications of the ACM*, Volume 46, Number 10, October 2003, pp. 25-28.
- [12] M. N. Huns, M. P. Singh: "**Service-Oriented Computing: Key Concepts and Principles**", *IEEE Internet Computing*, Volume 9, Number 1, January/February 2005, pp. 75-81.
- [13] A. Elfatatty, P. Layzell: "**Negotiating in Service-Oriented Environments**", *Communications of the ACM*, Volume 47, Number 8, August 2004, pp. 103-108.
- [14] J. N. Lee, M. Q. Huynh, R. C. W. Kwok, S. M. Pi: "**IT Outsourcing: Past, Present, and Future**", *Communications of the ACM*, Volume 46, Number 5, October 2003, pp. 84-89.

- [15] R. Sessions: "**Fuzzy Boundaries: Objects, Componentes, and Web Services**", *ACM Queue*, Volume 9, Number 9, December/January 2004/2005, pp. 40-47.
- [16] Microsoft .NET, (<http://www.microsoft.com/net/default.mspx>).
- [17] Java 2 Platform, Enterprise Edition, (<http://java.sun.com/j2ee/>).
- [18] M. Shaw, D. Garlan: "**Software Architecture: Perspectives on an Emerging Discipline**", *Pretence-Hall, Inc.*, April 1996.
- [19] M.P. Papazoglou: "**Service-Oriented Computing: Concepts, Characteristics and Directions**", *Fourth International Conference on Web Information Systems Engineering (WISE'03)*, December 2003, pp. 3-12.
- [20] M.P. Papazoglou, J. Dubray: "**A Survey of Web Service Technologies**", *University of Tereno*, Technical report #DIT-04-058, June 2004.
- [21] E. Newcomer, G. Lomow: "**Understanding SOA with Web Services**", *Addison Wesley Professional*, December 2004.
- [22] A. Dan, et al.: "**Web Services on Demand: WSLA-driven Automated management**", *IBM Systems Journal*, Volume 43, Number 1, January 2004, pp. 136-157.
- [23] D. A. Menascé: "**QoS Issues in Web Services**", *IEEE Internet Computing*, Volume 6, Number 6, November/December 2002, pp. 72-75.
- [24] M. P. Singh, A. K. Chopra, N. Desai, A. U. Mallya: "**Protocols for Processes: Programming in the Large for Open Systems**", *ACM SIGPLAN Notices*, Volume 39, Number 12, August 2004, pp. 73-83.
- [25] J. Sutherland, W. J. van den Heuvel: "**Enterprise Application Integration and Complex Adaptive Systems**", *Communications of the ACM*, Volume 45, Number 10, pp. 59-64, October 2002.
- [26] W. Hasselbring: "**Information System Integration**", *Communications of the ACM*, Volume 43, Number 6, June 2000, pp. 32-38.
- [27] J. Lee, K. Siau, S. Hong: "**Enterprise Integration with ERP and EAI**", *Communications of the ACM*, Volume 46, Number 2, February 2003, pp. 54-60.
- [28] G. Bell, J. Gray: "**What's Next in High Performance Computing**", *Communications of the ACM*, Volume 45, Number 2, February 2002, pp. 91-95.
- [29] I. Foster, C. Kesselman, S. Tuecke: "**The Anatomy of the Grid: Enabling Scalable Virtual Organizations**", *International Journal of Supercomputer Applications*, Volume 15, Number 3, 2001, pp. 200-222.

- [30] L. J. Zhang, H. Li, H. Lam: "**Toward a Business Process Grid for Utility Computing**", *IT Professional*, Volume 6, Number 5, September/October 2004, pp. 62-64.
- [31] L. J. Zhang, H. Li i H. Lam: "**Services Computing: Grid Applications for Today**", *IT Professional*, Volume 6, Number 4, July/August 2004, pp. 5-7.
- [32] R. Figueiredo, P. A. Dinda, J. Fortes: "**Resource Virtualization Renaissance**", *Computer*, Volume 38, Number 35, May 2005, pp. 28-31.
- [33] The Globus Alliance, (www.globus.org).
- [34] Globus Toolkit, (<http://www.globus.org/toolkit/>).
- [35] D. Talia: "**The Open Grid Service Architecture: Where the Grid Meets the Web**", *IEEE Internet Computing*, Volume 6, Number 6, November/December 2002, pp. 67-71.
- [36] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von Reich: "**The Open Grid Service Architecture**", *Global Grid Forum*, GFD-I.030, ver. 1.0, January 2005.
- [37] M. Champion, et al.: "**Web Services Architecture**", *W3C working draft*, November 2002, (www.w3.org/TR/ws-arch).
- [38] Karl Czajkowski, et al.: "**The WS-Resource Framework**", *Globus Alliance*, January 2004, (<http://www.globus.org/wsrf>).
- [39] M. A. Rappa: "**The Utility Business Model and the Future of Computing Services**", *IBM Systems Journal*, Volume 43, Number 1, January 2004, pp.32-42.
- [40] M. P. Papazoglou, B. J. Krämer, J. Yang: "**Leveraging Web-Services and Peer-to-Peer Networks**", *In Proceedings of the 15th Conference On Advanced Information Systems Engineering (CAiSE03)*, Klagenfurt/Velden, Austria, June 2003, pp. 485-501.
- [41] Electronic Business using eXtensible Markup Language (ebXML), July 2005, (www.ebxml.org).
- [42] S. Patil, E. Newcomer: "**ebXML and Web Services**", *IEEE Internet Computing*, Volume 7, Number 3, May/June 2003, pp. 74-82.
- [43] S. Vinoski: "**WS-Nonexistent Standards**", *IEEE Internet Computing*, Volume 8, Number 6, November/December 2004, pp.94-96.
- [44] T. Brady, J. Paoli, C.M. Sperberg-McQueen: "**Extensible Markup Langauge (XML) 1.0 Recommendation (Third edition)**", *World Wide Web Consortium (W3C)*, February 2004, (<http://www.w3.org/TR/REC-xml>).
- [45] D.C. Fallside, et al.: "**XML Schema Part 0: Premier Second Edition**", *W3C*, October 2004, (<http://www.w3.org/TR/xmlschema-0>).

- [46] E. Wilde, "**XML Technologies Dissected**", *IEEE Software*, Volume 7, Number 5, September/October 2003, pp. 74-78.
- [47] D. Box, et al.: "**SOAP 1.1**", *World Wide Web Consortium (W3C) Note*, February 2000., (<http://www.w3.org/TR/SOAP>).
- [48] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana: "**Web Services Description Language (WSDL) 1.1**", *World Wide Web Consortium (W3C) Note*, February 2001., (<http://www.w3.org/TR/wsdl>).
- [49] T. Bellwood, et al.: "**UDDI Version 3.0.2 Published specification**", October 2004., (http://uddi.org/pubs/uddi_v3.htm).
- [50] World Wide Web Consortium, (www.w3c.org).
- [51] Organization for the Advancement of Structured Information Standards (OASIS), (<http://www.oasis-open.org/home/index.php>).
- [52] Web Services Interoperability, (www.ws-i.org).
- [53] Steve Graham, et al.: "**Web Services Resource Properties (WS-ResourceProperties)**", IBM, Globus, Computer Associates, Hewlett-Packard, Fujitsu Laboratories of Europe, February 2004, (<http://www-128.ibm.com/developerworks/library/specification/ws-resource/ws-resourceproperties.pdf>).
- [54] Chris Kaler, et al.: "**Web Services Security (WS-Security)**", Microsoft, IBM, VeriSign, (<http://www-106.ibm.com/developerworks/webservices/library/ws-secure>).
- [55] Christopher Ferris, David Langworthy, et al.: "**Web Services Reliable Messaging Protocol (WS-ReliableMessaging)**", IBM, Microsoft, BEA, TIBCO, February 2005, ([ftp://www6.software.ibm.com/software/developer/library/ws-reliabilemessagin_g2005_02.pdf](http://www6.software.ibm.com/software/developer/library/ws-reliabilemessagin_g2005_02.pdf)).
- [56] Web Services Transactions specifications, (<http://www-128.ibm.com/developerworks/webservices/library/specification/ws-tx>).
- [57] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic i S. Weerawarana. "**Business Process Execution Language for Web Services (BPEL4WS) 1.1**", May 2003, (<http://www-106.ibm.com/developerworks/webservices/library/ws-bpel>).
- [58] N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon: "**Web Services Choreography Description Language**", version 1.0, *World Wide Web Consortium (W3C) Working Draft*, October 2004., (<http://www.w3.org/TR/ws-cdl-10>).
- [59] Don Box, Francisco Curbera, et al.: "**Web Services Addressing (WS-Addressing)**", Microsoft, IBM, SUN, BEA, SAP, August 2004, (<http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810>).

- [60] S. Vinoski, "**Web Service References**", *IEEE Internet Computing*, Volume 9, Number 2, March/April 2005, pp. 94-96.
- [61] I. Foster, et al.: "**Modeling Stateful Resources with Web Services**", *Globus Alliance*, ver. 1.1, March 2004., (<http://www-128.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf>).
- [62] Jeffrey Frey, Steve Graham, et al.: "**Web Services Resource Lifetime (WS-ResourceLifetime)**", IBM, Globus, Computer Associates, Hewlett-Packard, Fujitsu Laboratories of Europe, March 2004, (<http://www-128.ibm.com/developerworks/library/specification/ws-resource/ws-resourcelifetime.pdf>).
- [63] Steve Graham, Tom Maguire, et al.: "**Web Services Service Group (WS-ServiceGroup)**", IBM, Computer Associates International, Fujitsu Laboratories of Europe, Globus, Hewlett-Packard, March 2004, (<http://www-128.ibm.com/developerworks/library/ws-resource/ws-servicegroup.pdf>).
- [64] Steve Tuecke, et al.: "**Web Services Base Faults (WS-BaseFaults)**", Globus, IBM, Computer Associates International, Fujitsu Laboratories of Europe, Hewlett-Packard, March 2004, (<http://www-128.ibm.com/developerworks/library/specification/ws-resource/ws-basefaults.pdf>).
- [65] United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT), (<http://www.unece.org/cefact>).
- [66] Selim Aissi, et al.: "**Collaboration-Protocol Profile and Agreement Specification**", OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee, ver. 2.0, September 2002, (<http://www.oasis-open.org/committees/ees/ebxml-cppa/documents/ebcpp-2.0.pdf>)
- [67] Jim Clark, et al.: "**ebXML Business Process Specification Schema**", Business Process Project Team, May 2001, (<http://www.ebxml.org/specs/ebBPSS.pdf>).
- [68] Mark Potts, Bill Cox i Bill Pope: "**Business Transaction Protocol: Primer**", OASIS, June 2002, (<http://www.oasis-open.org/committees/business-transactions/documents/primer/Primerhtml/BTP%20Primer%20D1%2020020602.html>)
- [69] Assaf Arkin: "**Business Process Modeling Language**", *Italiano*, November 2002., (<http://www.bpmi.org/BPML.htm>).
- [70] N.Milanovic, M.Malek: "**Current Solutions for Web Service Composition**", *IEEE Internet Computing*, Volume 8, Number 6, November/December 2004, pp.51-59.
- [71] Steve Vinoski: "**RPC Under Fire**", *IEEE Internet Computing*, Volume 9, Number 5, September/October 2005, pp. 93-95.

- [72] D.A. Menascé: "**Composing Web Services: A QoS View**", *IEEE Internet Computing*, Volume 8, Number 6, November/December 2004, pp. 88-90.
- [73] S. Narayanan, S.A. McIlraith: "**Simulation, Verification and Automated Composition of Web Services**", *In Proceedings of the 11th international conference on World Wide Web*, Honolulu, Hawaii, USA, May 2002, pp. 77-88.
- [74] J. Rao, X. Su: "**A Survey of Automated Web Service Composition Methods**", *In Proceedings of First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA, July 2004.
- [75] C. Peltz: "**Web Services Orchestration and Choreography**", *Computer*, Volume 36, Number 10, October 2003, pp. 46-52.
- [76] M. Solanki, C. Abela: "**The Landscape of Markup Languages for Web Service Composition**", May 2003.
- [77] W.M.P. van der Aalst, M. Dumas i A.H.M. ter Hofstede: "**Web Service Composition Languages: Old Wine in New Bottles?**", *In Proceeding of the 29th EUROMICRO Conference*, Belek near Antalya, Turkey, September 2003, pp. 298-305.
- [78] S. Thatte: "**XLANG: Web Services for Business Process Design**", Microsoft, 2001.
- [79] F. Leymann: "**Web Services Flow Langauges (WSFL)**", *IBM Software Group*, ver. 1.0, 2001.
- [80] D. Chappell: "**Understanding BizTalk Server 2004**", *BizTalk Server 2004 Technical Article*, Microsoft, February 2004.
- [81] OASIS Web Services Business Process Execution Language (WSBPEL) TC, (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel).
- [82] J. Yang: "**Web Service Componentization**", *Communications of the ACM*, Volume 46, Number 10, October 2003, pp. 35-40.
- [83] C.A. Petri: "**Kommunikation mit Automaten. Schriften des IIM Nr. 2**", Institut für Instrumentelle Mathematik, Bonn, 1962. prevedeno na engleski jezik: Technical Report RADC-TR-65-377, Griffiths Air Force Base, New York, Volume 1, Number 1, 1966.
- [84] R. Hamadi, B. Benatallah: "**A Petri Net-based Model for Web Service Composition**", *In Proceedings of the Fourteenth Australasian Database Conference on Database Technologies (ADC2003)*, Adelaide, Australia, January 2003, pp. 191-200.
- [85] K. Jensen: "**Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use**", Volume 1, *Basic Concepts, Monographs in Theoretical Computer Science*, Springer-Verlag, 1997.

- [86] W.M.P. van der Aalst: "**The Application of Petri-Nets to Workflow Management**", *The Journal of Circuits, Systems and Computers*, Volume 8, Number 1, February 1998, pp. 21-66.
- [87] K. Jensen: "**Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use**", Volume 2, *Anaysis methods*, Monographs in Theoretical Computer Science, Springer-Verlag, 1997.
- [88] J.C.M. Baeten: "**A Brief History of Process Algebra**", *Vakgroep Informatica*, Technische Universiteit Eindhoven, Rapport CSR 04-02, 2004.
- [89] R. Milner: "**The Polyadic π -Calculus: a Tutorial**", *Proceedings of the International Summer School on Logic Algebra of Specification*, Springer Verlag, Berlin, Germany, 1992, pp. 203-246, (<ftp://ftp.cl.cam.ac.uk/users/rm135/ppi.ps.Z>).
- [90] J. F. Groote i M. A. Reniers: "**Algebraic Process Verification**", *Handbook of Process Algebra*, Elsevier, Amsterdam, 2001, pp. 1151-1208.
- [91] L.G. Meredith i Steve Bjorg: "**Contracts and Types**", *Communications of the ACM*, Volume 46, Number 10, October 2003, pp. 41-47.
- [92] D. Berardi, G. De Giacomo, M. Lenzerini, M. Mecella, D. Calvanese: "**Synthesis of Underspecified Composite e-Services based on Automated Reasoning**", *In Proceedings of the 2nd International Conference on Service-Oriented Computing* (ICSOC 2004), New York, NY, USA, November 2004, pp. 105-114.
- [93] D. Bodoff, P. C. K. Hung, M. Ben-Menachem, "**Web Metadata Standards: Observations and Prescriptions**", *IEEE Software*, Volume 22, Number 1, January/February 2005, pp. 78-85.
- [94] W. R. King, P. V. Marks, S. Mccoy: "**The Most Imporatnt Issues in Knowledge Management**", *Communications of the ACM*, Volume 45, Number 9, September 2002, pp. 93-97.
- [95] T. Berners-Lee, J. Hendler, O. Lassila: "**The Semantic Web**", *Scientific American*, May 2001, pp. 34-43.
- [96] G. Klyne, J. Carroll: "**Resource Description Framework (RDF): Concepts and Abstract Syntax**". January 2003, (<http://www.w3.org/TR/rdf-concepts>).
- [97] M. Gruninger, J. Lee: "**Ontology: Applications and Design**", *Communications of the ACM*, Volume 45, Number 2, October 2002, pp. 39-41.
- [98] S. A. McIlraith, T. C. Son, H. Zeng: "**Semantic Web Services**", *IEEE Intelligent Systems*, Volume 16, Number 2, March/April 2001, pp. 46-53.
- [99] OWL-S Coalition: "**OWL-S 1.0 Release**", (<http://www.daml.org/services/owl-s/1.0/>).

- [100] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, Katia Sycara: "**Bringing Semantics to Web Services: The OWL-S Approach**", *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA, July 2004.
- [101] Deborah L. McGuinness, Frank van Harmelen. "**OWL Web Ontology Language Overview**". *World Wide Web Consortium (W3C) Candidate Recommendation*. October 2003, (<http://www.w3.org/TR/owl-features/>).
- [102] T. Kawamura, T. Hasegawa, A. Ohsuga, M. Paolucci, Katia Sycara: "**Web Services Lookup: A Matchmaker Experiment**", *IT Professional*, Volume 7, Number 2, March/April 2005, pp. 36-41.
- [103] James Clark: "**XSL Transformations (XSLT)**", ver. 1.0, November 1999., (<http://www.w3.org/TR/xslt>)
- [104] Yolanda Gil, Ewa Deelman, Jim Blythe, Carl Kesselman, and Hongsuda Tangmunarunkit: "**Artificial Intelligence and Grids: Workflow Planning and Beyond**", *IEEE Intelligent Systems*, Volume 19, Number 1, January/February 2005, pp.26-33.
- [105] M. B. Juric, B. Mathew, P. Sarang: "**Business Process Execution Language for Web Services**", 4. chapter: "*Oracle BPEL Process Manager*", Packt Publishing, October 2004.
- [106] Scott Boag, et al.: "**XQuery 1.0: An XML Query Language**", IBM Research, AT&T Labs, Oracle, DataDirect Technologies, September 2005. (<http://www.w3.org/TR/2005/WD-xquery-20050915>).
- [107] XSQL - Combining XML and SQL, (<http://xsql.sourceforge.net>).
- [108] SQL Server Home, (<http://www.microsoft.com/sql/default.mspx>).
- [109] DB2 Product family, (<http://www-306.ibm.com/software/data/db2>).
- [110] Oracle database, (<http://www.oracle.com/database/index.html>).
- [111] ActiveBPEL, (<http://www.activebpel.org>).
- [112] Boualem Benatallah, Quan Z. Sheng, Marlon Dumas: "**The Self-Serv Environment for Web Services Composition**", *IEEE Internet Computing*, Volume 7, Number 1, January/February 2003, pp.40-48.
- [113] Object Management Group – UML, (www.uml.org).
- [114] D. Skrobo, A. Milanović, S. Srbljić: "**Distributed Program Interpretation in Service-Oriented Distributed Systems**", *Proceedings of 9th World Multi-*

- Conference on Systemics, Cybernetics and Informatics (WMSCI'05)*, Orlando, Florida, USA, July 2005, pp. 193-197.
- [115] A. Tanenbaum, M. van Steen: "**Distributed Systems: Principles and Paradigms**", *Pretence-Hall*, January 2002.
- [116] A. Milanović, S. Srbljić, D. Skrobo, D. Čapalija, S. Rešković: "**Coopetition Mechanisms for Service-Oriented Distributed Systems**", *Proceedings of 3th International Conference on Computing, Communications and Control Technologies, Cybernetics and Informatics (CCCT'05)*, Austin, Texas, USA, July 2005, pp. 118-123.
- [117] S. Rešković: "**Sinkronizacijski i komunikacijski mehanizmi za sustave zasnovane na uslugama**", Diplomski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, Zagreb, rujan 2005.
- [118] D. Čapalija: "**Objava-preplata mehanizmi za ostvarivanje mreža zasnovanih na sadržaju**", Diplomski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, Zagreb, lipanj 2005.
- [119] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, A. P. Barros: "**Workflow Patterns**", *Distributed and Parallel Databases*, Volume 14, Number 1July 2003, pp. 5-51.,
- [120] Paul V. Biron, et al.: "**XML Schema Part 2: Datatypes Second Edition**", Kaiser Permanente, for Health Level Seven i Microsoft, August 2004., (<http://www.w3.org/TR/xmlschema-2/#dateTime>).
- [121] J. Aycock: "**A Brief History of Just-In-Time**", *ACM Computing Surveys*, Volume 35, Number 2, June 2003, pp. 97–113.
- [122] D. Feitelson, L. Rudolph, U. Schwiegelshohn, "**Parallel Job Scheduling a Status Report**", *10th Workshop on Job Scheduling Strategies for Parallel Processing*, Columbia University, New York, NY, June 2004.
- [123] J. H. Abawajy, S. P. Dandamudi: "**Parallel Job Scheduling on Multicluster Computer Systems**", *In Proceedings of 5th IEEE International Conference on Cluster Computing (CLUSTER'03)*, Hong Kong, China, December 2003), pp. 11-18.
- [124] ASP.NET, (<http://asp.net>).
- [125] Web Services Enhancements, (<http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx>).
- [126] Internet Information Services, (<http://www.microsoft.com/WindowsServer2003/iis/default.mspx>).

10. poglavlje

Životopis

Rođen sam 20. prosinca 1979. godine u Zagrebu, gdje pohađam osnovnu školu i srednju elektrotehničku školu. Nakon mature 1998. godine upisujem dodiplomski studij na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Za vrijeme studija radim na nekoliko studentskih projekata Grupe računarski sustavi i procesi na Zavodu za automatiku i procesno računarstvo i na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave. Tijekom posljednje dvije godine studija radim kao demonstrator na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave na laboratorijskim vježbama kolegija: "Automati, formalni jezici i jezični procesori I" i "Automati, formalni jezici i jezični procesori II".

U studenom 2001. godine Znanstveno-nastavno vijeće Fakulteta elektrotehnike i računarstva odobrava mi upis završetaka studija s naglaskom na znanstveno-istraživačkom radu. Diplomirao sam u rujnu 2003. studij računarstva. Diplomski rad ostvarujem u području presretanja Internet telefonije s naslovom "Presretanje telefonskog prometa u lokalnoj mreži zasnovanoj na IP protokolu". Rezultate diplomskog rada ostvarujem u sklopu projekta u suradnji sa tvrtkom Cisco Systems, Inc., USA, a pod vodstvom Prof. dr. sc. Siniše Srblića.

Akademске godine 2003/2004 upisujem poslijediplomski studij za stjecanje akademskog stupnja magistra znanosti. Od siječnja 2004. godine zaposlen sam na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu, u sklopu tehnološkog projekta TP-01/036-29 "Okrilje posrednika javnog informacijskog sustava". U sklopu suradnje Fakulteta elektrotehnike i računarstva i tvrtke Ericsson Nikola Tesla sudjelujem u izvedbi projekata "Ericsson Summer Camp 2004" i "Ericsson Summer Camp 2005" kao voditelj studenata dodiplomske nastave. U okviru nastavnih aktivnosti na Zavodu, aktivno sudjelujem u provedbi laboratorijskih vježbi iz kolegija "Digitalna elektronika" i "Operacijski sustavi 1", "Automati, formalni jezici i jezični procesori I" i "Automati, formalni jezici i jezični procesori II".

U svojem dosadašnjem znanstveno-istraživačkom radu na međunarodnim konferencijama sudjelovao sam s četiri članka. Dva članka objavio sam u području legalnog presretanja prometa Internet telefonije i dva članka u području raspodijeljenog računarstva zasnovanog na uslugama.

11. poglavlje

Sažetak

Programske arhitekture zasnovane na uslugama imaju sve važniju ulogu za oblikovanje i izgradnju računalnih sustava velikih razmjera u globalnoj mreži Internet. Raspodijeljene aplikacije zasnovane na uslugama u globalnoj mreži Internet moguće je graditi primjenom skupa aplikacijskih usluga, raspodijeljenih programa i koordinacijskih mehanizama. Raspodijeljeni programi pozivaju aplikacijske usluge, te se međusobno natječe i suraduju primjenom koordinacijskih mehanizama kako bi ostvarili funkcionalnosti raspodijeljene aplikacije.

U magistarskom radu definiran je jezik CL (*Coopetition Langauge, CL*) koji omogućava izgradnju raspodijeljenih aplikacija zasnovanih na uslugama. Nadalje, definirana je arhitektura i opisano programsko ostvarenje raspodijeljenog interpretatora programa za postavljanje i izvođenje raspodijeljenih aplikacija. Jezik CL definiran je primjenom jezika *WSDL (Web Service Description Language)* i podskupa naredbi jezika *BPEL4WS (Business Process Execution Language for Web Services)* koji omogućava opisivanje logike raspodijeljenih programa. Raspodijeljeni programi napisani u jeziku CL su kompaktni, prenosivi i sadrže sve informacije potrebne za njihovo interpretiranje. Izvođenje raspodijeljenih aplikacija napisanih u jeziku CL ostvaruje se primjenom raspodijeljenog interpretatora programa. Raspodijeljeni interpretator ostvaren je kao skup jednostavnih i slabo povezanih interpretatora programa postavljenih na skupu radnih računala u globalnoj mreži Internet. Primjena jezika CL i ostvarenog raspodijeljenog interpretatora omogućava izgradnju, postavljanje i izvođenje raspodijeljenih aplikacija zasnovanih na uslugama sa svojstvima razmernog rasta i otpornosti na pogreške.

12. poglavje

Summary

"Distributed Parallel Program Interpretation in Service-Oriented Architectures"

Service-oriented architectures have an increasingly important role in the design and development of large-scale distributed systems on the Internet. Service-oriented applications can be developed on the Internet by using a set of application services, distributed programs, and coordination mechanisms. Distributed programs invoke application services, and mutually cooperate and compete by using coordination mechanisms in order to implement distributed application functionalities.

This thesis defines a service composition language, named Coopetition Language (*CL*), which enables development of service-oriented distributed applications. Furthermore, the thesis defines architecture and describes implementation of a distributed program interpreter that can be used for deployment and execution of distributed applications developed using *CL*. Language *CL* is based on combination of *WSDL (Web Service Description Language)* and a subset of *BPEL4WS (Business Process Execution Language for Web Services)*, which is used for describing the logic of the distributed programs. Distributed programs built using language *CL* are compact, mobile, and contain all information required for their interpretation. Distributed applications built by using *CL* are executed on distributed program interpreter. Distributed program interpreter is implemented using a set of light-weight and loosely-coupled program interpreters deployed as services on a set of hosts available on the Internet. Language *CL* used in conjunction with distributed program interpreter enables development, deployment, and execution of large-scale and fault tolerant service-oriented distributed applications.

13. poglavljje

Ključne riječi

Interpretiranje programa, raspodijeljeni sustavi zasnovani na uslugma, arhitektura zasnovana na uslugama, kompozicija usluga zasnovana na opisima procesa.

Program interpretation, service-oriented distributed systems, service-oriented architecture, process-oriented service composition.

Dodatak A

Definicija programskog jezika za suradnju i natjecanje

U ovom dodatku prikazana je definicija programskog jezika za suradnju i natjecanje. Definicija jezika za suradnju i natjecanje opisana je *XML Schema* dokumentom prikazanim na slici A.1.

```

<s:schema
    elementFormDefault="qualified"
    targetNamespace="http://www.ris.fer.hr/OpenCollectives/CoopetitionLanguageCode"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.ris.fer.hr/OpenCollectives/CoopetitionLanguageCode"
    xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

    <s:import
        namespace="http://schemas.xmlsoap.org/ws/2003/03/business-process/" />
    <s:import
        namespace="http://schemas.xmlsoap.org/wsdl/" />

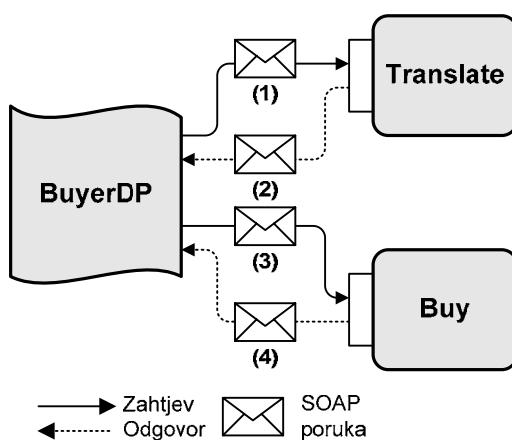
    <s:element name="dpCode" nillable="true" type="tns:DPCode" />
    <s:complexType name="DPCode">
        <s:sequence>
            <s:element minOccurs="1"
                       maxOccurs="1"
                       name="process"
                       ref="bpel:process" />
            <s:element minOccurs="0"
                       maxOccurs="1"
                       name="ProcessDefinitions"
                       type="tns:ProcessDefinitions" />
            <s:element minOccurs="0"
                       maxOccurs="1"
                       name="PartnerDefinitions"
                       type="tns:PartnerDefinitions" />
        </s:sequence>
    </s:complexType>
    <s:complexType name="ProcessDefinitions">
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" ref="wsdl:definitions" />
        </s:sequence>
    </s:complexType>
    <s:complexType name="PartnerDefinitions">
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="unbounded" ref="wsdl:definitions" />
        </s:sequence>
    </s:complexType>
</s:schema>
```

Slika A.1: XML Schema definicija programskog jezika CL

Dodatak B

Primjer raspodijeljenog programa napisanog u jeziku za suradnju i natjecanje

U ovom dodatku prikazan je primjer raspodijeljenog programa napisanog primjenom jezika za suradnju i natjecanje.



Slika B.1: Prikaz tijeka izvođenja raspodijeljenog programa *BuyerDP*

Slika B.1 prikazuje primjer raspodijeljenog programa *BuyerDP*. Raspodijeljeni program *BuyerDP* koristi uslugu *Translate* i uslugu *Buy*. Usluga *Translate* omoguća prevodenje rečenica iz hrvatskog u engleski jezik, dok usluga *Buy* omogućava kupovinu knjiga putem globalne mreže Internet. Raspodijeljeni program *BuyerDP* na početku izvođenja poziva uslugu *Translate* kako bi preveo rečenicu "Turistički vodič grada Zagreba" (1) i prihvaća prevod rečenice na engleski jezik (2). Nakon prevodenja rečenice, raspodijeljeni program *BuyerDP* poziva uslugu *Buy* (3) kako bi ostvario kupovinu knjige s zadanim naslovom. Nakon uspješne kupovine, raspodijeljeni program *BuyerDP* prihvaci pozitivnu potvrdu od usluge *Buy* i završava sa izvođenjem. Opis najvažnijih elemenata ostvarenja raspodijeljenog programa *BuyerDP* primjenom jezika *CL* prikazan je u tablici B.1. Zbog ograničenosti u prostoru, *WSDL* opisi pristupnog sučelja raspodijeljenog programa *BuyerDP* i *WSDL* opisi sučelja usluga *Translate* i *Buy* su izostavljeni.

Tablica B.1: Opis raspodijeljenog programa *BuyerDP* ostvaren primjenom jezika *CL*

CL kôd raspodijeljenog programa	Opis
<dpCode xmlns="...">	Početak opisa raspodijeljenog programa <i>BuyerDP</i> u jeziku CL
<process name="BuyerDP" targetNamespace="..." xmlns="..." xmlns:tns="..." xmlns:abs="..." xmlns:gts="..." xmlns:wsa="...">	Početak opisa procesne logike u jeziku BPEL4WS
<variables> <variable name="Translate_Req" messageType="gts:TranslateSoapIn" /> <variable name="Translate_Res" messageType="gts:TranslateSoapOut" /> <variable name="Buy_Req" messageType="abs:AmazonSoapIn" /> <variable name="Buy_Res" messageType="abs:AmazonSoapOut" /> </variables>	Deklaracije skraćenica za XML prostore imena korištenih u programu Deklaracije varijabli koje koristi raspodijeljeni program.
<partnerLinks> <partnerLink name="Translate" role="TranslateService" partnerLinkType="tns:GoogleServicePLT" /> <partnerLink name="Buy" role="BuyService" partnerLinkType="tns:AmazonSoapPLT" /> </partnerLinks>	Deklaracije simboličkih imena za naslovljavanje usluga Translate i Buy koje raspodijeljeni program koristi tijekom izvođenja.
<sequence>	Početak bloka naredbi procesne logike raspodijeljenog programa
<assign>	Naredba za upravljanje vrijednostima varijabli
<copy> <from> <gts:Translate> <gts:keywords> Turistički vodič grada Zagreba</gts:keywords> </gts:Translate> </from> <to variable="Translate_Req" part="parameters" /> </copy>	Priprema sadržaja SOAP poruke koju raspodijeljeni program šalje usluzi translate.
<copy> <from> <wsa:EndpointReference> <wsa:Address>http://www.google.com/Translate.asmx</wsa:Address> </wsa:EndpointReference> </from> <to partnerLink="Translate" /> </copy>	Pridruživanje adrese usluge Translate simboličkoj varijabli za naslovljavanje usluge Translate. Pridružuje se adresa http://www.google.com/Translate.asmx.
<copy> <from> <wsa:EndpointReference> <wsa:Address>http://www.amazon.com/Buy.asmx</wsa:Address> </wsa:EndpointReference> </from> <to partnerLink="Buy" /> </copy>	Pridruživanje adrese usluge Buy simboličkoj varijabli za naslovljavanje usluge Buy. Pridružuje se adresa http://www.amazon.com/Buy.asmx.
</assign>	Završetak naredbe za upravljanje vrijednostima varijabli.
<invoke partnerLink="Translate"	Poziv operacije Translate usluge Google. Varijabla Translate_req

<pre> portType="gts:TranslateSoap" operation="Translate" inputVariable="Translate Req" outputVariable="Translate Res" /> <assign> <copy> <from variable="Translate Res" part="parameters" query="/TranslateResponse" /> <to variable="Buy" part="parameters" query="/Buy" /> </copy> </assign> <invoke partnerLink="Buy" portType="abs:BuySoap" operation="Buy" inputVariable="Buy Req" outputVariable="Buy Res" /> </sequence> </process> <processDefinitions> <definitions xmlns="..." /> </processDefinitions> <partnerDefinitions> <definitions xmlns="..." targetNamespace=".." /> <definitions xmlns="..." targetNamespace=".." /> </partnerDefinitions> </dpCode></pre>	<p>sadrži SOAP poruku zahtjeva, a varijabla Translate Res primljenu SOAP poruku odgovora.</p> <p>Preslikavanje vrijednosti sadržaja SOAP poruke odgovora primljene od usluge Translate u varijablu koja sadrži SOAP poruku zahtjeva koja će biti upućena usluzi Buy.</p> <p>Poziv operacije Buy usluge. Varijabla Buy_req sadrži SOAP poruku zahtjeva, a varijabla Buy_Res primljenu SOAP poruku odgovora.</p> <p>Kraj bloka naredbi procesne logike.</p> <p>Završetak opisa procesne logike</p> <p>Element koji sadrži WSDL opis sučelja raspodijeljenog programa.</p> <p>Element koji sadrži WSDL opise sučelja svih usluga koje koristi raspodijeljeni program.</p> <p>Element koji sadrži WSDL opis sučelja usluge Translate.</p> <p>Element koji sadrži WSDL opis sučelja usluge Buy.</p> <p>Kraj elementa koji sadrži WSDL opise usluga koje koristi raspodijeljeni program.</p> <p>Kraj opisa raspodijeljenog programa u jeziku CL</p>
---	---