

SVEUČILIŠTE U ZAGREBU  
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Ivan Gavran

**KORISNIČKI JEZIK PROGRAMSKOG  
MODEL A ZASNOVANOG 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 141 stranicu.

Rad br.: \_\_\_\_\_

Povjerenstvo za ocjenu u sastavu:

1. Prof. dr. sc. Nikola Bogunović
2. Prof. dr. sc. Siniša Srbljić
3. Doc. dr. sc. Saša Dešić

Povjerenstvo za obranu u sastavu:

1. Prof. dr. sc. Nikola Bogunović
2. Prof. dr. sc. Siniša Srbljić
3. Doc. dr. sc. Saša Dešić

Datum obrane: 22. 03. 2006.

## **Zahvala**

*Zahvaljujem prof. dr. sc. Siniši Srbljiću na pruženoj prilici za rad na CroGrid projektu, na stručnom vodstvu tijekom izrade magistarskog rada, te na prenesenim životnim iskustvima koji su bili velika motivacija za nastavak školovanja i završetak ovog rada.*

*Zahvaljujem dr. sc. Andri Milanoviću na pomoći tijekom pisanja magistarskog rada i Danielu Skrobi dipl. ing. na pruženoj pomoći tijekom izrade praktičnog dijela rada.*

*Zahvaljujem mr. sc. Ivanu Bencu i mr. sc. Ivanu Skuliberu, Matiji Podravcu dipl. ing., Miroslavu Popoviću dipl. ing. i Dejanu Škvorcu dipl. ing., za svu pomoć koju su mi pružili tijekom zajedničkog rada na CroGrid projektu.*

*Velika zahvala obitelji za svu potporu, razumijevanje i savjete koji su mi omogućili nastavak školovanja i doprinijeli uspješnom završetku ovog rada.*

## SADRŽAJ

<b>1</b>	<b>UVOD .....</b>	<b>1</b>
<b>2</b>	<b>RAČUNARSTVO ZASNOVANO NA USLUGAMA.....</b>	<b>3</b>
2.1	MREŽNE USLUGE .....	3
2.1.1	<i>Modeli medudjelovanja mrežnih usluga.....</i>	5
2.1.2	<i>Svojstva mrežnih usluga .....</i>	7
2.2	ARHITEKTURA ZASNOVANA NA USLUGAMA.....	9
2.2.1	<i>Osnovni model arhitekture zasnovane na uslugama .....</i>	10
2.2.2	<i>Prošireni model arhitekture zasnovane na uslugama .....</i>	11
2.3	WEB SERVICES TEHNOLOGIJE OSTVARENJA SUSTAVA ZASNOVANIH NA USLUGAMA.....	14
2.3.1	<i>Osnovne Web Services tehnologije.....</i>	14
2.3.2	<i>Napredne Web Services tehnologije .....</i>	20
2.4	SEMANTIČKE MREŽNE USLUGE .....	22
2.4.1	<i>Norme semantičkih mrežnih usluga.....</i>	23
2.5	PROGRAMSKI MODEL OBLIKOVANJA SUSTAVA ZASNOVANIH NA USLUGAMA.....	25
2.5.1	<i>Odnos objekta, komponente i mrežne usluge.....</i>	26
<b>3</b>	<b>KOMPOZICIJA MREŽNIH USLUGA .....</b>	<b>29</b>
3.1	PODJELA METODA KOMPOZICIJE MREŽNIH USLUGA .....	29
3.2	KOMPOZICIJA MREŽNIH USLUGA ZASNOVANA NA OPISU PROCESA.....	30
3.2.1	<i>Poslovni procesi .....</i>	30
3.2.2	<i>Razvoj sustava oznaka za opis procesa .....</i>	31
3.2.3	<i>Teorijska osnova metoda oblikovanja poslovnih procesa .....</i>	32
3.2.4	<i>Kompozicija usluga primjenom orkestracije i koreografije .....</i>	34
3.3	KOMPOZICIJA MREŽNIH USLUGA ZASNOVANA NA SEMANTICI.....	35
3.4	JEZICI KOMPOZICIJE USLUGA .....	37
3.4.1	<i>Svojstva jezika za kompoziciju.....</i>	37
3.4.2	<i>BPEL4WS.....</i>	38
3.4.3	<i>BPML .....</i>	41
3.4.4	<i>WSCl .....</i>	41
3.4.5	<i>WS-CDL .....</i>	42
3.4.6	<i>CL.....</i>	44
<b>4</b>	<b>RAZVOJ PRILAGOĐEN KRAJNJIM KORISNICIMA.....</b>	<b>48</b>
4.1	DEFINICIJA RAZVOJA PRILAGOĐENOG KRAJNJEM KORISNIKU.....	49
4.2	ASPEKTI KRAJNJEM KORISNIKU PRILAGOĐENOG RAZVOJA PROGRAMSKIH SUSTAVA .....	50
4.3	DIMENZIJE SUSTAVA PRILAGOĐENIH KRAJNJEM KORISNIKU .....	52
4.4	PROGRAMIRANJE PRILAGOĐENO KRAJNJIM KORISNICIMA .....	53
4.4.1	<i>Značajke programiranja prilagođenog krajnjim korisnicima .....</i>	53
4.4.2	<i>Primjeri programske jezike namijenjenih krajnjim korisnicima .....</i>	57

<b>5 SSCL KORISNIČKI JEZIK ZA KOMPOZICIJU MREŽNIH USLUGA .....</b>	<b>64</b>
5.1 PROGRAMSKI MODEL ZASNOVAN NA USLUGAMA.....	64
5.1.1 <i>Primjenske mrežne usluge</i> .....	65
5.1.2 <i>Raspodijeljeni programi</i> .....	65
5.1.3 <i>Mrežne usluge za suradnju i natjecanje</i> .....	65
5.1.4 <i>Ostvarenje složene mrežne usluge</i> .....	69
5.2 JEZIK KOMPOZICIJE MREŽNIH USLUGA NAMIJENJEN KRAJNIM KORISNICIMA .....	71
5.3 STRUKTURA SSCL PROGRAMA.....	72
5.4 NAREDBE PROGRAMSKOG JEZIKA SSCL .....	74
5.4.1 <i>Naredbe deklaracije</i> .....	74
5.4.2 <i>Naredba poziva mrežnih usluga</i> .....	75
5.4.3 <i>Naredbe upravljanja tijekom izvođenja programa</i> .....	76
5.4.4 <i>Naredbe za uporabu usluge poštanskog pretinca</i> .....	77
5.4.5 <i>Naredbe za uporabu usluga općeg i binarnog semafora</i> .....	77
5.4.6 <i>Naredbe za uporabu usluge usmjernika događaja</i> .....	78
5.5 IZRAŽAJNOST SSCL KORISNIČKOG JEZIKA.....	79
5.5.1 <i>Osnovni uzorci tijeka izvođenja</i> .....	80
5.5.2 <i>Napredni uzorci tijeka izvođenja</i> .....	87
<b>6 RASPODIJELJENI PREVODITELJ JEZIKA SSCL.....</b>	<b>97</b>
6.1 PROCES PREVOĐENJA.....	97
6.1.1 <i>Analiza izvornog SSCL programa</i> .....	98
6.1.2 <i>Generiranje ciljnog CL programa</i> .....	104
6.2 SUSTAV PREVOĐENJA I IZVOĐENJA RASPODIJELJENIH PROGRAMA .....	107
6.3 ARHITEKTURA RASPODIJELJENOG SUSTAVA PREVOĐENJA .....	108
6.3.1 <i>Arhitektura SSCL prevoditelja</i> .....	108
6.3.2 <i>Arhitektura raspoređivača programa</i> .....	110
6.4 PROGRAMSKO OSTVARENJE RASPODIJELJENOG SUSTAVA PREVOĐENJA .....	111
6.4.1 <i>Programsko ostvarenje modula za prevođenje</i> .....	112
6.4.2 <i>Programsko ostvarenje modula SSCL prevoditelj i modula Rasporedioca programa</i> ..	120
<b>7 ZAKLJUČAK .....</b>	<b>124</b>
<b>8 LITERATURA.....</b>	<b>126</b>
<b>9 ŽIVOTOPIS .....</b>	<b>134</b>
<b>10 SAŽETAK .....</b>	<b>135</b>
<b>11 SUMMARY .....</b>	<b>136</b>
<b>12 KLJUČNE RIJEČI.....</b>	<b>137</b>
<b>DODATAK A.....</b>	<b>138</b>
<b>DODATAK B .....</b>	<b>139</b>

# 1 Uvod

Nova područja računalne znanosti, sveprisutno (engl. *ubiquitous*) i prožimljujuće (engl. *pervasive*) računarstvo, predviđaju širenje primjene računarstva na sve sfere ljudskog života. Posljedica sveprisutne primjene računalnih sustava je veliki broj korisnika različitih znanja. U takvom okruženju samo mali udio korisnika računalnih sustava su profesionalni razvijatelji računalnih sustava, dok većinu čine krajnji korisnici računalnih sustava koji koriste računala i različite programske sustave za obavljanje svakodnevnih poslovnih i osobnih zadataka. Krajnji korisnici su svakodnevno okruženi računalnim uređajima i programskim sustavima koje koriste u različitim fizičkim okruženjima te u različitim socijalnim i tehnološkim okruženjima. Zbog raznolike uporabe računalnih uređaja i programskih sustava, razvijatelji ne mogu predvidjeti sve načine uporabe sustava što često ograničava uporabljivost sustava. Pritom, korisnik nema mogućnosti prilagodbe postojećih računalnih sustava niti mogućnosti oblikovanja novih programskih sustava koji će odgovarati okruženju u kojem ih želi koristiti. Zbog toga se proces oblikovanja i izgradnje programskih sustava pokušava približiti krajnjim korisnicima. Programiranje prilagođeno krajnjim korisnicima (engl. *End User Programming*) je područje računarstva koje proučava tehnike programiranja i programske jezike namijenjene krajnjim korisnicima.

Globalna mrežna Internet je najrašireniji računalni sustav kojeg koristi veliki broj krajnjih korisnika. Razvojem računarstva zasnovanog na uslugama omogućeno je brzo i učinkovito povezivanje raznorodnih programskih sustava dostupnih na Internetu. Osnovni gradivi element računarstva zasnovanog na uslugama je mrežna usluga. Mrežne usluge su samostalne programske cjeline koje omogućuju pristup skupu funkcionalnosti putem normiranih komunikacijskih protokola i programskih sučelja.

Najrašrenija programska paradigma za oblikovanje sustava zasnovanih na uslugama je kompozicija mrežnih usluga. Programske sustave izgrađeni prema načelima kompozicije mrežnih usluga koriste funkcionalnosti postojećih mrežnih usluga te definiraju koordinacijsku logiku kojom se postojeće funkcionalnosti povezuju u jedinstvenu složeniju funkcionalnost. Najrašrenija metoda kompozicije mrežnih usluga je kompozicija zasnovana na opisu procesa. Trenutno je razvijeno više jezika kompozicije mrežnih usluga, od kojih su neki postali opće prihvaćeni i predstavljaju *de facto* norme za opis kompozicije usluga.

Složenost postojećih jezika za opis kompozicije mrežnih usluga osnovna je zapreka primjene kompozicije mrežnih usluga kao programske paradigme namijenjene krajnjim korisnicima. Postojeći jezici, poput jezika BPEL4WS i WS-CDL, zasnovani su na jeziku XML i imaju složena sintaksna i semantička pravila nerazumljiva krajnjem korisniku. Stoga je u sklopu

magistarskog rada razvijen korisnički jezik za kompoziciju mrežnih usluga namijenjen korisnicima bez prethodnog znanja o tehnikama programiranja i programskim jezicima

U magistarskom radu opisan je korisnički jezik za definiciju kompozicije mrežnih usluga, nazvan SSCL (*Simple Service Composition Language*). Jezik SSCL namijenjen je krajnjim korisnicima računalnih sustava koji nemaju znanja o tehnikama programiranja. U radu je oblikovana arhitektura te je programski ostvaren raspodijeljeni sustav prevođenja programa napisanih u jeziku SSCL. Jezik SSCL i raspodijeljeni sustav prevođenja programa napisanih u jeziku SSCL omogućuju jednostavan, brz i krajnjem korisniku prilagođen razvoj programskih sustava zasnovanih na uslugama.

U drugom poglavlju rada opisana su osnovna načela računarstva zasnovanog na uslugama. Navedena su osnovna svojstva mrežnih usluga i postupci izgradnje računalnih sustava zasnovanih na uslugama. Nadalje, opisan je osnovni i prošireni model arhitekture sustava zasnovanih na uslugama, te skup *Web Services* tehnologija za ostvarenje programskih sustava oblikovanih prema načelima arhitekture zasnovane na uslugama.

Treće poglavlje sadrži opis kompozicije mrežnih usluga, najraširenijeg postupka izgradnje programskih sustava zasnovanih na uslugama. Opisana je razredba postupaka kompozicije i programski jezici za opis kompozicije mrežnih usluga.

Četvrto poglavlje sadrži opis područja razvoja prilagođenog krajnjem korisniku. Osnovni cilj istraživanja u području razvoja prilagođenog krajnjem korisniku je približavanje procesa izgradnje programskih sustava krajnjim korisnicima koji nemaju potrebna znanja za izgradnju sustava primjenom programskih jezika opće namjene. Opisane su osnovne značajke i smjernice istraživanja, s naglaskom na opis programskih paradigma i jezika prilagođenih krajnjem korisniku.

U petom poglavlju opisan je jezik SSCL (*Simple Service Composition Language*) za opis kompozicije mrežnih usluga namijenjen krajnjim korisnicima. Jezik SSCL namijenjen je oblikovanju raspodijeljenih sustava prema načelima programskog modela zasnovanog na uslugama. Nakon opisa programskog modela zasnovanog na uslugama, opisana je osnovna struktura programa napisanih u jeziku SSCL te je opisan skup naredbi jezika. Poglavlje završava analizom izražajnosti jezika SSCL.

Raspodijeljeni sustav prevođenja programa napisanih u jeziku SSCL opisan je u šestom poglavlju. Opisana je funkcionalna dekompozicija procesa prevođenja, logička arhitektura sustava te arhitektura programskog ostvarenja. U sedmom poglavlju iznesen je zaključak i smjernice budućeg istraživanja u području prilagodbe kompozicije mrežnih usluga krajnjim korisnicima.

## 2 Računarstvo zasnovano na uslugama

Računarstvo zasnovano na uslugama je nova grana raspodijeljenog računarstva koja značajno mijenja način oblikovanja i izgradnje raspodijeljenih programskih sustava te način njihova plasiranja i uporabe u otvorenim i raznorodnim okolinama. Osnovni element izgradnje programskih sustava je mrežna usluga (engl. *web service*). Mrežna usluga je autonomni, o računalnom okrilju neovisan programski entitet koji se opisuje, objavljuje, otkriva i poziva skupom normiranih jezika, sučelja i komunikacijskih protokola. Nadalje, normiranim jezicima moguće je definirati kompoziciju mrežnih usluga. Kompozicijom usluga stvara se složena usluga koja se poziva, opisuje, objavljuje i otkriva istim normiranim jezicima, sučeljima i protokolima kao i jednostavne usluge. Složene usluge povezuju se postupcima kompozicije s drugim uslugama kako bi se ostvarili složeni programski sustavi. Ovakvim pristupom izgradnji programskih sustava stvara se mreža samostalnih programskih entiteta, ostvarenih kao mrežne usluge, koji međusobno surađuju unutar i izvan organizacijskih granica s ciljem ostvarenja složene funkcionalnosti.

*Web Services* su najrašireniji skup tehnologija za izgradnju računalnih sustava zasnovanih na uslugama. Osnovni skup *Web Services* tehnologija sastoji se od normiranih tehnologija kojima se ostvaruje opisivanje, objava, otkrivanje i pozivanje usluga. *Web Services* tehnologije omogućuju izgradnju normiranih sučelja i primjenu otvorenih protokola koji su osnova razvoja i izvođenja raspodijeljenih poslovnih procesa na sveprisutnoj, globalnoj računalnoj mreži Internet.

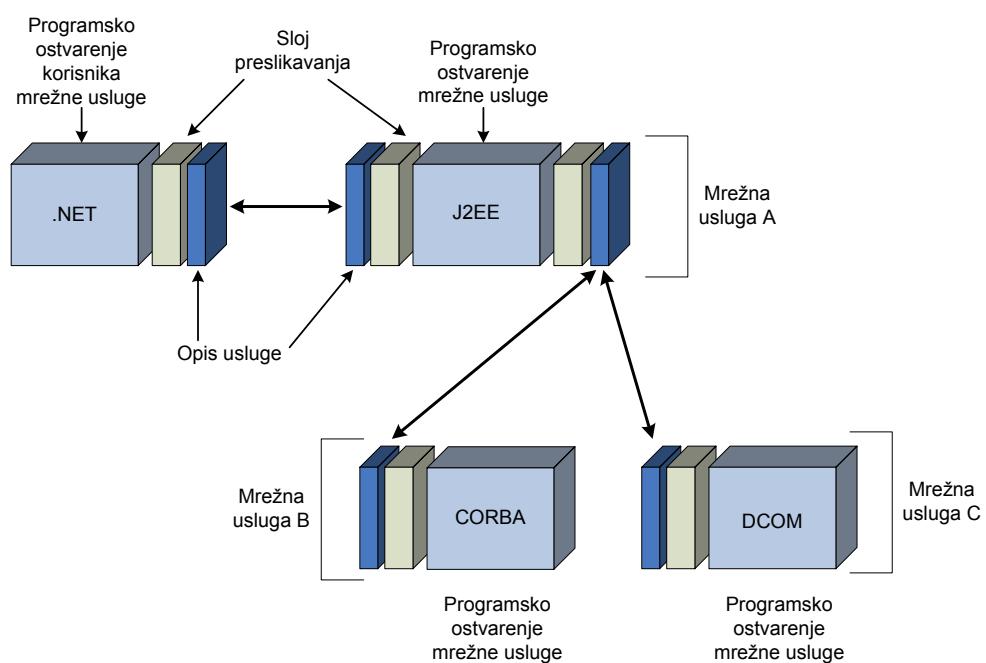
Računarstvo zasnovano na uslugama omogućuje automatsku izgradnju programskih sustava koji ispunjavaju postavljene zadatke koristeći funkcionalnosti postojećih usluga. Tako oblikovani programski sustavi samostalno otkrivaju postojeće usluge i koristiti njihove funkcionalnosti, bez intervencije programera ili korisnika. Ako nije moguće otkriti uslugu koja ostvaruje potrebnu funkcionalnost, pronalaze se usluge koje ostvaruju djelomične funkcionalnosti, te se njihovom kompozicijom riješava postavljeni zadatak.

### 2.1 Mrežne usluge

Mrežne usluge su autonomni (engl. *autonomous*), samoopisujući (engl. *self-describing*) i potpuni (engl. *self-contained*) programski entiteti dostupni putem računalne mreže.

Na slici 2.1 prikazani su osnovni elementi mrežnih usluga i njihovi odnosi. Osnovni elementi usluge su programsko ostvarenje usluge (engl. *service implementation*), opis usluge (engl. *service description*) i sloj preslikavanja (engl. *mapping layer*) između ostvarenja usluge i njenog opisa.

Ostvarenje usluge definira primjensku logiku mrežne usluge i razvijeno je u programskom okruženju najpogodnijem za ostvarenje konkretnе primjenske logike usluge. Pružatelj usluge osim primjenske logike usluge definira i objavljuje opis usluge. Opis usluge je sastavni dio mrežne usluge koji definira način njene uporabe. Opis usluge sadrži informacije o funkcionalnim i nefunkcionalnim svojstvima usluge, te opis programskog sučelja usluge. Informacije dostupne u opisu usluge su sve informacije potrebne korisniku za uporabu usluge. Jedno od najznačajnijih svojstava definicije mrežne usluge je odvojenost ostvarenja usluge od njenog opisa. Opis usluge povezuje se s više ostvarenja usluge ili jedno ostvarenje usluge sadržava više različitih opisa usluge. Opis usluge je odvojen od samog ostvarenja usluge slojem preslikavanja (engl. *transformation layer*). Sloj preslikavanja najčešće je ostvaren primjenom dva programa zastupnika, jednog na korisničkoj strani (engl. *proxy*) i jednog na poslužiteljskoj strani (engl. *stub*). Sloj preslikavanja zadužen je za transformacije struktura podataka korisničkog i poslužiteljskog programa u poruke, čija je struktura definirana opisom usluga, te za razmjenu poruka između korisnika i usluge.

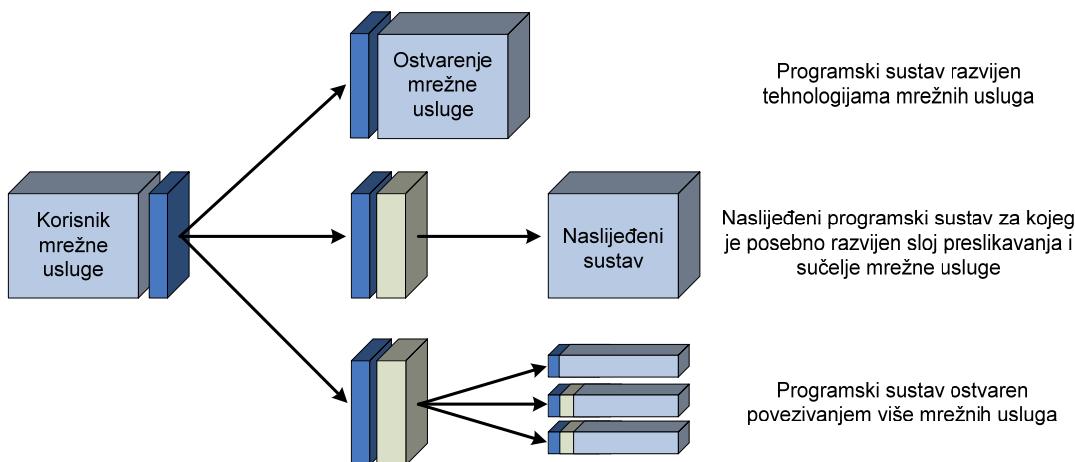


Slika 2.1 Osnovni elementi mrežnih usluga

U primjeru prikazanom na slici 2.1 korisnički program je ostvaren i izvodi se u radnom okruženju Microsoft .Net, a mrežna usluga kojoj se pristupa ostvarena je upotrebom J2EE tehnologije. Kako se međudjelovanje odvija posredstvom sloja preslikavanja i uporabom normiranih sučelja, raznorodnim programskim sustavima korisnika i pružatelja usluge omogućeno je međusobno komuniciranje. Na slici je prikazano da je mrežnoj usluzi omogućeno definiranje više različitih modula preslikavanja i više sučelja, te je prikazano kako

mrežna usluga istovremeno ima ulogu pružatelj i korisnik usluge. U prikazanom primjeru mrežna usluga A sadrži sučelje putem kojim korisnik pristupa ostvarenim funkcionalnostima, ali sadrži i zastupnike kojima usluga A pristupa funkcionalnostima mrežnih usluga B i C.

Jedna od najvećih prednosti mrežnih usluga je mogućnost jednostavne i jednoobrazne uporabe raznorodnih programskih sustava. Primjer na slici 2.2. prikazuje tri raznorodna programska sustava koji su prema korisniku izloženi kao mrežne usluge: programski sustavi oblikovan i izgrađen u skladu s tehnologijama mrežnih usluga, naslijeđeni programski sustav (engl. *legacy systems*) nadograđen normiranim sučeljem mrežne usluge i programski sustav ostvaren kompozicijom mrežnih usluga.

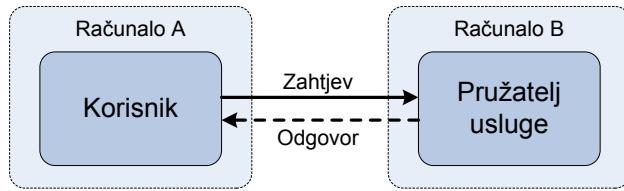


Slika 2.2 Jednoobrazno korištenje raznorodnih programskih sustava

### 2.1.1 Modeli međudjelovanja mrežnih usluga

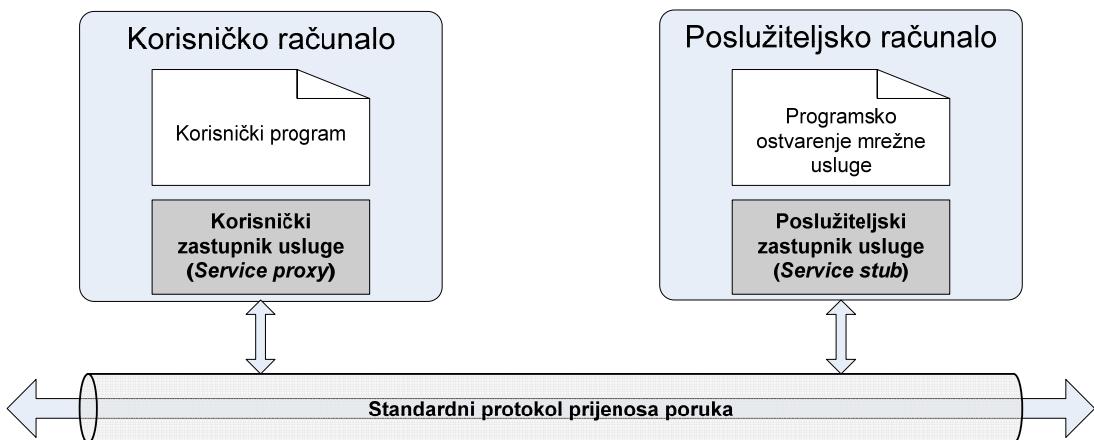
Pristup mrežnim uslugama zasniva se na dva osnovna načela međudjelovanja, poziv udaljenih procedura (engl. *Remote Procedure Call, RPC*) i međudjelovanje porukama (engl. *message centric interaction*).

Međudjelovanje korisnika i mrežne usluge zasnovano na načelu *poziva udaljenih procedura* (engl. *Remote Procedure Call, RPC*) je sinkroni model međudjelovanja. Poziv udaljenih procedura je ostvarenje programskog modela *korisnik-poslužitelj* (engl. *client-server*), kojim se pozivanje programskih funkcionalnosti ostvarenih na udaljenom poslužiteljskom računalu prividno ostvaruje kao poziv funkcionalnosti ostvarenih na lokalnom računalu. U okruženju mrežnih usluga, poslužitelj se naziva pružatelj usluge (engl. *service provider*), a korisnik se naziva korisnik usluge (engl. *service requestor*).



**Slika 2.3 Međudjelovanje korisnika i pružatelja usluge zasnovano na načelu poziva udaljenih procedura**

Slika 2.3 prikazuje međudjelovanje korisnika i pružatelja usluge zasnovano na načelu poziva udaljenih procedura. Korisnik usluge šalje poruku sa zahtjevom pružatelju mrežne usluge. Zahtjev sadrži naziv operacije mrežne usluge koju korisnik želi izvesti i skup parametara operacije. Pružatelj usluge prima zahtjev, poziva operaciju s navedenim skupom parametara i dobiveni rezultat šalje natrag korisniku. Međudjelovanje zasnovano na načelu poziva udaljenih procedura je sinkroni model međudjelovanja u kojem korisnik čeka odgovor na poslani zahtjev.



**Slika 2.4 Korisnički i poslužiteljski zastupnici mrežnih usluga**

Pravid lokalnog korištenja udaljene procedure ostvaruje se primjenom zastupničkih programa na korisničkom i poslužiteljskom računalu (slika 2.4). Zastupnički programi na korisničkom računalu nazivaju se korisnički zastupnici usluga (engl. *service proxy*) i njihova zadaća je oblikovanje i slanje poruke na osnovi poziva usluge ostvarene u lokalnom programu. Korisnički zastupnici usluga pakiraju pokazatelj udaljene procedure i parametre poziva mrežne usluge u normirani oblik poruke definiran opisom usluge i šalju poruku poslužiteljskom zastupniku usluge. Zastupnici na poslužiteljskim računalima (engl. *service stub*) prihvataju poruke, raspakiravaju ih i pretvaraju u strukture podataka korištene u programskom ostvarenju usluge (C#, Java i sl.) Nakon oblikovanja potrebnih struktura podataka poziva se naslovljena metoda programskog ostvarenja usluge. U međudjelovanju zasnovanom na načelu poziva udaljenih procedura jedna primljena poruka sa zahtjevom korisnika preslikava se u poziv jedne metode programskog ostvarenja mrežne usluge.

Rezultat izvođenja odgovarajuće metode predaje se zastupniku na poslužiteljskom računalu. Poslužiteljski zastupnik pakira odgovor u normirani oblik poruke te oblikovanu poruku šalje korisničkom zastupniku usluge. Korisnički zastupnik usluge prima poruku s rezultatom izvođenja udaljene procedure te pretvara poruku u strukture podataka korištene u programskom ostvarenju korisničkog programa.

Za razliku od međudjelovanja zasnovanog na načelu poziva udaljenih procedura, međudjelovanje porukama zasniva se na razmjeni poruka koje sadrže dokumente. Zbog toga se međudjelovanje porukama još naziva i *međudjelovanje razmjenom dokumenata* (engl. *document-centric interaction*). Dokumenti su samostalni i potpuni (engl. *self-contained*) skupovi podataka zapisani u obliku neovisnom o programskom okrilju. Za razliku od poruka razmijenjenih u međudjelovanju zasnovanom na pozivu udaljenih procedura, primljeni dokumenti se ne preslikavaju u poziv jedne metode programskog ostvarenja usluge. Najčešće se primljeni dokument prosljeđuje na obradu različitim komponentama programskog sustava. Rezultat obrade je novi primjerak dokumenta koji se vraća korisniku kao korisna informacija poruke odgovora.

Međudjelovanje razmjenom dokumenata omogućuje sinkrono i asinkrono međudjelovanje korisnika i pružatelja usluge. Asinkrono međudjelovanje razmjenom dokumenata učestalije se koristi od sinkronog jer je znatno pogodnije za raspodijeljene okoline velikih razmjera (engl. *large scale systems*).

### 2.1.2 Svojstva mrežnih usluga

Programski sustavi zasnovani na uslugama oblikuju se u skladu s osnovnim svojstvima mrežnih usluga. Osnovna svojstva koje je potrebno primijeniti tijekom oblikovanja, ostvarenja i upravljanja mrežnim uslugama su slaba povezanost, dobro definirani ugovori usluga, prilagođenost opisa usluge korisnicima i usklađenost sa normama.

#### *Slaba povezanost*

Značajno svojstvo mrežnih usluga je slaba povezanost (engl. *loose coupling*). Slaba povezanost je rašireni pojam koji se u praksi odnosi na različite elemente usluge, njenog ostvarenja i njene uporabe.

*Povezanost sučelja* (engl. *interface coupling*) odnosi se na razinu povezanosti korisnika i pružatelja usluge. Povezanost sučelja određuje u kojoj mjeri pružatelj usluge nameće ograničenja i ovisnost korisniku usluge. Što je ovisnost korisnika o pružatelju usluge manja, to je povezanost slabija. U idealnom slučaju, korisnik usluge ima mogućnost korištenja usluge samo na osnovi objavljenog ugovora (engl. *published service contract*) i dogovora na

razini usluge (engl. *service level agreement*), te korisniku nisu potrebne informacije o ostvarenju usluge.

*Povezanost tehnologije* (engl. *technology coupling*) određuje u kojoj mjeri mrežna usluga ovisi o određenoj tehnologiji, proizvodu ili razvojnom okrilju. Na primjer, ako neka organizacija koristi programsko okrilje .NET za ostvarenje svih svojih usluga i ako ista organizacija zahtijeva da svi korisnici usluga koriste generator zastupnika mrežnih usluga *wsdl.exe*, alata koji je dio programskog okrilja .Net, onda su mrežne usluge čvrsto povezane s .Net tehnologijom. Čvrsta povezanost znatno ograničava skup korisničkih programa kojima je omogućen pristup mrežnoj usluzi. U najgorem slučaju, čvrsto povezana ostvarenja ograničavaju pružatelje i korisnike usluga na korištenje programskog okrilja koje je vlasništvo samo jednog proizvođača što rezultira ovisnošću programskog rješenja o proizvođaču okrilja. Postizanje slabe tehnološke povezanosti usluga jedan je od glavnih razloga korištenja *Web Services* tehnologija za ostvarenje i pristup uslugama. *Web Services* tehnologije su opće prihvaćeni skup tehnologija za ostvarenje mrežnih usluga zasnovan na postojećim Internet normama i protokolima.

*Povezanost procesa* (engl. *process coupling*) definira mjeru povezanosti složenog primjenskog procesa i mrežne usluge koja ostvaruje dio funkcionalnosti primjenskog procesa. Mrežna usluga i proces su slabo povezani ako usluga ima dobro definiranu funkcionalnost koja je neovisna o samom primjenskom procesu. Slabo povezane usluge moguće je ponovno iskoristiti u ostvarenju drugih primjenskih procesa. S druge strane, kako povezana usluga izgrađena je kao dio primjenskog procesa i njena funkcionalnost nema uporabe izvan okruženja tog procesa. Čvrsto povezane usluge, za razliku od slabo povezanih, nemaju svojstvo ponovne iskoristivosti u drugim primjenskim procesima.

### *Dobro definirani ugovori mrežnih usluga*

Bilo koja mrežna usluga treba imati dobro definiran opis usluge. Opis usluge često se naziva *ugovor usluge* (engl. *service contract*). Ugovorom su definirana funkcionalna i nefunkcionalna svojstva usluge, opis programskog sučelja usluge i način pozivanja usluge. Ugovorom su jasno odvojena sučelja usluge od programskog ostvarenja usluge. Ugovor usluge definira se uporabom normiranog skupa jezika. S ciljem omogućavanja međudjelovanja između raznorodnih računalnih sustava korisnika i pružatelja usluga, normirani jezici za opis ugovora usluge definirani su neovisno o računalnom okrilju.

Ugovor usluge potrebno je definirati na osnovi znanja području primjene usluge, a ne na osnovi ostvarenja usluge. Dobro definirani ugovor usluge izuzetno je značajan u poslovnoj primjeni gdje veliki broj korisnika pristupa uslugama. Promjena ugovora usluge uzrokuje promjene svih programa korisnika mrežnih usluga. Nasuprot tome, promjena ostvarenja

usluge uz zadržavanje postojećih ugovora usluge ne uzrokuje promjene u korisničkim programima, jer ugovor usluge sakriva promjene u programskom ostvarenju.

#### *Prilagođenost opisa usluge korisnicima*

Usluge i ugovori usluga definiraju se na razini apstrakcije koja je razumljiva korisniku usluge. Prikladna razina apstrakcije ugovora usluge obuhvaća bit primjenske logike usluge, ali ne sadrži informacije koje ograničavaju način uporabe usluge ili način programskog ostvarenja usluge. U ugovoru se ne navode tehnički detalji programskog ostvarenja usluge, jer su ti podaci za korisnika usluge nebitni.

Apstraktna sučelja su sučelja koja sadrže samo informacije o primjenskoj logici usluge. Tako oblikovana sučelja omogućuju zamjenu postojećeg pružatelja usluge s novim pružateljem bez utjecaja na korisnike usluge. Primjenom apstraktnih sučelja omogućena je slaba povezanost usluga.

#### *Zasnovanost na otvorenim, općeprihvaćenim normama*

Usluge se zasnivaju na otvorenim normama. Primjenom otvorenih normi izbjegava se uporaba vlasničkih protokola i tehnologija koje uzrokuju ovisnost korisnika o određenim pružateljima usluga. Na taj način se korisniku omogućuje jednostavna promjena pružatelja usluge, ali i pružateljima usluga omogućuje se ponuda svojih usluge velikom broju korisnika koji koriste raznorodne računalne sustave. Uporaba otvorenih normi ne odnosi se samo na ostvarenje elementarnih mrežnih usluga, već uključuje i oblikovanje poslovnih procesa zasnovanih na uslugama u skladu sa normama više razine, poput normi kompozicije mrežnih usluga.

## **2.2 Arhitektura zasnovana na uslugama**

Arhitektura zasnovana na uslugama (engl. *Service Oriented Architecture, SOA*) definira način oblikovanja raspodijeljenih programskih sustava koji koriste koncepte mrežnih usluga. U programskim sustavima oblikovanim u skladu s arhitekturom zasnovanom na uslugama sve se ostvarene funkcionalnosti izlažu kao mrežne usluge. Funkcionalnosti mrežnih usluga koriste se za obavljanje korisničkih zadataka, ali i kao jedini elementi izgradnje novih, složenih funkcionalnosti.

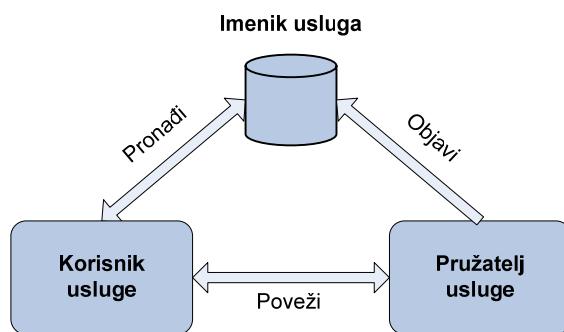
Računalna arhitektura zasnovana na uslugama opisuje osnovni i prošireni model arhitekture programskih sustava zasnovanih na uslugama. Osnovni model arhitekture definira programske jedinice i model međudjelovanja kojim je omogućeno ostvarenje raspodijeljenih sustava neovisnih o računalnom okružju, tehnologijama ostvarenja dijelova sustava i korištenim mrežnim komunikacijskim protokolima. Proširena arhitektura definira nadogradnju

osnovne arhitekture sa slojem kompozicije mrežnih usluga i slojem upravljanja sustavima zasnovanim na uslugama.

### 2.2.1 Osnovni model arhitekture zasnovane na uslugama

Osnovni model arhitekture zasnovane na uslugama definira osnovne elemente sustava zasnovanih na uslugama i njihovo međudjelovanje, kako je prikazano na slici 2.5. Osnovni elementi programskih sustava zasnovanih na uslugama su *pružatelj usluge*, *korisnik usluge* i *imenik usluga*. Osnovne operacije kojime se opisuje međudjelovanje između navedenih elemenata su *objavi*, *pronađi* i *poveži*.

Pružatelj usluge definira sučelje mrežne usluge putem kojeg se pristupa funkcionalnostima primjenskog programa koji se izvodi na računalu pružatelja usluge. Pružatelj usluge opisuje mrežnu uslugu normiranim jezicima za opis mrežne usluge. Opis usluge *objavljuje* se u javno dostupni imenik usluga. Korisnik *pretražuje* imenik mrežnih usluga tražeći opise usluga koje ostvaruje željenu funkcionalnost. Korisnik dohvata opis željene usluge koji sadrži sve informacije potrebne za *povezivanje* s uslugom, odnosno za uporabu usluge. Nakon povezivanja, korisnik započinje uporabu usluge koja se sastoji od niza razmjena poruka između korisnika i pružatelja usluge. Korisnik, u skladu s dohvaćenim opisom usluge, oblikuje poruke sa zahtjevom za određenom funkcionalnosti i šalje ih usluzi. Usluga prima poruke sa zahtjevima korisnika, izvodi tražene funkcionalnosti te korisniku šalje poruku s odgovorom.



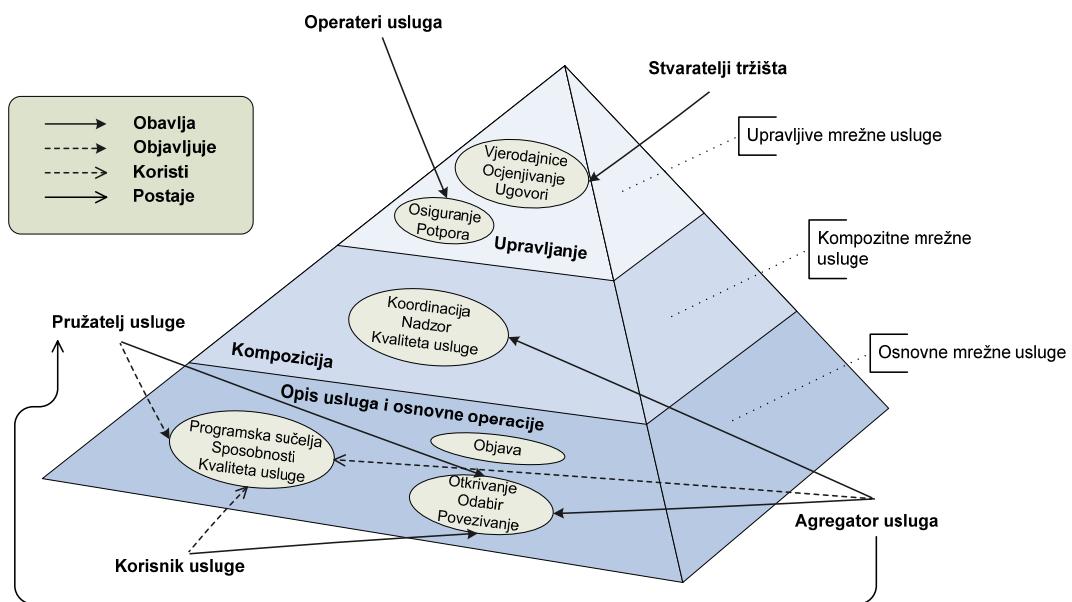
Slika 2.5 Osnovni model arhitekture zasnovane na uslugama

Opisana arhitektura omogućuje prestrukturiranje postojećih samostalnih i nepodudarnih programskih sustava u skup podudarnih mrežnih usluga, dostupnih putem normiranih sučelja i komunikacijskih protokola. Jednom kada se postojeći programski sustavi izlože kao mrežne usluge, svi budući sustavi kao gradivne elemente koriste postojeće mrežne usluge. Arhitektura zasnovana na uslugama je posebno prikladna i primjenjiva u okruženjima u kojima je potrebno omogućiti komunikaciju između programskih sustava izgrađenih različitim tehnologijama, koji se izvode na različitim računalnim okriljima i operacijskim sustavima.

Kako mrežne usluge izlažu funkcionalnosti postojećih programskih sustava na višoj razini apstrakcije, povezivanje usluga ne zahtijeva napredne programske vještine i omogućuje brz razvoj novih programskih sustava.

### 2.2.2 Prošireni model arhitekture zasnovane na uslugama

Osnovni model arhitekture zasnovane na uslugama definira mehanizme izgradnje, opisa, objave i korištenja mrežnih usluga. Osnovna arhitektura omogućuje međudjelovanje usluga ostvarenih na raznorodnim računalnim okriljima, povezanih raznorodnim komunikacijskim protokolima i uporabom raznorodnih programskih jezika. Za potrebe oblikovanja složenih programskih sustava zasnovanih na uslugama, odnosno oblikovanja složenih primjenskih procesa uporabom mrežnih usluga potrebno je osnovnu arhitekturu proširiti mogućnostima oblikovanja složenih mrežnih usluga. Složene mrežne usluge grade se različitim postupcima kompozicije mrežnih usluga. Nadalje, potrebno je razviti mehanizme nadzora i upravljanja izvođenjem složenih mrežnih usluga. Prošireni model arhitekture zasnovane na uslugama prikazan je piramidom na slici 2.6. Najniži sloj sadrži osnovni model arhitekture zasnovane na uslugama, dok dva viša sloja definiraju dodatnu potporu izgradnji složenih usluga postupcima kompozicije (engl. *service composition*) i upravljanju uslugama (engl. *service management*).



Slika 2.6 Prošireni model arhitekture zasnovane na uslugama

#### Sloj kompozicije

Sloj kompozicije (engl. *composition layer*) usluga definira elemente i funkcionalnosti koje se koriste za izgradnju složenih usluga postupcima kompozicije postojećih usluga. Rezultirajuća

složena usluga sadrži normirano sučelje kao i svaka druga usluga ostvarena na načelima osnovnog modela. Tako ostvarena složena usluga poziva se iz korisničkih programa ili se koristi u izgradnji novih složenih usluga.

Povezivači usluga (engl. *service aggregators*) su središnji elementi sloja kompozicije usluga. Povezivači usluga prema korisniku definiraju istu funkcionalnost kao i pružatelji usluga definirani osnovnim modelom arhitekture zasnovane na uslugama. Za razliku od pružatelja usluge, povezivač usluge ostvaruje funkcionalnosti usluge povezivanjem funkcionalnosti postojećih mrežnih usluga. Osnovne funkcije povezivača usluga su koordinacija usluga, prilagođavanje postojećih usluga, osiguravanje kakvoće složene usluge te provedba pravila korištenja složene usluge.

Koordinacija usluga definira tijek izvođenja i tok podataka složene usluge. Tijek izvođenja složene usluge određuje redoslijed pozivanja operacija mrežnih usluga kojima je ostvarena funkcionalnost složene usluge. Tokom podataka povezuju se ulazni parametri operacija složene usluge s ulaznim parametrima osnovnih usluga, definira se tok podataka između osnovnih usluga, te se povezuju izlazni parametri operacija osnovnih usluga s izlaznim parametrima operacije složene usluge.

Funkcija prilagođavanja osigurava cjelokupnost složene usluge prilagođavanjem tipova ulaznih i izlaznih parametara operacija složene usluge tipovima parametara operacija osnovnih usluga. Povezivači usluga po potrebi definiraju ograničenja vrijednosti izlaznih parametara osnovnih usluga kako bi se osigurala ispravnost logike složene usluge.

Povezivač usluga je zadužen za definiranje i osiguranje kakvoće usluge (engl. *Quality of Service, QoS*). Potrebno je osigurati definirane parametre kakvoće složene usluge na osnovi definiranih parametara kakvoće osnovnih usluga. Primjeri definiranja kakvoće složene usluge su definiranje ukupne cijene složene usluge na osnovi cijena osnovnih usluga, osiguranje sigurnosti, privatnosti, transakcijske cjelokupnosti, pouzdanosti i sličnih svojstava složene usluge koja ovise o svojstvima osnovnih usluga.

Povezivač usluga osigurava provedbu pravila korištenja usluge (engl. *policy enforcement*). Sposobnosti usluga i uvjeti njihova korištenja zadaju se pravilima. Na primjer, informacija da usluga podržava sigurnosnu normu nije dovoljna kako bi se osigurala uspješna kompozicija te usluge s ostalim uslugama. Korisnik mora saznati da li usluga nužno zahtijeva korištenje sigurnosne norme, te koje vrste sigurnosnih značaka usluga može obrađivati. Nadalje, korisnik mora saznati da li usluga zahtijeva digitalno potpisivanje poruka koje prima, enkripciju poruka, koji algoritam enkripcije se zahtijeva te druge podatke o pravilima korištenja usluge. Kompozicijom usluga bez poznavanja navedenih informacija postiže se pogrešni rezultat.

### Sloj upravljanja

Kako bi se omogućilo korištenje mrežnih usluga u izgradnji poslovnih programskih sustava ključnih za ispravno poslovanje organizacije, potrebno je osigurati mogućnost nadzora i upravljanja radnim svojstvima (engl. *performance management*) mrežnih usluga. S ciljem oblikovanja usluga nad kojima je moguće provoditi nadzor i upravljanje, proširena arhitektura zasnovana na uslugama definira *upravljive usluge*. Upravljive mrežne usluge definirane su slojem upravljanja (engl. *management layer*) koji se nalazi na vrhu piramide proširene arhitekture.

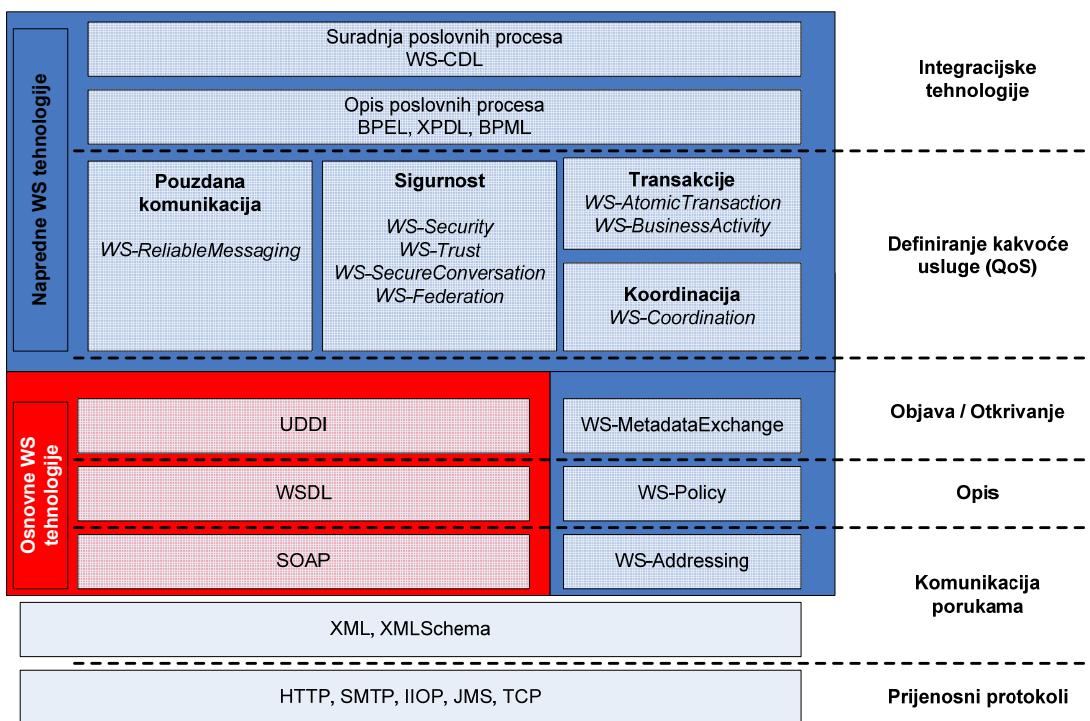
Primjeri funkcionalnosti koje su definirane slojem upravljanja su pružanje uvida u detaljnu statistiku radnih svojstava mrežnih usluga, praćenje tijeka izvođenja složenih usluga, definiranje nadzornih točaka u tijeku izvođenja složene usluge i mogućnost slanja izvještaja o izvođenju složenih usluga korisnicima. Jedinke sloja upravljanja koje su zadužene za ostvarenje navedenih funkcionalnosti nadzora i upravljanja su *operatori mrežnih usluga* (engl. *service operators*). *Operatori mrežnih usluga* su pružatelji mrežnih usluga ili povezivači mrežnih usluga ovisno o strukturi programskog sustava.

Nadalje, zadaća sloja upravljanja je pružanje potpore stvaranju *otvorenog tržišta usluga* (engl. *open service marketplace*). Namjena otvorenih tržišta usluga je potpora uspostavi poslova između kupaca (korisnika usluga) i trgovaca (pružatelja usluga) elektroničkim putem. Trgovac obavlja traženu uslugu na zahtjev kupca. Ako trgovcu nije omogućeno da korisniku izravno pruži traženu uslugu, onda pokušava korisniku pružiti traženu uslugu na osnovu drugih usluga dostupnih na tržištu. Usluga pružena korisniku povezivanjem više drugih usluga dostupnih na tržištu naziva se usluga s dodatnom vrijednošću (engl. *value-added service*). Vrijednost tako pružene usluge je za korisnika veća od zbroja vrijednosti svih pojedinačnih usluga na osnovi kojih je izgrađena.

Otvorena tržišta usluga pogodna su za oblikovanje automatiziranog lanca snabdijevanja (engl. *supply chain*), jer sudionicima pružaju jedinstveni način predstavljanja proizvoda i usluga, osiguravaju razumijevanje sudionika definiranjem normirane terminologije i omogućuju njihovo međudjelovanjem definiranjem normiranih poslovnih protokola. Nadalje, tržište usluga sadržava skup usluga potpore poslovnim procesima, kao što su usluge za izvođenje finansijskih transakcija, izdavanje vjerodajnica pružateljima usluge (engl. *service certification*), usluga ocjenjivanja poslovnih usluga (engl. *rating services*), te usluge pregovaranja i sklapanja ugovora o korištenju usluge (engl. *service level agreements*). Tržišta usluga stvaraju i nadziru organizacije koje se nazivaju *stvaratelji tržišta* (engl. *market-maker*). Stvaratelji tržišta povezuju proizvođače, odnosno pružatelje usluga s kupcima, preuzimaju odgovornost za administrativne poslove, te jamče otvorenost tržišta.

## 2.3 Web Services tehnologije ostvarenja sustava zasnovanih na uslugama

Najraširenije tehnologije ostvarenja sustava zasnovanih na uslugama su *Web Services* tehnologije. *Web Services* tehnologije specificiraju skup normiranih tehnologija potrebnih za ostvarenje programskih sustava oblikovanih prema načelu osnovnog i proširenog modela arhitekture zasnovane na uslugama. *Web Services* tehnologije moguće je hijerarhijski organizirati i predstaviti stogom tehnologija prikazanim na slici 2.7. Prikazana organizacija je slojevita, a svaki sloj ima definiranu namjenu prikazanu na desnoj strani slike. Niži slojevi stoga *Web Services* tehnologija sadrže osnovne *Web Services* tehnologije potrebne za ostvarenje programskih sustava oblikovanih prema načelima osnovnog modela arhitekture zasnovane na uslugama. Viši dijelovi stoga sadrže napredne *Web Services* tehnologije nužne za ostvarenje programskih sustava oblikovanih prema načelima proširenog modela arhitekture zasnovane na uslugama. Na slici 2.7. prikazan je samo dio *Web Services* tehnologija koje su najznačajnije i najraširenije.



Slika 2.7 Hijerarhijska organizacija Web Services tehnologija

### 2.3.1 Osnovne *Web Services* tehnologije

Osnovne *Web Services* tehnologije definiraju normirane protokole i jezike ostvarenja osnovnog modela arhitekture zasnovane na uslugama. Osnovne *Web Services* tehnologije definiraju normirani protokol komunikacije zasnovane na porukama (SOAP), normirani jezik

opisa mrežne usluge (WSDL), te normiranu specifikaciju arhitekture sustava imenika usluga, protokola objave usluga u imenicima i protokola otkrivanja objavljenih usluga (UDDI).

Iako *Web Services* skup tehnologija nije vezan za jedan mrežni transportni protokol, mrežne usluge najčešće koriste sveprisutne Internet protokole i infrastrukturu kako bi se osigurala mogućnost njihova korištenja u raznorodnim računalnim sustavima. Postojeće mrežne usluge stoga najčešće koriste HTTP (*HyperText Transport Protokol*) kao mrežni transportni protokol. HTTP je najrašireniji transportni protokol primjenske razine i tvori osnovu WWW sustava.

Sve *Web Services* tehnologije zasnovane su na XML jeziku, čime je osigurana neovisnost WS tehnologija o računalnim okriljima i mrežnim protokolima. XML (*Extensible Markup Language*) [15] je općeprihvaćen tekstualni format zapisa podataka koji je neovisan o računalnom okrilju. Struktura XML dokumenata koji se koriste za zapis podataka definira se XML Schema [16] jezikom. XML i XML Schema jezici su osnova izgradnje većine komunikacijskih protokola primjenske razine namjenjenih povezivanju raznorodnih računalnih sustava.

## SOAP

SOAP protokol [18] je jednostavan, na XML-u zasnovan, komunikacijski protokol razmjene poruka između računalnih sustava neovisan o računalnom okrilju, operacijskom sustavu i programskom jeziku. Zadaća SOAP protokola je savladavanje prepreka raznorodnosti koje razdvajaju raspodijeljena računalna okrilja. SOAP ne definira novu tehnologiju, već propisuje korištenje postojećih Internet tehnologija s ciljem normiranja komunikacije između raznorodnih računalnih sustava.

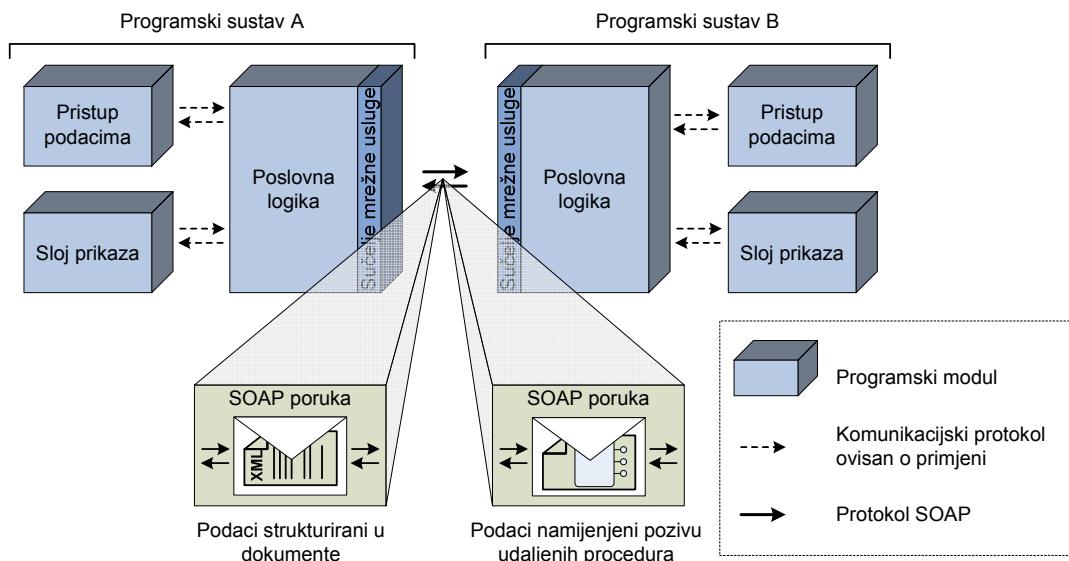
Osnovna jedinica komunikacije SOAP protokola je SOAP poruka. SOAP poruka je XML dokument strogo definirane strukture. Struktura SOAP poruke definirana je *SOAP omotnicom* (engl. *SOAP Envelope*) koja sadrži *tijelo poruke* (engl. *SOAP Body*) i opcionalno *zaglavljeporuke* (engl. *SOAP header*). Struktura SOAP poruke prikazana je na slikci 2.8. Tijelo poruke sadrži korisnu informaciju koja se prenosi od pošiljatelja prema primatelju. Unutarnja struktura tijela poruke nije definirana i sadrži proizvoljne informacije. Zaglavljeporuke sadrži informacije koje se koriste za prijenos i zaštitu SOAP poruke. Informacijama navedenim u zaglavljeporuce moguće je definirati korisnička proširenja osnovnog SOAP protokola. Na primjer, u zaglavljeporuce moguće je zadati informacije potrebne za ostvarenje autentikacije, digitalnog potpisivanja poruka i usmjeravanja poruka.



Slika 2.8 Struktura SOAP poruke

Iako je omogućeno da SOAP protokol koristi i druge prijenosne protokole poput TCP, SMTP i FTP protokola, najčešće korišteni prijenosni protokol je HTTP. SOAP poruke se prenose kao dijelovi HTTP zahtjeva i odgovora, što omogućuje jednostavnu komunikaciju SOAP protokolom putem svih mreža koje su zasnovane na HTTP prometu.

SOAP protokol omogućuje dva načina međudjelovanja mrežnih usluga koji su prikazani na slici 2.9, međudjelovanje zasnovano na načelu poziva udaljenih procedura i međudjelovanje razmjenom dokumenata. Ovisno o vrsti međudjelovanja, podaci koje se prenose SOAP porukama su podaci namijenjeni pozivu udaljenih procedura ili podaci strukturirani u dokumente (engl. *document-centric data*).



Slika 2.9 Vrste podataka koji se prenose SOAP porukama

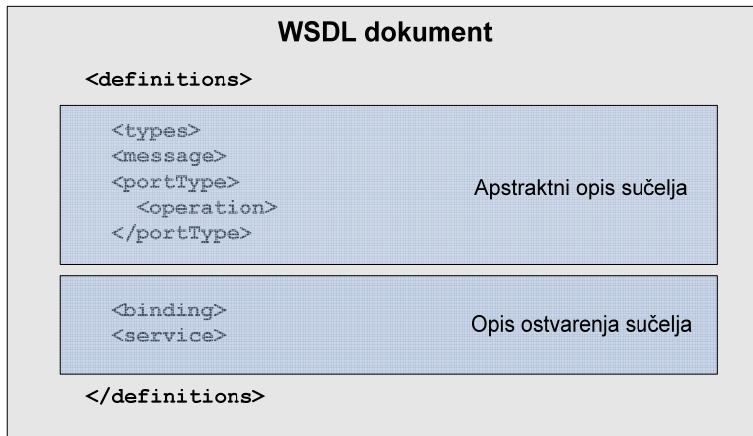
## WSDL

WSDL (*Web Service Description Language*) [19] je jezik za ostvarenje strojno-razumljivog opisa mrežnih usluga. WSDL opis sadrži informacije potrebne korisniku za povezivanje s mrežnom uslugom. Ugovor o uporabi usluge (engl. *service contract*) definira valjane uzorke razmjene poruka između korisnika i pružatelja usluge. Ugovor o uporabi usluge sastoji se od jednog ili više ugovora operacija (engl. *operation contract*). Svaki ugovor operacije definira strukturu poruke zahtjeva za izvođenjem operacije i strukturu poruke odgovora kojom se šalje rezultat izvođenja operacije.

Jezik WSDL zasnovan je na XML-u što omogućuje opis sučelja usluge neovisan o računalnom okrilju i programskom jeziku. Normirani i o računalnom okrilju neovisan opis mrežnih usluga osnova je slabe povezanosti programskih sustava zasnovanih na uslugama. Jezikom WSDL definirana je minimalna količina informacija potrebna za međusobnu komunikaciju korisnika i pružatelja usluge. Korisnik koji ima pristup WSDL dokumentu i komunicira putem SOAP protokola, poziva mrežnu uslugu bez informacija o programskom jeziku ostvarenja mrežne usluge, operacijskom sustavu ili računalnom okrilju na kojoj se izvodi mrežna usluga.

Struktura WSDL dokumenta prikazana je na slici 2.10. WSDL dokument sastoji se od apstraktnog opisa sučelja i opisa ostvarenja sučelja. Apstraktни opis sučelja sadrži definicije tipova podataka, poruka i operacija usluge. Tipovi podataka koji se koriste u ostatku WSDL dokumenta definirani su jezikom XMLSchema i nalaze se unutar XML elementa `<types>`. Strukture poruka koje se koriste u komunikaciji definirane su na osnovi imena XMLSchema tipova podataka. Poruke su definirane nizom XML elemenata `<message>`. Skup operacija mrežne usluge definira se unutar XML elementa `<portType>`. Element `<portType>` sastoji se od niza elemenata `<operation>` kojim se definiraju pojedine operacije mrežne usluge. Operacije usluge definiraju se specificiranjem ulaznih i izlaznih poruka. Ulazne i izlazne poruke definiraju se imenima poruka definiranih u elementu `<messages>`.

Opis ostvarenja mrežne usluge sadrži informacije o mrežnoj adresi na kojoj je dostupno sučelje, o prijenosnim mrežnim protokolima koji se koriste u komunikaciji s uslugom, te o načinu prijenosa apstraktnih tipova podataka prijenosnim mrežnim protokolom. Za razliku od apstraktnog dijela opisa koji definira programsko sučelje za poziv operacija usluge iz korisničkog programa, opis ostvarenja usluge definira stvarnu mrežnu krajnju točku putem koje korisnici pristupaju funkcionalnostima mrežne usluge.



Slika 2.10 Primjena WSDL opisa sučelja usluga

Informacije sadržane u apstraktnom opisu sučelja i informacije o ostvarenju sučelja usluge čine dovoljnu količinu informacija koja je potrebna korisniku za pristup i komunikaciju s mrežnom uslugom, te korištenje funkcionalnosti usluga. Zahvaljujući potpunosti informacija u WSDL dokumentima razvijeno je više programskih alata koji na osnovi WSDL dokumenata generiraju korisničke zastupnike mrežnih usluga koji omogućuju jednostavan pristup operacijama mrežne usluge iz korisničkog programa.

### UDDI

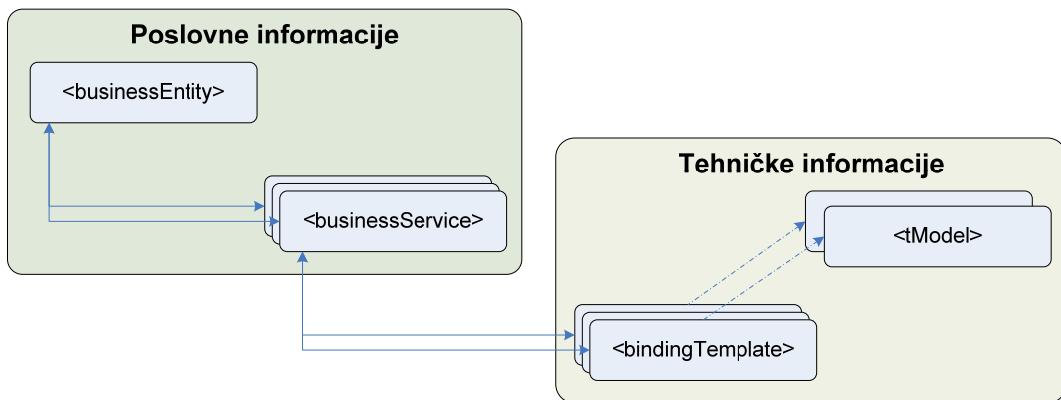
UDDI (*Universal Description, Discovery and Integration*) [20] je norma koji definira strukturu imenika s opisima usluga, protokol objave opisa i protokol otkrivanja usluga. Nakon oblikovanja i programskog ostvarenja mrežne usluge, potrebno je uslugu objaviti u UDDI imeniku kako bi je potencijalni korisnici mogli otkriti i koristiti. Proces otkrivanja mrežnih usluga sastoji se od pronalaženja i analiziranja WSDL opisa mrežnih usluga koji sadrže sve informacije potrebne za korištenje mrežne usluge.

UDDI imenici su globalni imenici neovisni o računalnom okrilju koji korisnicima mrežnih usluga pružaju funkcionalnosti otkrivanja informacija o organizacijama koje pružaju mrežne usluge, pronalaženja opisa mrežnih usluga koje organizacije pružaju i pronalaženja tehničkih informacija o sučeljima mrežnih usluga.

Osnovno svojstvo UDDI imenika je uporaba taksonomija kojima su ostvarene razredbe područja poslovanja i poslovnih usluga. Na primjer, pružatelj usluge koristeći taksonomiju izražava usklađenost usluge s nekom normom iz područja primjene usluge ili definira pružanje usluge samo na određenom geografskom području. Primjena taksonomije korisnicima olakšava pronalazak usluga koje odgovaraju njihovim potrebama.

UDDI imenici su logički podijeljeni na *bijele stranice* (engl. *white pages*), *žute stranice* (engl. *yellow pages*) i *zelene stranice* (engl. *green pages*). *Bijele stranice* sadrže imena poslovnih

organizacija, njihove adrese i ostale informacije o poslovnim organizacijama. Žute stranice sadrže informacije o području djelatnosti poslovnih organizacija i usluga koje pružaju, u skladu sa normiranim industrijskim taksonomijama. Zelene stranice sadrže tehničke informacije o uslugama koje organizacije javno izlažu prema korisnicima. Informacije o uslugama sadrže kazaljke na specifikacije mrežnih usluga, odnosno na WSDL opise sučelja usluga.



Slika 2.11 Struktura podataka UDDI zapisa

Osnovni element arhitekture UDDI imenika je *UDDI poslovni zapis* (engl. *UDDI business registration*). *UDDI poslovni zapis* je XML dokument kojim se opisuje poslovna organizacija i mrežne usluge koje organizacija izlaže. Hjерархија типова података и структура XML документа koji sadrži информације о пословном предмету који се појављују на *bijelim*, *žutим* и *зеленим* страницама UDDI именика приказана је на слици 2.11.

Osnovni XML element na osnovi kojeg se objavljuju i otkrivaju usluge je *<businessEntity>* element koji predstavlja stvarni poslovni subjekt koji pruža određene usluge. Informacije sadržane u ovom elementu pripadaju *bijelim stranicama* UDDI imenika i omogućuju jednostavnu pretragu UDDI imenika prema nazivu poslovne organizacije ili njenoj adresi. Sve usluge koje pruža poslovna organizacija opisane su XML elementima *<businessService>*. Informacije sadržane u ovim elementima логички pripadaju "žutим stranicама" UDDI imenika. Kako poslovne organizacije pružaju veći broj usluga, *<businessEntity>* element sadržava više *<businessService>* elemenata.

Osim poslovnih информација које се користе током проналaska жељене usluge, кориснику су потребне и техничке информације како би му било омогућено приступање услузи. Информације о адреси на којој је доступно остварење мрежне usluge и друге техничке особине остварења мрежне usluge садржане су у *<bindingTemplate>* елементу. Један логички опис usluge садржи више остварења на различитим мрежним адресама. Због тога *<businessService>* element sadržava više *<bindingTemplate>* elemenata. Osim информација о остварењу usluge потребне су информације о суочљима usluge, формату порука које се размјењују с uslугом i

razini sigurnosti. Navedene informacije sadržane su u elementu *<tModel>* (*Technology Model*). Tehničke informacije spremljene u UDDI imenik sljede model razdvajanja apstraktnih podataka o usluzi primjenom *<tModel>* elementa i informacija o stvarnom ostvarenju usluge primjenom *<bindingTemplate>* elementa.

### 2.3.2 Napredne *Web Services* tehnologije

Napredne *Web Services* tehnologije omogućuju ostvarenje programskih sustava oblikovanih prema načelima proširenog modela arhitekture zasnovane na uslugama. Napredne *Web Services* tehnologije dijele se na tehnologije koje proširuju funkcionalnosti osnovnih slojeva, tehnologije sloja definiranja kakvoće usluga i tehnologije objedinjavanja mrežnih usluga u složene sustave zasnovane na uslugama. Primjeri proširenja osnovnih slojeva su norme *WS-Addressing*, *WS-Policy* i *WS-MetaDataExchange*.

*WS-Addressing* [21] je proširenje SOAP komunikacijskog protokola koje definira mehanizam adresiranja mrežnih usluga neovisan o prijenosnom mrežnom protokolu. Korištenjem adresnih mehanizama mrežnog prijenosnog protokola, npr. HTTP protokola, moguće je adresirati mrežnu uslugu primjenom adrese mrežne usluge, ali nije moguće adresirati pojedine elemente mrežne usluge. *WS-Addressing* norma omogućuje preciznije adresiranje elemenata mrežnih usluga definiranjem strukture XML elementa *<EndpointReference>*. *<EndpointReference>* element osim obavezognog elementa *<Address>*, kojim je definirana mrežna adresa usluge, sadrži i niz neobaveznih elemenata kojima je moguće precizno adresirati element mrežne usluge.

*WS-Policy* [22] norma definira napredne mehanizme opisa mrežnih usluga. WSDL opis mrežne usluge ne sadrži dovoljno informacija potrebnih za poziv naprednih mrežnih usluga. Na primjer, WSDL jezikom nije moguće izraziti da li usluga zahtijeva od korisnika uporabu sigurnosnih mehanizama ili da li usluga ostvaruje transakcijske operacije. *WS-Policy* norma omogućuje definiranje funkcionalnosti koje se pružaju korisniku, ali i funkcionalnosti koje se očekuju od korisnika usluge. *Web Services* norme sigurnosti, pouzdane razmjene poruka i transakcija zahtijevaju definiranje konkretnih pravila korištenja usluge primjenom *WS-Policy* norme.

*WS-MetadataExchange* [23] norma definira protokol otkrivanja mrežnih usluga sukladan UDDI normi otkrivanja usluga. Dok UDDI norma definira posebne imenike mrežnih usluga, *WS-MetadataExchange* definira protokol kojim usluge same pružaju korisnicima opis svojih sučelja. *WS-MetadataExchange* norma definira normirani skup operacija mrežne usluge kojima korisnik pristupa kako bi od usluge dohvatio WSDL opis sučelja. *WS-*

*MetadataExchange* omogućava povezivanje korisničkog programa s uslugama tijekom oblikovanja (engl. *design-time binding*) i tijekom izvođenja (engl. *run-time binding*) sustava.

Norme za definiranje funkcionalnosti kojima se osigurava kakvoća usluge prikazane na slici 2.7 dijele se na norme za ostvarenje pouzdane komunikacije porukama *WS-ReliableMessanging*, ostvarenje sigurnosti mrežnih usluga *WS-Security*, *WS-Trust*, *WS-SecureConversation* i *WS-Federation*, ostvarenje koordinacijskih mehanizama *WS-Coordination* te ostvarenje transakcijskih mehanizama *WS-AtomicTransaction* i *WS-BusinessActivity*.

*WS-ReliableMessaging* [24] norma je nadogradnja SOAP protokola kojom se omogućava definiranje pouzdanog prijenosa poruke od izvorišta do odredišta. Kao i kod drugih komunikacijskih protokola, komunikacija korištenjem SOAP protokolom podložna je različitim greškama. Najčešće pogreške su gubitak poruke, pojava više preslika iste poruke i promjena redoslijeda poslanih poruka. *WS-ReliableMessaging* protokol osigurava pouzdan prijenos SOAP poruke od izvorišta do odredišta u nepouzdanim okolinama.

*WS-Security* [25] norma definira proširenje SOAP protokola kojim se omogućuje definiranje razine zaštite poruka. Različite razine zaštite poruke ostvaruju se mehanizmima očuvanja cjelovitosti poruke, povjerljivosti sadržaja poruke i autentikacije pojedinačne poruke (engl. *single message authentication*). Navedenim mehanizmima ostvaruju se različite tehnike enkripcije poruka te je moguće zadovoljiti različite sigurnosne modele. Nadalje, *WS-Security* norma određuje općeniti mehanizam ugradnje sigurnosnih znački (engl. *security token*) u poruke. Normom nije propisan određeni tip sigurnosnih znački, već je norma oblikovana kako bi omogućila ugradnju različitih formata sigurnosnih znački.

Koordinacijske i transakcijske norme prikazani na slici 2.7 koriste se za ostvarenje različitih protokola koordinacije raspolijeljenih primjenskih sustava zasnovanih na uslugama. U računalnim sustavima koristi se veliki broj koordinacijskih protokola koji se razlikuju po funkcionalnostima, ali je svima zajedničko da se tijekom koordinacije između svih sudionika razmjenjuje informacija koja se naziva *koordinacijsko okružje* (engl. *coordination context*). Koordinacijsko okružje osigurava povezivanje pojedinačnih operacija sudionika u logički jedinstvenu koordiniranu operaciju ili aktivnost.

*WS-Coordination* norma [29] definira općeniti koordinacijski radni okvir (engl. *framework*) koji se koristi tijekom oblikovanja posebnih poslovnih procesa. *WS-Coordination* normom definira se općeniti i proširivi način zapisa i prenošenja koordinacijskog okružja između svih sudionika koordinacije.

Dvije osnovne jedinke definirane *WS-Coordination* normom su *koordinator* i *sudionik*. Koordinator je odgovoran za stvaranje, održavanje i prenošenje koordinacijskog okružja

sudionicima. Sudionici su jedinke koji se ponašaju u skladu s pravilima definiranim koordinacijskim protokolom i pod definiranim pravilima obavljaju određeni dio logike raspodijeljenog primjenskog sustava.

Akcije definirane *WS-Coordination* normom su *pokretanje* i *registracija*. Pokretanjem se stvara koordinator i pripadajuće koordinacijsko okružje. Nakon što je koordinator stvoren, svi se sudionici prijavljuju koordinatoru procesom registracije i odabiru željeni protokol koordinacije. Nakon što su se sudionici registrirali i odabrali protokol, koordinator im šalje poruke koje sadrže koordinacijsko okružje.

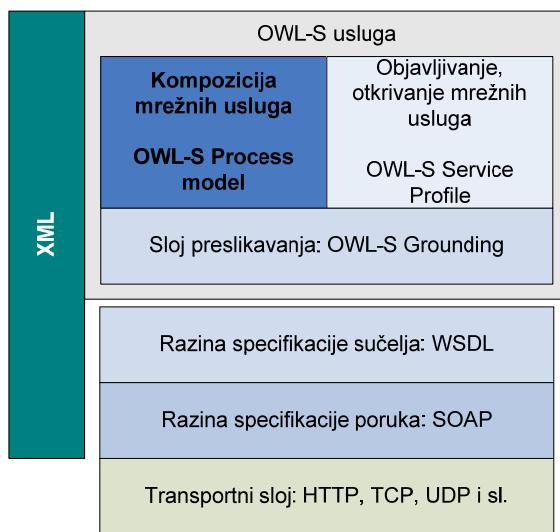
*WS-Transactions* [21] je prvi normirani koordinacijski protokol zasnovan na *WS-Coordination* normi. *WS-Transactions* je proširenje *WS-Coordination* norme i definira dva protokola transakcijske koordinacije: *WS-AtomicTransaction* i *WS-BusinessActivities*. Normom su definirana dva transakcijska modela, jer ne postoji jedinstveni transakcijski model koji je pogodan za Web okruženje u kojem se izvode programski sustavi zasnovani na uslugama. *WS-AtomicTransaction* model koordinacije koristi se za razvoj programskih sustava koji se zasnivaju na kratkotrajnim raspodijeljenim aktivnostima od kojih se svaka izvršava u cijelosti. *WS-BusinessActivity* transakcijski model koristi se za razvoj programskih sustava koji su zasnovani na dugotrajnim raspodijeljenim aktivnostima kod kojih nije moguće osigurati svojstvo nedjeljivosti.

## 2.4 Semantičke mrežne usluge

Semantičke mrežne usluge definirane su u sklopu razvoja semantičkog Web-a. Konačni cilj semantičkog Web-a je proširiti mogućnosti današnjeg Web-a i omogućiti pristup sredstvima na osnovi njihova sadržaja, a ne samo na osnovi ključnih riječi. Polazna točka razvoja semantičkog Web-a definiranje je sustava oznaka kojima je omogućen opis svojstava sredstava. Za potrebe opisa svojstava sredstava razvijen je niz ontologija i jezika za opis ontologija. Jedan od prvih jezika namijenjen opisu računalno razumljivih ontologija i označavanju semantičkih svojstava mrežnih sadržaja je *DARPA Agent Markup Language* (DAML) [54]. DAML jezik zasnovan je na jezicima XML i RDF (*Resource Description Framework* [55]). Za potrebe opisa mrežnih usluga razvijena je posebna ontologija OWL-S (*Web Ontology Language for Services*) [59]. Ontologija mrežnih usluga opisana OWL-S jezikom omogućava automatsko otkrivanje, pozivanje, kompoziciju, međudjelovanje i nadgledanje izvođenja mrežnih usluga.

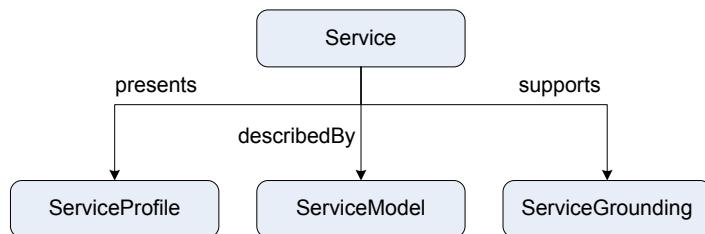
### 2.4.1 Norme semantičkih mrežnih usluga

Semantičke mrežne usluge zasnivaju se na osnovnim Web Services normama SOAP i WSDL. Iako WSDL opis mrežnih usluga definira sučelje, komunikacijski protokol i mjesto mrežne usluge, on ne sadrži informacije o funkcionalnostima mrežne usluge. Zbog toga semantičke mrežne usluge proširuju spomenute WS norme s OWL-S ontologijom. Slika 2.12. prikazuje odnose Web Services normi SOAP i WSDL, te normi semantičkih usluga zasnovanih na OWL-S ontologiji.



**Slika 2.12 Norme semantičkih mrežnih usluga**

OWL (*Web Ontology Language*) [60] je na XML-u zasnovan jezik namijenjen opisu ontologija. Ontologija je u računalnoj znanosti definirana kao formalno određenje dijeljenih koncepata [57]. *Koncept* je apstraktni model opisa objekata i pojava iz okoline koji sadrži samo njihova bitna svojstva. Pojam *formalno* u definiciji ontologije označava da ontologija mora imati dobro definiranu semantiku, dok pojam *dijeljeno* označava da ontologija definira znanja koja su opće prihvaćena. Ontologija definirana jezikom OWL sastoji se od skupa razreda i svojstava kojima se opisuje koncept iz neke domene. Nadalje, jezik OWL definira široki skup konstrukata za opis odnosa između razreda i svojstava.



**Slika 2.13 OWL-S ontologija**

*Ontology Web Language for Services* (OWL-S) [58] je specifična OWL ontologija koja sadrži razrede i svojstva prikladna za semantički opis mrežnih usluga. Dijagram na slici 2.13.

prikazuje OWL razrede i svojstva od kojih se sastoji OWL-S opis mrežne usluge. Čvorovi predstavljaju OWL razrede, a grane svojstva. Semantička mrežna usluga opisana je primjerkom razreda *Service* s odgovarajućim vrijednostima svojstava *presents*, *describedBy* i *supports*. Vrijednosti navedenih svojstava definirane su razredima *ServiceProfile*, *ServiceModel* i *ServiceGrounding*.

Razred *ServiceProfile* sadrži svojstva kojima se opisuju funkcionalna i nefunkcionalna svojstva usluge. Primjeri funkcionalnih svojstava su ulazni i izlazni parametri, uvjeti i posljedice poziva operacije usluge, dok su primjeri nefunkcionalnih svojstava poslovni razred usluge, informacije o vlasniku usluge i informacije o kakvoći usluge. Razred *ServiceProfile* omogućuje objavu svojstava i funkcionalnosti mrežnih usluga i time omogućuje automatsko otkrivanje mrežnih usluga. Informacije sadržane u primjerku *ServiceProfile* razreda su proširenje informacija dostupnih u UDDI imenicima mrežnih usluga ostvarenih WS tehnologijama.

Preslikavanje apstraktnog OWL-S modela mrežne usluge na njeno stvarno ostvarenje definirano je razredom *ServiceGrounding*. Ostvarenje mrežne usluge najčešće je opisano jezikom WSDL. Razred *ServiceGrounding* definira preslikavanja OWL razreda u WSDL tipove podataka, ulaze i izlaze OWL procesa u ulazne i izlazne WSDL poruke, te jednostavne OWL procese u WSDL operacije. Osnovna namjena *ServiceGrounding* razreda je potpora automatskom pozivanju mrežnih usluga.

Razredi *ServiceProfile* i *ServiceGrounding* omogućuju ostvarenje sustava oblikovanih prema načelu osnovnog modela arhitekture zasnovane na uslugama proširenog potporom za automatsko izvođenje procesa objave, pronalaska i povezivanja mrežnih usluga.

Razred *ServiceModel* opisuje način rada mrežne usluge. Funkcionalnost mrežne usluge oblikuje se kao proces opisan posebnom ontologijom čiji je osnovni element razred *Process*. OWL-S ontologija dvojako definira proces. Proces obavlja transformaciju skupa ulaznih podataka u skup izlaznih podataka, dok s druge strane proces mijenja stanje okoline u kojoj se izvodi. Promjena stanja okoline određena je preuvjetima (engl. *preconditions*) i posljedicama (engl. *effects*) izvođenja procesa.

Razred *Process* može biti oblikovan na tri načina, kao nedjeljivi proces uporabom razreda *AtomicProcess*, kao jednostavni proces uporabom razreda *SimpleProcess* i kao složeni proces uporabom razreda *CompositeProcess*.

*AtomicProcess* je razred kojim se opisuje jednostavni proces koji ne sadrži niti jedan podproces, te ga je moguće izravno pozvati slanjem odgovarajuće ulazne poruke. Sa stajališta korisnika usluge nedjeljivi proces izvodi se u jednom koraku i korisnik nema uvida u tijek izvođenja procesa. Svaki nedjeljivi proces povezan je s primjerkom razreda

*ServiceGrounding* kako bi se omogućilo njegovo pozivanje, odnosno kako bi korisnik znao formatirati ulaznu poruku potrebnu za izvođenje procesa.

Razred *SimpleProcess* definira jednostavni proces koji se sa stajališta korisnika također izvodi u jednom koraku, ali ga nije moguće izravno pozvati. Jednostavni proces nije moguće izravno pozvati, jer nije povezan s primjerkom razreda *ServiceGrounding* koji bi definirao vezu s ostvarenjem usluge. Jednostavni procesi koriste se za poopćavanje nedjeljivih procesa ili za pojednostavljeni prikaz složenog procesa koji se koristi u metodama planiranja i zaključivanja.

Razred *CompositeProcess* definira način povezivanja više jednostavnih ili složenih procesa u jedan složeni proces. Redoslijed izvođenja procesa koji tvore složeni proces opisuje se konstruktima tijeka izvođenja. Primjeri konstrukata tijeka izvođenja su *sequence*, *if-then-else*, *split* i *unordered*.

## 2.5 Programski model oblikovanja sustava zasnovanih na uslugama

Programski model definira metodologiju oblikovanja programskih sustava i tehnike programskog ostvarenja oblikovanih sustava. Potpuni programski model određen je modelom *izračunljivosti* (engl. *computation model*) i modelom *koordinacije* (engl. *coordination model*). Model izračunljivosti omogućava programerima izgradnju samostalnih programskih cjelina koje obavljaju jednodretvene, slijedne aktivnosti. Model koordinacije omogućava programerima povezivanje jednostavnih aktivnosti u složenije programske cjeline. Koordinacijski model određuje način stvaranja pojedinih aktivnosti i način njihove komunikacije.

U potpunom programskom modelu, ostvarenje sustava zasnovanih na uslugama promatra se na dvije razine. Prva razina definira tehnike programiranja za ostvarenje pojedinačnih mrežnih usluga i naziva se *programiranje na malo* (engl. *programming in the small*). Programiranje na malo koristi tradicionalne programske jezike poput jezika C, C++, Java i C#, a zasniva se na tradicionalnim komponentnim tehnologijama i okolinama za njihovo izvođenje poput J2EE, .NET okolina. Razvijene komponente izvode se u poslužiteljskim okruženjima i izložene su putem računalne mreže kao mrežne usluge. Tako oblikovane mrežne usluge nazivaju se *jednostavne mrežne usluge*.

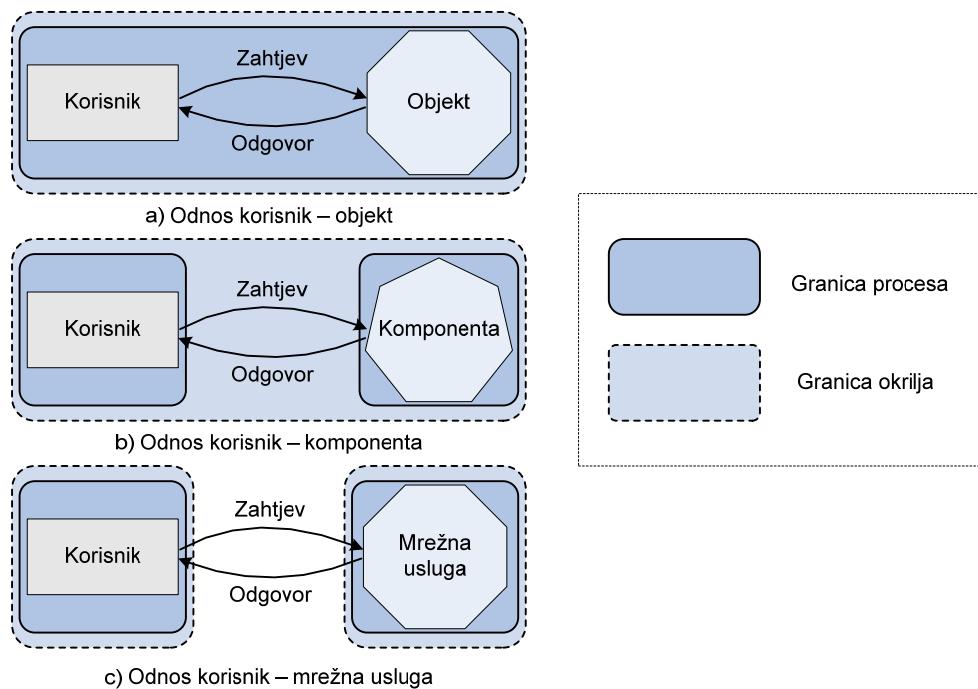
Koordinacijski model u okruženju mrežnih usluga definira drugu razinu izgradnje programskih sustava kojom se povezuju postojeće mrežne usluge. Ovako izgrađene programske sustave moguće je putem računalne mreže izložiti kao nove, složene mrežne usluge. Izgradnja programskih sustava kompozicijom postojećih mrežnih usluga naziva se

*programiranje na veliko* (engl. *programming in the large*). *Programiranje na veliko* određuje tehnike programiranja koje koriste različite koordinacijske jezike (engl. *coordination languages*) i jezike definiranja tijeka rada (engl. *workflow languages*).

### 2.5.1 Odnos objekta, komponente i mrežne usluge

Objekti, komponente i mrežne usluge su općeprihvaćeni elementi izgradnje programskih sustava, imaju mnoga zajedničkih svojstava, ali i znatne razlike u radnim svojstvima. Navedeni elementi ostvaruju određenu funkcionalnost definiranu programskim kôdom i izlažu ostvarene funkcionalnosti putem dobro definiranih programskih sučelja. Nadalje, navedeni programski elementi izvode se unutar računalnog procesa i omogućuju povezivanje s korisničkim procesom.

Dva su ključna čimbenika kojima se definira razlika objekata, komponenti i mrežnih usluga: *okruženje* i *okrilje*. *Okruženje* je čimbenik koji definira odnos mesta izvođenja procesa programskega elementa i mesta izvođenja korisničkog procesa. *Okrilje* definira radnu okolinu koja pruža potporu izvođenju programskega elementa i korisničkog programa. Objekt, komponenta i mrežna usluga imaju i drugih razlika, ali je većina razlika posljedica navedenih čimbenika.



**Slika 2.14 Odnosi korisnika i objekta, komponente, mrežne usluge obzirom na lokaciju i okruženje**

Razlike objekta, komponente, i mrežne usluge zasnovane na okrilju i okolini izvođenja prikazane su slikom 2.14. Objekti se izvode u istom procesu kao i korisnički program. Okrilje

u kojem se izvodi objekt i korisnički program je također isto, jer nije moguće da se jedan proces istovremeno izvodi u više radnih okrilja. Raširene tehnologije ostvarenja objektnog odnosa programskog elementa i korisnika su jezici C# i Java. Komponente se izvode u različitom procesu od procesa u kojem se izvodi korisnički program, ali okrilja u kojima se izvode ti procesi su jednaka. Primjeri tehnologija ostvarenja komponentnih odnosa programskog elementa i korisnika su programska okrilja Corba i Microsoft .NET. Mrežne usluge se, kao i komponente, izvode u različitom procesu od procesa u kojem se izvodi korisnički program. Razlika između komponenata i mrežnih usluga je da se mrežna usluga i korisnički program ne moraju izvoditi u istim okriljima.

### ***Učinkovitost***

Različita učinkovitost izvođenja objekta, komponente i mrežne usluge posljedica je navedenih razlika u načinu povezivanja i raspodijeljenosti programskih elemenata i korisnika.

Kako se objekti izvode u istom procesu kao i njihovi korisnici, razrješavanje imena metode (engl. *method resolution*), odnosno preslikavanje poziva metode u izvođenje programskog kôda kojim je ostvarena metoda, odvija se unutar istog procesa kao što je prikazano na slici 2.14a. Zbog toga je omogućeno brzo, učinkovito i sigurno ostvarenje razrješavanje imena metode uporabom optimirane tablice pretraživanja.

Komponente se izvode u različitom procesu od korisničkog procesa, kao što je prikazano na slici 2.14b, te svaki korisnički poziv metode komponente uzrokuje međuprocesnu komunikaciju. Međuprocesni komunikacijski mehanizam raspodijeljen je na dio programskog kôda koji se izvodi u korisničkom procesu i dio koji se izvodi na poslužiteljskom računalu. Korisnički komunikacijski modul prihvata pozive metode od korisničkog programa te prosljeđuje ime metode i listu parametara komunikacijskom modulu u poslužiteljskom procesu. Nakon što poslužiteljski proces primi podatke o imenu metode, primjenjuje se isti postupak razrješavanja imena metode kao i kod objekata. Vrijeme između poziva metode komponente i početka njenog izvođenja veće je od vremena između poziva metode objekta i početka njenog izvođenja za vrijeme utrošeno za međuprocesnu komunikaciju s poslužiteljskim procesom. Osnovna značajka međuprocesne komunikacije unutar istog okrilja je zasnovanost na binarnim komunikacijskim protokolima.

Mrežne usluge su zbog razdvojenosti procesa i različitih okrilja izvođenja manje učinkovite od objekata i komponenata. Zbog raznorodnih okrilja izvođenja, mrežne usluge koriste normirane WS protokole komunikacije (SOAP protokol) koji su manje učinkoviti od binarnih međuprocesnih komunikacijskih protokola specifičnih za komponente tehnologije. Transportni protokoli mrežnih usluga su protokoli prihvaćeni od svih proizvođača okrilja za izvođenje i sa stajališta učinkovitosti definiraju najmanji zajednički nazivnik radnih svojstava

svih okrilja. Lošija radna svojstva komunikacijskog protokola mrežnih usluga posljedica su rješavanja problema međudjelovanja raznorodnih okrilja izvođenja procesa. Osim toga, normirani komunikacijski protokoli mrežnih usluga su zasnovani na tekstu i zbog toga uvode veće probleme sigurnosti od binarnih protokola. Sažetak značajki objekata, komponenti i mrežnih usluga naveden je u tablici 2.1.

**Tablica 2-1 Usporedba osnovnih značajki objekta, komponente i mrežne usluge**

Svojstvo	Objekti	Komponente	Mrežne usluge
<b>Okruženje izvođenja</b>	Unutar istog procesa kao i korisnički program	Različiti proces izvođenja od korisničkog procesa	Različiti proces izvođenja od korisničkog procesa
<b>Okrilje izvođenja</b>	Isto okrilje kao i okrilje korisnika	Isto okrilje kao i okrilje korisnika	Različito okrilje od okrilja korisnika
<b>Brzina</b>	Vrlo brza komunikacija s korisnikom	Spora komunikacija s korisnikom	Vrlo spora komunikacija s korisnikom
<b>Razvoj</b>	Ista osoba koja razvija i korisnički program	Razvija ista grupa koja razvija i korisnički program	Razvija različita organizacija od one koja razvija korisnički program

Navedena svojstva i različitosti objekata, komponenti i mrežnih usluga određuju smjernice odabira programskih elemenata najprikladnijih za ostvarenje programskog sustava. Objekti, komponente i mrežne usluge ne smiju se razmatrati kao međusobno isključivi elementi tijekom oblikovanja i ostvarenja programskog sustava, već ih je potrebno promatrati kao nadopunjuće elemente izgradnje sustava. Mrežne usluge je najbolje koristiti za povezivanje samostalnih programskih sustava, komponente za povezivanje procesa unutar sustava, a objekte kao elemente izgradnje programskih cjelina unutar procesa.

## 3 Kompozicija mrežnih usluga

Razvoj *Web Services* tehnologija i računalne arhitekture zasnovane na uslugama potaknuo je razvoj nove paradigme izgradnje raspodijeljenih programskih sustava koja se zasniva na kompoziciji mrežnih usluga. Kompozicija mrežnih usluga je postupak izgradnje programskog sustava čija je funkcionalnost ostvarena povezivanjem mrežnih usluga. Iako je načelo izgradnje programskih sustava kompozicijom postojećih programskih komponenti poznato već dulje vrijeme, sve do nedavno bilo je mnogo čimbenika koji su ograničavali raširenu primjenu kompozicije. Glavni ograničavajući čimbenici bili su nerazvijeno tržište komponenti, problemi međudjelovanja komponenti ostvarenih različitim tehnologijama i nedostatak programskih jezika namijenjenih kompoziciji.

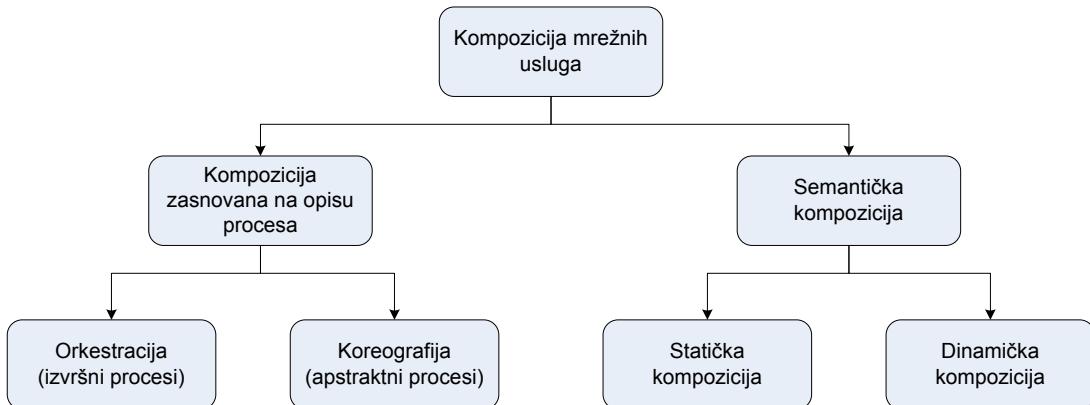
Većina problema međudjelovanja riješena je razvojem i normizacijom *Web Services* tehnologija. Prelaskom s komponentnih tehnologija na tehnologije mrežnih usluga, omogućeno je stvaranje globalnog, otvorenog tržišta programskih komponenti koje održavaju pružatelji usluga (engl. *service providers*). Pružatelji usluga korisnicima jamče određenu kvalitetu usluge (engl. *quality of service*), odnosno brinu o svim programskim i sklopopovskim problemima koji mogu utjecati na funkcionalnost usluge.

Nadalje, razvijeno je više metoda kompozicije, programskih jezika i razvojnih alata namijenjenih razvijateljima programskih sustava zasnovanih na uslugama. Najraširenija metoda kompozicije mrežnih usluga je kompozicija zasnovana na opisu procesa. Nadalje, razvijeno je mnogo programskih jezika namijenjenih različitim metodama kompozicije usluga. Postojeći programski i skriptni jezici proširuju se konstruktima za uporabu mrežnih usluga, ali se istodobno razvijaju novi programski jezici isključivo namijenjeni opisu kompozicije mrežnih usluga. Kako bi se razvoj programskih sustava zasnovanih na uslugama pojednostavio, razvijeno je više razvojnih alata koji omogućuju brzi razvoj programskih sustava zasnovanih na uslugama. Unatoč velikom broju programskih jezika i razvojnih alata, kompozicija mrežnih usluga trenutno nije prilagođena potrebama krajnjih korisnika koji imaju ograničena znanja o tehnikama programiranja.

### 3.1 Podjela metoda kompozicije mrežnih usluga

Razvijeno je mnogo metoda i jezika za opis kompozicije mrežnih usluga koji se zasnivaju na različitim pristupima kompoziciji. Podjela postupaka kompozicije mrežnih usluga prikazana je na slici 3.1. Postupci kompozicije mrežnih usluga dijele se na kompoziciju zasnovanu na opisu procesa i semantičku kompoziciju mrežnih usluga. Kompozicija zasnovana na opisu procesa dijeli se na orkestraciju i koreografiju. Semantička kompozicija mrežnih usluga

zasniva se na semantičkom opisu mrežnih usluga koji omogućava statičku i dinamičku kompoziciju mrežnih usluga.



Slika 3.1 Podjela postupaka kompozicije mrežnih usluga

### 3.2 Kompozicija mrežnih usluga zasnovana na opisu procesa

Kompozicija zasnovana na opisu procesa razvijena je na osnovi tehnika i metoda oblikovanja poslovnih procesa poznatih iz područja upravljanja poslovnim procesima (engl. *Business Process Management*). Poslovni se proces definira kao logički povezani skup aktivnosti koje organizacija mora obaviti određenim redoslijedom kako bi se postigao zadani poslovni cilj.

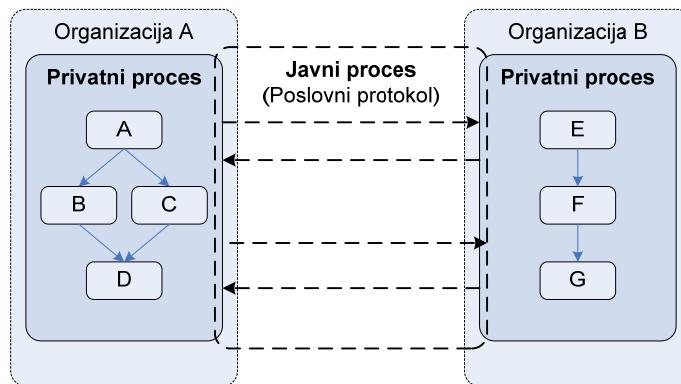
Osnovni elementi poslovnog procesa su aktivnosti, tok podataka i tijek izvođenja. Aktivnosti poslovnog procesa su dobro definirane poslovne funkcije koje mogu biti jednostavne ili složene. Jednostavne aktivnosti ne mogu se podijeliti na manje aktivnosti, a složene aktivnosti sastoje se od niza aktivnosti koje su oblikovane kao poslovni procesi, odnosno podprocesi. Tok podataka opisuje podatke koji se razmjenjuju između pojedinih aktivnosti, dok tijek izvođenja definira redoslijed izvođenja aktivnosti, npr. slijedno, paralelno ili uvjetno izvođenje aktivnosti. Primjena opisa poslovnih procesa na opis kompozicije mrežnih usluga zasniva se na preslikavanju aktivnosti poslovnih procesa u pozive mrežnih usluga. Pritom mrežne usluge, kao i aktivnosti, mogu biti jednostavne i složene.

#### 3.2.1 Poslovni procesi

Poslovni se procesi, ovisno o raspodijeljenosti procesa, dijele na izvodljive poslovne procese i na poslovne protokole. Izvodljivi poslovni procesi specificiraju poslovni proces unutar granica jedne organizacije. Poslovni protokoli ili apstraktni poslovni procesi koriste se za definiranje poslovnog procesa raspodijeljenog između više organizacija. Izvodljivim poslovnim procesima definira se stvarno ponašanje sudionika u poslovnom međudjelovanju, dok

poslovni protokoli koristeći opis procesa specificiraju globalno vidljivo međudjelovanje svih sudionika u poslovnim protokolima, bez opisa njihovog internog ponašanja.

Slika 3.2 prikazuje odnos izvodljivih poslovnih procesa i poslovnih protokola. Izvodljivi poslovni procesi su pod nadzorom jedne poslovne organizacije i zbog toga se još nazivaju i privatni poslovni procesi. Poslovni protokoli nisu pod potpunim nadzorom niti jedne organizacije koja sudjeluje u protokolu te se nazivaju javni procesi.



**Slika 3.2 Odnos privatnih i javnih poslovnih procesa**

Poslovni protokoli često definiraju složene modele međudjelovanja. Modeli poslovnog međudjelovanja sadrže niz razmjena poruka između ravnopravnih korisnika te dugotrajno međudjelovanje s očuvanjem stanja između dvije ili više strana. Kako bi se definiralo složeno poslovno međudjelovanje, potreban je formalni opis *protokola razmjene poruka* korištenog u međudjelovanju poslovnih procesa. Definicija takvih poslovnih protokola mora omogućiti određivanje globalno vidljivih pravila razmjene poruka svih strana uključenih u protokol, ali bez opisa njihovog unutarnjeg ostvarenja.

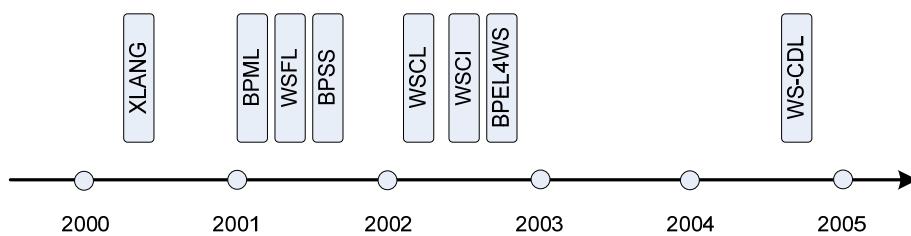
Dva su glavna razloga zašto se odvaja javni aspekt poslovnog protokola od unutarnjih ili privatnih aspekata. Prvi je da poslovne organizacije ne žele poslovnim suradnicima, koji sudjeluju u protokolu, javno objaviti unutarnje ostvarenje svojih poslovnih procesa. Druga prednost odvajanja javnog i privatnog dijela poslovnog protokola je mogućnost promjene ostvarenja privatnih procesa bez utjecaja na javni poslovni protokol.

### 3.2.2 Razvoj sustava oznaka za opis procesa

Prvi računalni sustavi koji su koristili izravne definicije poslovnih procesa bili su sustavi automatiziranja uredskih informacijskih sustava (engl. *Office Information Systems*) razvijeni 70-tih godina prošlog stoljeća, poput sustava *OfficeTalk* i *Scoop* [49][50]. Navedeni su sustavi koristili matematički formalizam Petrijevih mreža za opis poslovnih procesa. Pravi razvoj računalnih sustava za oblikovanje poslovnih procesa dogodio se tek tijekom 90-tih godina paralelno s razvojem i širenjem računalnih mreža. U tom razdoblju znatno je

napredovao razvoj tehnologija i sustava upravljanja *tijekom rada* (engl. *workflow technology, workflow management systems*).

Razvojem sustava za upravljanje tijekom rada pojavio se problem uporabe različitih sustava oznaka za opis poslovnih procesa. Osim toga, različiti sustavi koristili su različite tehnologije povezivanja komponenti u cjelokupni proces. Pokušaji organizacije WfMC (*Workflow Management Coalition*) da definira normirani jezik opisa poslovnih procesa rezultirao je objavom norme XPDL (*XML Process Definition Language*) [31] 2002. godine. No ta je norma tek djelomično riješila problem nepodudarnosti različitih sustava za upravljanje tijekom rada.

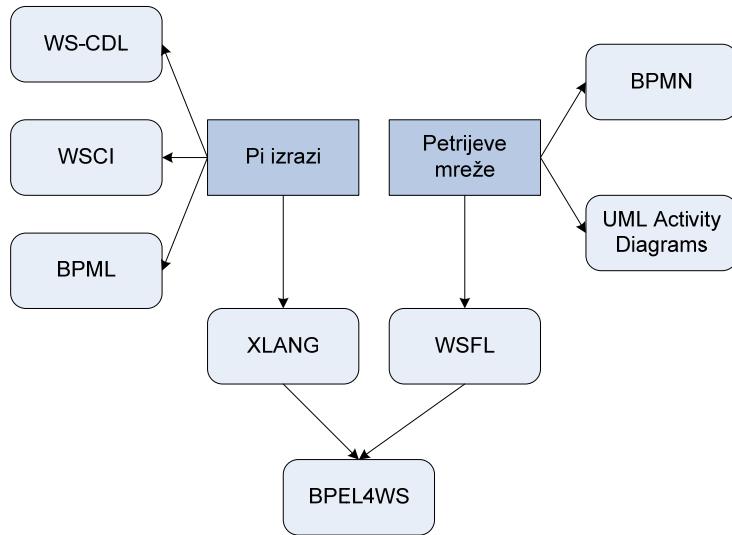


**Slika 3.3 Razvoj jezika opisa kompozicije mrežnih usluga zasnovane na opisu procesa**

Razvojem koncepta mrežnih usluga rezultati istraživanja iz područja oblikovanja poslovnih procesa i upravljanja radnim tijekom primjenjeni su u okruženju definiranja jezika za opis kompozicije mrežnih usluga zasnovane na opisu procesa. Slika 3.3 prikazuje značajnije jezike kompozicije mrežnih usluga razvijene tijekom posljednjih pet godina. Iako je razvijeno mnogo jezika za kompoziciju mrežnih usluga, neki jezici su postali *de facto* norma čime je riješen problem međudjelovanja različitih okrilja za izvođenje kompozicije mrežnih usluga. Primjeri jezika koji su postali *de facto* norme za opis kompozicije mrežnih usluga su jezici BPEL4WS i WS-CDL.

### 3.2.3 Teorijska osnova metoda oblikovanja poslovnih procesa

Već su prvi sustavi za definiranje poslovnih procesa bili teorijski zasnovani na određenim matematičkim modelima, poput sustava *OfficeTalk* koji je zasnovan na Petrijevim mrežama. Većina jezika kompozicije mrežnih usluga za oblikovanje poslovnih procesa također ima teorijsku pozadinu u matematičkim modelima. Dva matematička modela na kojima se zasnivaju trenutno najraširenije norme kompozicije mrežnih usluga su Petrijeve mreže i Pi izrazi. Na slici 3.4 prikazane su teorijske osnove nekoliko normiranih jezika za opis kompozicije mrežnih usluga.



Slika 3.4 Teorijske osnove jezika kompozicije mrežnih usluga

Kao što prikazuje slika 3.4 četiri normirana jezika kompozicije mrežnih usluga izvedena iz teorijskog modela Pi izraza su WS-CDL (*Web Services Choreography Description Language* [43]), WSCI (*Web Services Choreography Interface* [45]), BPML (*Business Process Modeling Language* [47]) i XLANG [41]. Tri jezika razvijena na osnovi modela Petrijevih mreža su BPMN (*Bussines Process Modeling Notation* [46]), UML Activity Diagrams i WSFL (*Web Services Flow Language* [42]). Jezik BPEL4WS (*Business Process Execution Language for Web Services*) [40] je specifičan, jer je nastao na osnovi specifikacija jezika XLANG i WSFL, te zbog toga sadrži osobine oba teorijska modela.

### Pi izrazi

Pi izrazi (engl. *Pi Calculus*) su matematički model za definiranje konkurentnih procesa koji izvode dinamičko međudjelovanje. Proces se sastoji od jedne ili više akcija čiji se redoslijed izvođenja definira konstruktima tijeka izvođenja. Akcija predstavlja slanje ili primanje poruke putem kanala. Kanal je konstrukt na osnovi čijeg imena se određuje proces koji je *druga strana* u procesu komunikacije. Tijekom slanja poruke, proces navodi ime kanala kojeg drugi proces koristi kako bi poslao odgovor na primljenu poruku. Jedna od osnovnih značajki Pi izraza je mobilnost koja omogućuje dinamičke promjene topologije komunikacijskog procesa kao odgovor na promjenu uvjeta okoline izvođenja procesa.

Tri osnovna aspekta modeliranja procesa koji se formalno oblikuju na osnovi Pi izraza i koji se koriste u jezicima kompozicije mrežnih usluga su upravljanje tijekom izvođenja, komunikacija zasnovana na porukama i mobilnost procesa. Upravljanje tijekom izvođenja opisuje se konstruktima slijednog, paralelnog i rekurzivnog izvođenja, te konstruktima uvjetnog grananja. Svojstvo mobilnosti omogućuje dinamičko prestrukturiranje procesa u smislu redefiniranja adresa sudionika u procesu.

### Petrijeve mreže

Petrijeve mreže [51] su formalni grafički jezik modeliranja procesa koji je primjenjiv i na opise poslovnih procesa. Petrijeve mreže omogućuju opis semantike upravljačkog tijeka izvođenja procesa, od jednostavnih opisa semantike pravila grana i spajanja tijeka izvođenja, do složenih opisa semantike sinkronizacijskih tehnika.

Osnovni elementi jezika su mjesto, prijelaz, značka i grana. Mjesto se grafički označava kružnicom i predstavlja fiksnu točku u izvođenju procesa. Prijelaz se označava kvadratom i predstavlja akciju ili događaj u procesu. Značka se označava točkom i tijekom izvođenja se pomiciće iz jednog mjesta u drugo. Granom se povezuje prijelaz i mjesto, odnosno mjesto i prijelaz. Proces se opisuje na osnovi veza prijelaza i mjesta, a stanje procesa određeno je mjestima koja sadrže značke. Jezici za kompoziciju mrežnih usluga razvijeni iz modela Petrijevih mreža koriste sustav oznaka prenošenja značke (engl. *token passing*) za opis semantike upravljanja tijekom izvođenja procesa.

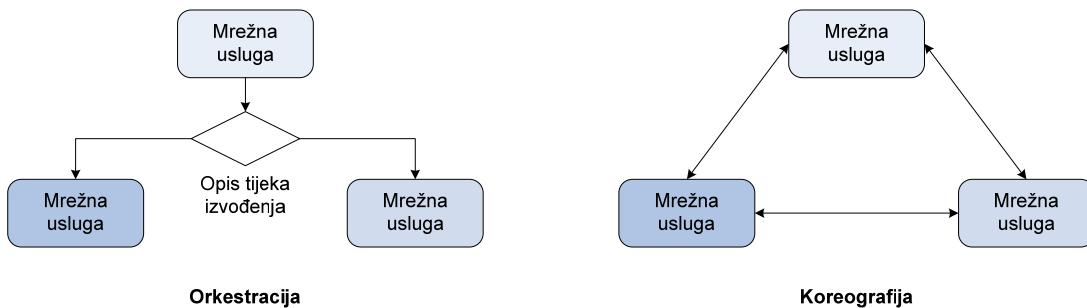
### 3.2.4 Kompozicija usluga primjenom orkestracije i koreografije

Kompozicija mrežnih usluga zasnovana na opisu procesa dijeli se na kompoziciju primjenom orkestracije i kompoziciju primjenom koreografije mrežnih usluga [37]. Podjela metoda kompozicije slijedi podjelu poslovnih procesa na izvodljive poslovne procese i na javne poslovne protokole.

Orkestracija (engl. *orchestration*) mrežnih usluga odnosi se na oblikovanje izvodljivih poslovnih procesa. Orkestracijom se definira izvodljivi proces koji povezuje funkcionalnosti mrežnih usluga prema definiranim poslovnim pravilima. Tako oblikovani složeni proces ponovno se izlaže kao mrežna usluga i njegova se funkcionalnost može koristiti u oblikovanju drugih poslovnih procesa. Orkestracija definira centralizirani tijek izvođenja međudjelovanja s mrežnim uslugama te definira međudjelovanje na razini poruka. Orkestracijom se definira međudjelovanje s mrežnim uslugama unutar ili izvan organizacijskih granica, što omogućuje definiranje izvodljivih procesa u koji su uključene različite organizacije, ali definiranim uvek upravlja jedna od strana uključenih u proces.

Za razliku od orkestracije koja ostvaruje centralizirani opis kompozicije, kompozicija ostvarena koreografijom ne sadrži centralizirano definirano međudjelovanje mrežnih usluga, već je opis međudjelovanja raspodijeljen. Raspodijeljeni opis podrazumijeva da svaka strana uključena u proces definira ulogu koju obavlja u procesu kompozicije. Koreografija opisuje javnu razmjenu poruka, odnosno slijed poruka koje se razmjenjuju između više mrežnih usluga. Proces razmjene poruka definiran koreografijom nije pod isključivim upravljanjem samo jedne strane uključene u komunikaciju.

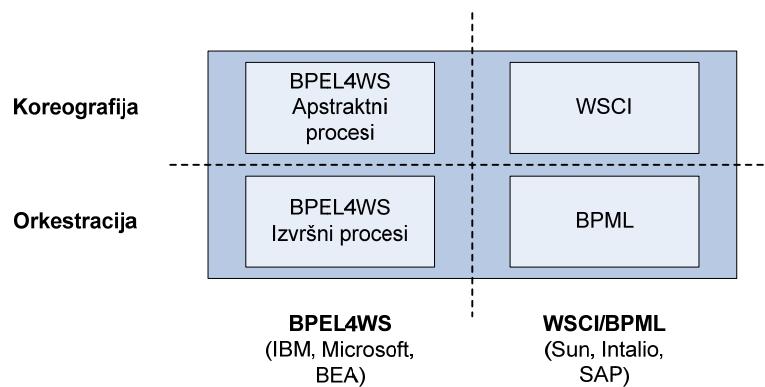
Slika 3.5 prikazuje odnos metoda koreografije i orkestracije. Metoda orkestracije prikladna je za potrebe opisa privatnih izvodljivih poslovnih procesa, dok je metoda koreografije prikladnija za opis javnih apstraktnih poslovnih procesa, odnosno poslovnih protokola.



**Slika 3.5 Odnos orkestracije i koreografije**

Primjeri jezika orkestracije su jezici BPEL4WS i BPML, dok su primjeri jezika koreografije jezici WSCI i WS-CDL. Iako se jezik BPEL4WS najčešće koristi za opis kompozicije izvodljivih procesa, odnosno za orkestraciju mrežnih usluga, uporabom jezika BPEL4WS moguće je definirati i apstraktne poslovne procese definirane koreografijom mrežnih usluga.

Trenutno još nije definirana općeprihvaćena norma jezika za opis orkestracije i koreografije, ali dvije industrijske skupine pokušavaju nametnuti vlastite norme orkestracije i koreografije. Slika 3.6. prikazuje prijedloge industrije za normiranje jezika orkestracije i koreografije. Tvrte IBM, Microsoft i BEA podupiru BPEL4WS kao jezik opisa orkestracije i koreografije, dok tvrtke Sun, Intalio i SAP pokušavaju nametnuti BPML kao jezik za opis orkestracije mrežnih usluga i WSCI kao jezik koreografije.



**Slika 3.6 Dvije predložene norme orkestracije i koreografije**

### 3.3 Kompozicija mrežnih usluga zasnovana na semantici

Za razliku od statičke kompozicije definirane metodama zasnovanim na opisu procesa, semantička kompozicija omogućuje definiranje statičke i dinamičke kompozicije mrežnih usluga. Statička se kompozicija odnosi na kompoziciju usluga za koju je tijek izvođenja

složene usluge unaprijed poznat i definira se tijekom oblikovanja kompozicije. Dinamička kompozicija podrazumijeva otkrivanje potrebnih usluga i njihovo povezivanje za vrijeme izvođenja složene usluge. Dinamička se kompozicija oblikuje na zahtjev korisnika (engl. *on-demand composition*). Ako korisnik zahtijeva funkcionalnost koju ne ostvaruje niti jedna postojeća mrežna usluga, mrežne se usluge mogu dinamički povezati kako bi zajedno ostvarile traženu funkcionalnost i odgovorile na zahtjev korisnika.

Dinamička kompozicija usluga podrazumijeva mogućnost pronalaženja usluga na osnovi njihovih sposobnosti i prepoznavanje usluga koje je potrebno povezati u složenu uslugu. Opisi sposobnosti mrežnih usluga podrazumijevaju semantičke opise sučelja mrežnih usluga, definiranih operacija, ulaznih i izlaznih parametara te drugih nefunkcionalnih svojstava usluga. Postojeće WS-\* norme ne propisuju normizirani sustav oznaka za semantički opis sučelja mrežnih usluga što je jedna od glavnih zapreka raširenijoj uporabi semantičkih mrežnih usluga i pripadajućih metoda kompozicije. Jezik koji omogućuje opis funkcionalnosti, svojstava i međusobnih odnosa mrežnih usluga je *Web Ontology Language for Services* (OWL-S) opisan u odjeljku 2.4.1. Jezikom OWL-S, osim semantičkih svojstava mrežne usluge, moguće je opisati i statičku kompoziciju.

Osim što omogućuje statičku kompoziciju mrežnih usluga, jezik OWL-S je osnova većine postupaka dinamičke, automatske kompozicije mrežnih usluga. Većina metoda automatske kompozicije mrežnih usluga zasniva se na postupcima planiranja iz područja umjetne inteligencije.

Općenito se problem planiranja u području umjetne inteligencije definira petorkom ( $S, S_0, G, A, \Gamma$ ), gdje  $S$  označava skup mogućih stanja,  $S_0$  označava početno stanje,  $G$  je željeno stanje koje sustav planiranja želi postići,  $A$  je skup akcija koje sustav planiranja izvodi kako bi promijenio stanje sustava, a  $\Gamma$  je relacija prijelaza koja za svaku akciju definira stanje koje je preduvjet za izvođenje akcije i novo stanje koje je posljedica izvođenja akcije.

U okruženju mrežnih usluga,  $S_0$  i  $G$  su početno i željeno stanje koje je specificirano zahtjevima korisnika mrežne usluge,  $A$  je skup raspoloživih mrežnih usluga, a relacija  $\Gamma$  opisuje promjenu stanja svake mrežne usluge nakon izvođenja operacije te mrežne usluge.

Osnovna osobina jezika OWL-S, po kojoj se razlikuje od ostalih jezika kompozicije, je mogućnost definiranja preduvjeta i posljedica poziva mrežne usluge. Konstruktima definiranja preduvjeta i posljedica moguće je definirati relaciju  $\Gamma$  postupka planiranja.

Metode dinamičke kompozicije mrežnih usluga zasnovane na postupcima planiranja uključuju postupke zasnovane na situacijskim izrazima (engl. *situation calculus*), planiranje zasnovano na pravilima (engl. *rule-based planning*) i dokazivanje teorema (engl. *theorem proving*) [36].

## 3.4 Jezici kompozicije usluga

Jezici kompozicije mrežnih usluga su jezici koji su definirani na višoj razini apstrakcije od tradicionalnih programskih jezika kao što su jezici C, C# i Java. Tradicionalni programski jezici zasnivaju se na modelu izračunljivosti (engl. *computation*), odnosno definiraju slijed operacija kojima se specificira *kako* rješiti određeni problem. Jezici kompozicije mrežnih usluga zasnivaju se na modelu koordinacije (engl. *coordination*) i njima se specificira način povezivanja postojećih programskih komponenti. Iako tradicionalni programski jezici omogućuju određene aspekte koordinacije između komponenti, oni nisu pogodni za opis međudjelovanja slabo-podijeljenih (engl. *coarse-grained*) elemenata kompozicije kao što su mrežne usluge. Međudjelovanje mrežnih usluga zasniva se na razmjeni složenih XML dokumenata, a rukovanje takvim dokumentima upotrebom tradicionalnih programskih jezika često je vrlo složeno.

### 3.4.1 Svojstva jezika za kompoziciju

Osnovni zahtjevi koje moraju zadovoljiti jezici orkestracije mrežnih usluga su prilagodljivost, grupiranje jednostavnih naredbi u složene i rekurzivna kompozicija.

Jedno od najznačajnijih svojstava koje mora pružiti jezik kompozicije je prilagodljivost. Prilagodljivost se postiže definiranjem jasne granice između koordinacijske logike procesa i poslovne logike mrežnih usluga koje se pozivaju. Time je omogućena promjena ili nadogradnja postojećih usluga bez potrebe za promjenom koordinacijske logike procesa kojim je opisana kompozicija.

Jezik kompozicije mora omogućiti jednostavne naredbe za komunikaciju s mrežnim uslugama i strukturalne naredbe za upravljanje tijekom izvođenja. Jednostavna naredba opisuje međudjelovanje s nekom jedinkom izvan granica procesa. S druge strane, strukturalne naredbe upravljaju tijekom cjelokupnog procesa i definiraju način povezivanja jednostavnih naredbi u složene.

Rekurzivna kompozicija omogućava definiranje kompozicije korištenjem rezultata prethodnih kompozicija. Poslovnom procesu omogućeno je uzajamno djelovanje s više mrežnih usluga. Tako definirani poslovni proces može i sam biti ostvaren kao jedna mrežna usluga i omogućiti ponovnu iskoristivost procesa u definiranju novog poslovnog procesa više razine.

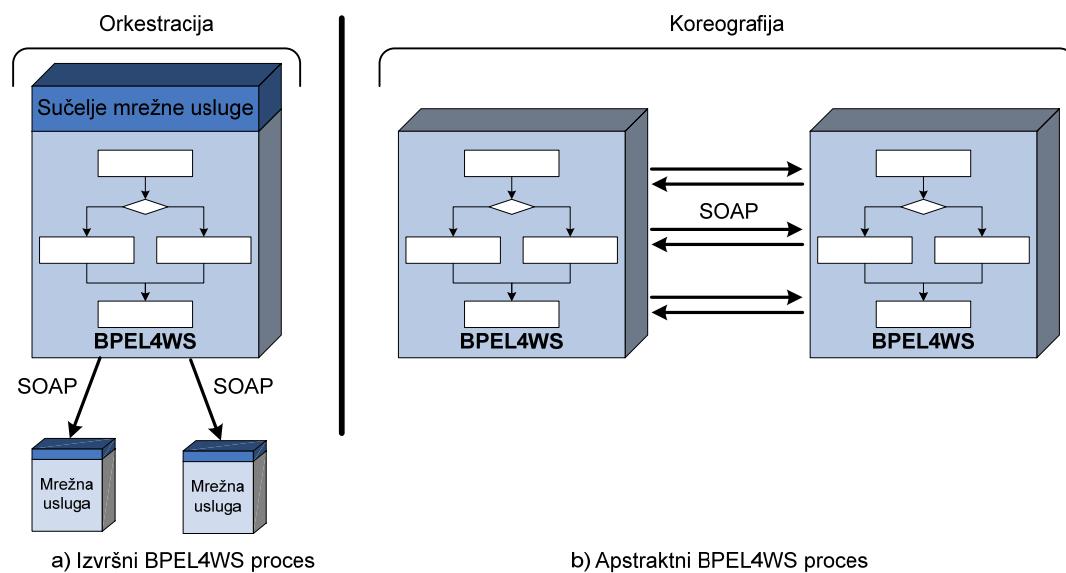
Nadalje, jezici orkestracije i koreografije moraju zadovoljavati osnovne zahtjeve za očuvanjem cjelokupnosti i jednoznačnosti cjelokupnog međudjelovanja mrežnih usluga. Ti zahtjevi sadrže očuvanje stanja međudjelovanja, korelaciju poruka asinkrone komunikacije, upravljanje iznimkama i potporu transakcijskom načinu rada.

Jezici za opis dugotrajnih (engl. *long-running services*) kompozicija mrežnih usluga moraju omogućiti rukovanje iznimkama i očuvanje cijelokupnosti transakcije. Na primjer, dugotrajne mrežne usluge ne mogu držati isključivi pristup nekom dijeljenom sredstvu tijekom cijelog razdoblja izvođenja transakcije.

### 3.4.2 BPEL4WS

BPEL4WS (*Business Proces Execution Language for Web Services*) je jezik za opis poslovnih procesa zasnovanih na mrežnim uslugama. BPEL4WS procesi koriste funkcionalnosti drugih mrežnih usluga, povezuju ih prema pravilima poslovnog procesa i tako definiranu složenu funkcionalnost izlažu putem normiranih sučelja mrežnih usluga.

Korištenjem jezika BPEL4WS moguće je formalno opisati privatne poslovne procese i javne protokole poslovnog međudjelovanja. Na taj se način proširuje osnovni model međudjelovanja mrežnih usluga s mogućnošću definiranja poslovnih transakcija. Prošireni model međudjelovanja mrežnih usluga omogućava objedinjavanje poslovnih procesa unutar i izvan granica poslovnih organizacija.



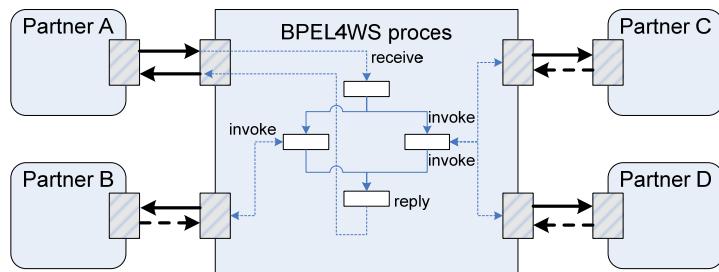
Slika 3.7 Izvodljivi i apstraktni BPEL4WS procesi

Primjer izvodljivog BPEL4WS procesa, koji je ostvaren orkestracijom dvije mrežne usluge, prikazan je na slici 3.7a, dok je na slici 3.7b prikazan apstraktni proces definiran koreografijom dva izvodljiva BPEL4WS procesa.

Jezik BPEL4WS zasnovan je na XML-u i zbog toga se njegova specifikacija oslanja na općeprihvaćene XML norme WSDL, XML Schema i XPath. Komunikacija s mrežnim uslugama definirana je WSDL opisom operacija mrežnih usluga. Format podataka korišten za komunikaciju s mrežnim uslugama definiran je dijelom WSDL opisa usluge koji definira

parametre operacija. Interni podaci korišteni za opis stanja BPEL4WS procesa i upravljanje tijekom izvođenja definirani su XML Schema normom. Manipulacija XML podacima BPEL4WS procesa ostvaruje se upotrebom jezika XPath. Zbog zasnovanosti na XML-u, BPEL4WS omogućuje definiranje opisa poslovnih procesa neovisnih o računalnom i programskom okrilju.

Jezik BPEL4WS definira model i gramatiku opisa ponašanja poslovnog procesa koji se zasnivaju na međudjelovanju dva tipa jedinki, procesa i mrežnih usluga suradnika. Odnos navedenih jedinki prikazan je na slici 3.8. Proces koristi funkcionalnosti suradnika isključivo međudjelovanjem putem njihovih normiranih sučelja. Funkcionalnosti usluga suradnika ostvaruju dijelove funkcionalnosti poslovnog procesa, a naredbama jezika BPEL4WS definira se koordinacijska logika potrebna za povezivanje dijelova funkcionalnosti u cjeloviti poslovni proces. U jeziku BPEL4WS moguće je definirati rukovanje iznimkama i pogreškama u izvođenju poslovnog procesa, te je moguće definirati kompenzacijске aktivnosti kojima se, u slučaju pogreške, poništavaju određene aktivnosti procesa.



Slika 3.8 Kompozicija mrežnih usluga definirana BPEL 4WS procesom

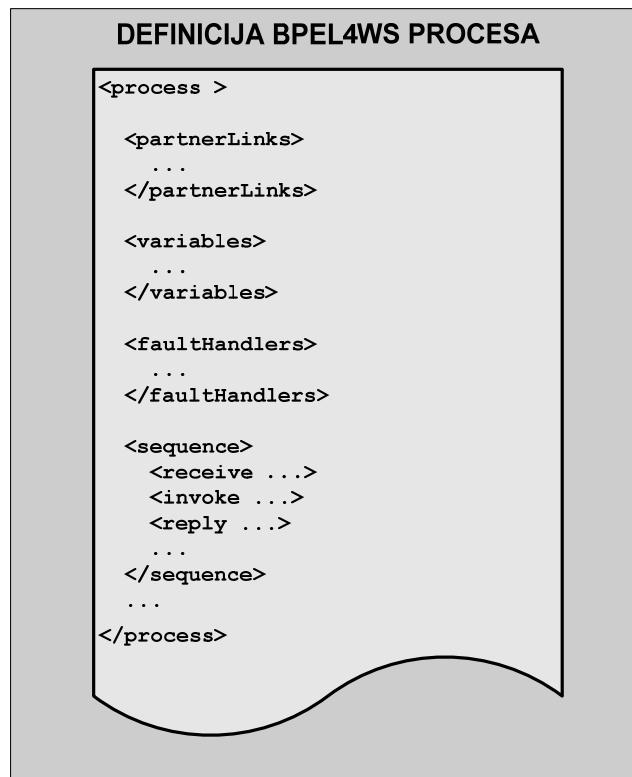
Osnovna struktura opisa BPEL4WS procesa, prikazana na slici 3.9, sastoji se od elementa `<partnerLinks>`, elementa `<variables>`, elementa `<faultHandlers>` i elementa `<sequence>`.

Element `<partnerLinks>` definira lokalna imena mrežnih usluga suradnika s kojima BPEL4WS proces komunicira. Tijekom definiranja lokalnog imena usluge suradnika specificira se i uloga suradnika u komunikaciji s BPEL4WS procesom. Usluge suradnici u komunikaciji imaju uloge korisnika BPEL4WS procesa, odnosno oni pozivaju BPEL4WS proces ili BPEL4WS proces poziva usluge suradnike.

Sve podatkovne varijable potrebne za komunikaciju sa suradnicima ili za spremanje stanja koordinacijske logike BPEL4WS procesa definiraju se unutar elementa `<variables>`. Deklaracija varijable sastoji se od imena varijable i tipa varijable. Tip varijable određuje se primjenom elementa `<messageType>` definiranog u WSDL opisu suradnika ili jednostavnog tipa definiranog imenom XML Schema elementa unutar WSDL opisa.

Kompenzacijске akcije koje se izvode u slučaju pojave pogreške tijekom komunikacije s mrežnim uslugama suradnicima definiraju se unutar elementa `<faultHandlers>`. Sve

pogreške unutar BPEL4WS procesa identificiraju se prema imenu definiranom u WSDL opisu usluge suradnika. Element `<faultHandlers>` sastoji se od niza parova imena pogrešaka i definicije kompenzacijskih akcija. Kompenzacijске akcije definiraju se primjenom istog skupa aktivnosti kojim se opisuju i aktivnosti procesa unutar bloka `<sequence>`.



Slika 3.9 Struktura opisa BPEL4WS procesa

Blok `<sequence>` sadrži glavni opis logike poslovnog procesa. Osnovni elementi definicije poslovne logike BPEL procesa su aktivnosti. Aktivnosti se dijele na jednostavne i strukturirane. Primjeri jednostavnih aktivnosti su naredba `<invoke>`, naredba `<receive>` i naredba `<reply>` kojima se ostvaruje međudjelovanje sa suradnicima, te naredba `<assign>` kojom se pridružuju vrijednosti varijablama. Strukturirane aktivnosti definiraju redoslijed izvođenja skupa naredbi. Primjeri strukturiranih aktivnosti su naredba slijednog poretku aktivnosti `<sequence>`, naredba uvjetnog grananja `<switch>`, naredba petlje (`<while>`) i naredba paralelnog izvođenja `<flow>`.

BPEL definira mehanizam za upravljanje transakcijama i iznimkama koji je zasnovan na *WS-Coordination* i *WS-Transaction* specifikacijama. Skup aktivnosti grupira se u transakciju primjenom bloka `<scope>`. Blok `<scope>` grupira aktivnosti koje se sve moraju uspješno izvesti. Za svaki blok `<scope>` definira se kompenzacijска akcija koja se izvodi u slučaju pogreške unutar bloka. Upravljanje iznimkama usko je povezano s opisanim transakcijskim mehanizmom i sastoji se od naredbe `<throw>` za stvaranje iznimaka i naredbe `<catch>` za hvatanje iznimaka unutar bloka `<scope>`.

### 3.4.3 BPML

BPML (*Business Process Modeling Language*) je meta-jezik namijenjen oblikovanju poslovnih procesa. Jezik BPML inicijalno je definiran kako bi se opisali poslovni procesi koji se izvode na raznorodnim sustavima za upravljanje poslovnim procesima (engl. *Business Process Management System*). Već prva verzija jezika BPML uskladena je s WSCI normom za koreografiju mrežnih usluga. Tako je uporabom jezika BPML omogućeno definiranje izvodljivih poslovnih procesa, a WSCI jezikom se opisuje međudjelovanje i koreografija BPML procesa s drugim mrežnim uslugama.

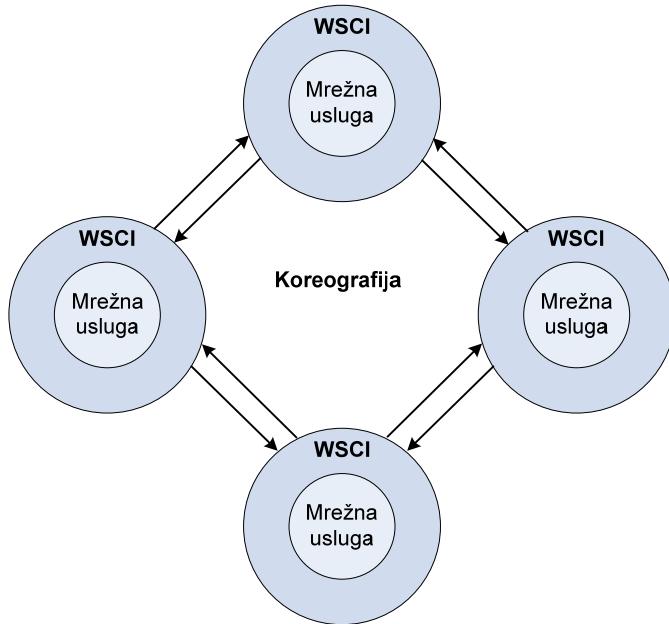
Jezik BPML zasnovan na XML-u definira apstraktни model izvođenja surađujućih i transakcijskih poslovnih procesa koji se zasniva na konceptu konačnih automata. Poslovni se proces u jeziku BPML oblikuje definiranjem tijeka izvođenja, toka podataka i tijeka događaja, uz dodatne mogućnosti ugradnje poslovnih pravila, sigurnosnih pravila i transakcijskog okruženja. Jezik BPML je oblikovan kao sredstvo objedinjavanja postojećih poslovnih programskih sustava u poslovni računalni sustav zasnovan na procesima. Kao dio potpore opisu i izvođenju poslovnih procesa BPML omogućuje oblikovanje sinkronih i asinkronih raspodijeljenih transakcija.

Struktura jezika BPML slična je jeziku BPEL4WS i sadrži slične konstrukte opisa tijeka izvođenja procesa i aktivnosti. BPML sadrži jednostavne aktivnosti sinkronih i asinkronih poziva mrežnih usluga. Ponuđene naredbe tijeka izvođenja procesa su uvjetno grananje i uvjetne petlje, te slijedno i paralelno izvođenje skupa aktivnosti. Nadalje, jezik BPML sadrži konstrukte za definiranje postojanosti podataka potrebne za ostvarenje dugotrajnih transakcija, definiranje suradnika i njihovih uloga u komunikaciji, te definiranje rekurzivne kompozicije podprocesa u složeniji proces. BPML pruža potporu za rukovanje iznimkama i potporu za transakcijske operacije. Ponuđene su kratkotrajne i dugotrajne transakcije. Kompenzacijnska pravila definiraju se sličnim tehnikama kao u jeziku BPEL4WS, definiranjem opsega transakcije i definiranjem kompenzacijskih akcija za opseg transakcije.

### 3.4.4 WSCI

Jezik WSCI (*Web Services Choreography Interface*) definira proširenje WSDL opisa sučelja mrežnih usluga. WSCI je jezik koreografije usluga koji opisuje pravila razmjene poruka između mrežnih usluga koje sudjeluju u zajedničkom međudjelovanju. Ključna značajka jezika WSCI je da se njime ne definira izvodljivi poslovni proces, već se definira samo javno međudjelovanje mrežnih usluga. Jedno WSCI sučelje opisuje sudjelovanje jednog suradnika u procesu razmjene poruka. Kao što je prikazano na slici 3.10, koreografija usluga definirana jezikom WSCI sastoji se od skupa WSCI sučelja, definiranih za svakog suradnika u

međudjelovanju. Jezik WSCI ne omogućuje definiranje jednog upravljačkog procesa koji upravlja međudjelovanjem usluga.



**Slika 3.10 Koreografija usluga primjenom WSCI jezika**

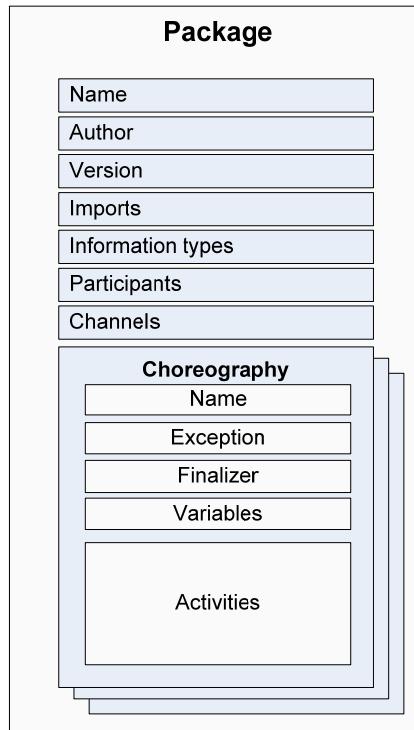
Tehnološki je jezik WSCI proširenje WSDL jezika. Operacije jezika WSDL definiraju pristupnu točku jedinici funkcionalnosti koju ostvaruje mrežna usluga, a akcije WSCI jezika određuju pravila međudjelovanja WSDL operacija. WSCI definira element `<action>` kojim se specificiraju osnovne ulazne i izlazne poruke akcija, WSDL operacije povezane s određenom akcijom, te uloge mrežnih usluga u međudjelovanju. Vanjske mrežne usluge pozivaju se primjenom elementa `<call>`. Skup aktivnosti povezuje se primjenom strukturiranih aktivnosti koje između ostalog omogućuju slijedno i paralelno izvođenje aktivnosti te definiranje uvjetnog grananja i uvjetnih petlji.

### 3.4.5 WS-CDL

WS-CDL (*Web Services Choreography Definition Language*) je normirani jezik za opis koreografije mrežnih usluga. Namjena WS-CDL jezika je definiranje ugovora koji opisuje javno vidljivo ponašanje mrežnih usluga i njihovih korisnika. Ponašanje mrežnih usluga opisano je na razini razmjene poruka.

WS-CDL opis koreografije sadržan je unutar dokumenta koji se naziva *package*. Dokument *package* sadrži skup aktivnosti koje izvode jedna ili više strana uključenih u koreografiju. Osnovna struktura dokumenta *package* prikazana je na slici 3.11. Osnovni elementi od kojih se sastoji dokument *package* su elementi *Name*, *Author* i *Version* kojima se određuje naziv složene usluge definirane koreografijom usluga, ime autora i verzija dokumenta *package*.

Nadalje, element *imports* povezuje koreografiju opisanu *package* dokumentom s drugim koreografijama koje su ostvarene jezikom WS-CDL. Element *Information types* određuje strukture poruka koje se razmjenjuju između usluga povezanih koreografijom. Elementom *Participants* definirane su mrežne usluge koje se povezuju koreografijom, a elementom *Channels* definiraju se simbolička imena za naslovljavanje usluga koje sudjeluju u koreografiji.

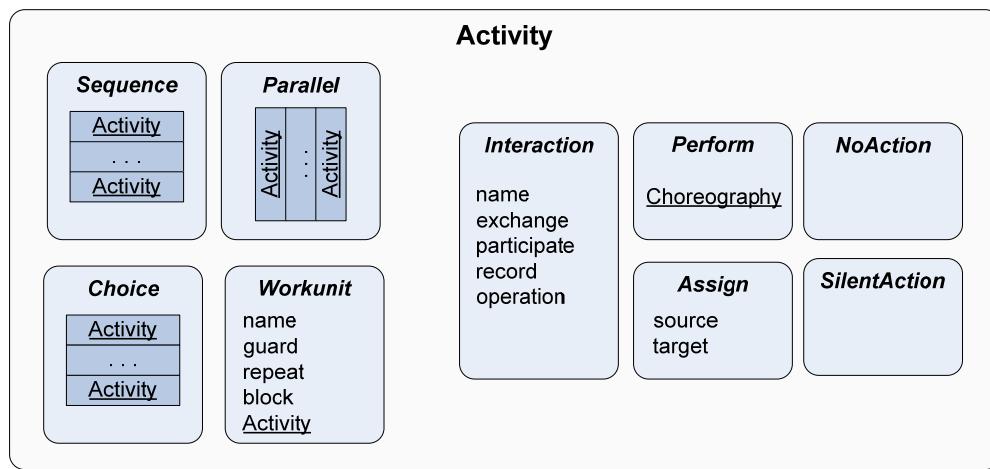


Slika 3.11 Struktura dokumenta *Package* jezika WS-CDL

Aktivnosti svakog sudionika u koreografiji opisane su elementom *Choreography*. Osnovna struktura elementa *Choreography* određena je elementima *Name*, *Exception*, *Finalizer*, *Variables* i *Activities*. Elementom *Name* određuje se naziv koreografije definirane za svakog sudionika. Pravila obrade iznimaka opisana su unutar elementa *Exception*. Element *Finalizer* definira aktivnosti koje se izvode nakon upješno završenog dijela koreografije opisanog roditeljskim elementom *Choreography*. Elementom *Variables* definiraju se varijable koje se koriste u aktivnostima navedenim unutar elementa *Activities*.

Element *Activities* sadrži naredbe kojima je opisana logika sudionika u koreografiji. Naredbe jezika WS-CDL, prikazane na slici 3.12, dijele se na naredbe za upravljanje tijekom izvođenja (engl. *control-flow*), naredbe definiranja jedinice rada (engl. *workunit*) i jednostavne naredbe (engl. *basic activities*). Naredbe upravljanja tijekom izvođenja sastoje se od naredbe *sequence* za slijedno izvođenje bloka naredbi, naredbe *parallel* za paralelno izvođenje bloka naredbi te naredbe *choice* za odabir jedne od mogućih naredbi. Naredba

*workunit* opisuje uvjetno ili višestruko izvođenje ugnježđenih naredbi. Uvjet izvođenja i uvjet ponavljanja definirani su kao logički izrazi. Uvjetom blokiranja moguće je definirati da li je prije izračunavanja uvjeta izvođenja i uvjeta ponavljanja potrebno čekati da varijable, sadržane u tim uvjetima, postanu dostupne. Jednostavne naredbe jezika WS-CDL su naredbe *Interaction*, *NoAction*, *SilentAction*, *Assign* i *Perform*. Naredbe *NoAction* i *SilentAction* označavaju točku u procesu koreografije u kojoj sudionik ne obavlja niti jednu aktivnost ili obavlja aktivnost koja ne utječe na globalni proces definiran koreografijom. Uporabom *Assign* naredbe dodjeljuje se vrijednost varijabli. Naredba *Perform* koristi se kako bi se pozvao drugi proces koreografije, odnosno omogućava hijerarhijsko oblikovanje procesa koreografije. Pozvani proces koreografije ne mora biti definiran unutar iste *Package* strukture.



Slika 3.12 Aktivnosti WS-CDL jezika

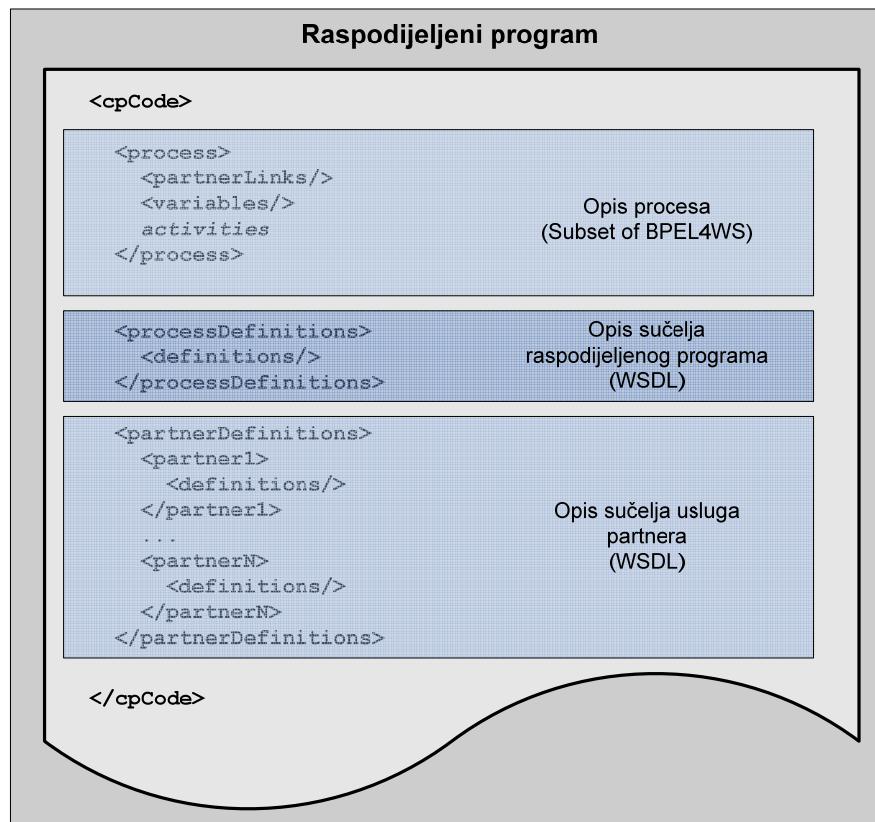
Središnja naredba koreografije opisane WS-CDL jezikom je naredba *Interaction* kojom se definira razmjena informacija između sudionika u koreografiji. *Interaction* naredba se koristi u tri svrhe: slanje zahtjeva, slanje odgovora ili slanje zahtjeva na koji se mora odgovoriti.

### 3.4.6 CL

CL (*Coopetition Language*) [77], [80] je jezik za oblikovanje raspodijeljenih programa. Raspodijeljeni programi su programske cjeline koje sadrže opis izvodljivog procesa složene mrežne usluge i opise sučelja svih mrežnih usluga suradnika. Tako definirani raspodijeljeni programi su potpuno definirane programske cjeline i sadrže sve informacije potrebne za njihovo izvođenje. Jezik CL definiran je u skladu sa normiranim jezicima BPEL4WS i WSDL.

Općenita struktura raspodijeljenog programa napisanog u jeziku CL prikazana je na slici 3.13. Raspodijeljeni program zapisan je unutar XML dokumenta koji se sastoji od opisa

procesa, opisa sučelja raspodijeljenog programa i skupa opisa sučelja mrežnih usluga suradnika definiranog procesa.



Slika 3.13 Struktura raspodijeljenog programa

Korijenski XML element zapisa raspodijeljenog programa je element `<cpCode>` koji se sastoji od tri podelementa unutar kojih se nalaze navedene cjeline. Opis procesa raspodijeljenog programa definiran je primjenom jezika BPEL4WS i nalazi se unutar elementa `<process>`. Sučelje raspodijeljenog programa definirano je WSDL jezikom i nalazi se unutar elementa `<processDefinitions>`. Opis sučelja sadrži definiciju operacija raspodijeljenog programa koje pozivaju mrežne usluge suradnici i opis poruka koje se tom prilikom razmjenjuju. Normiranim opisom sučelja raspodijeljenog programa omogućeno je međudjelovanje raspodijeljenog programa sa svim mrežnim uslugama koje su izgrađene u skladu sa WS normama. Skup opisa WSDL sučelja svih mrežnih usluga suradnika definiranog procesa naveden je unutar `<partnerDefinitions>` elementa.

Logika procesa definira se podskupom naredbi jezika BPEL4WS. Definirani podskup jezika BPEL4WS sadrži tri osnovna bloka: `partnerLinks`, `variables` i `activities`. Blok `partnerLinks` sadrži definiciju kazaljki na vanjske servise s kojima raspodijeljeni program komunicira. Blok `variables` sadrži deklaraciju varijabli za zapis stanja raspodijeljenog programa. Blok `activities` sadrži naredbe BPEL4WS aktivnosti kojima se definira logika procesa raspodijeljenog programa.

**Tablica 3-1 Podskup naredbi jezika BPEL4WS obuhvaćen specifikacijom CL jezika**

<b>Naredbe jezika CL</b>	<b>Opis naredbe jezika CL</b>
<variable/>	Naredba za deklaraciju varijabli u kojima raspodijeljeni program sprema stanje tijekom izvođenja
<partnerLink/>	Naredba za deklaraciju simboličkih imena mrežnih usluga koje se pozivaju iz raspodijeljenog programa
<sequence> activities+ </sequence>	Naredba za definiranje slijednog izvođenja skupa naredbi
<while/>	Naredba za ostvarivanje petlje s uvjetom ponavljanja
<invoke/>	Naredba za ostvarivanje sinkronog poziva udaljenih mrežnih usluga
<receive/>	Naredba za primanje poruke zahtjeva poslane tijekom poziva operacije sučelja raspodijeljenog programa
<reply/>	Naredba za proslijedivanje poruke odgovora za prethodno primljenu poruku zahtjeva
<switch> <case1/> ... <caseN/> </otherwise> </switch>	Naredba za ostvarivanje uvjetnog grananja tijeka izvođenja raspodijeljenog programa
<pick> <onMessage1/> ... <onMessageN/> <onTimer/> </pick>	Naredba za obradu poruka zahtjeva za dvije ili više operacija pristupnog sučelja raspodijeljenog programa
<assign/>	Naredba za dodjelu vrijednosti varijablama
<wait/>	Naredba za ostvarivanje čekanja
<terminate/>	Naredba za zaustavljanje tijeka izvođenja
<empty/>	Naredba bez akcije

Jezik CL sadrži samo dio naredbi definiranih BPEL4WS jezikom. Podskup BPEL4WS naredbi ponuđenih u jeziku CL naveden je u tablici 3.1. Odabrani podskup BPEL4WS naredbi pojednostavljuje strukturu raspodijeljenog programa napisanog CL jezikom. Osim toga, smanjenje skupa naredbi omogućuje jednostavno i učinkovito interpretiranje raspodijeljenih programa.

Opisani jezik CL objedinjuje dobre osobine koncepata koreografije i orkestracije za definiciju procesa kompozicije mrežnih usluga. Raspodijeljeni program opisan jezikom CL definira proces razmjene poruka jednog raspodijeljenog programa sa skupom mrežnih usluga, što odgovara konceptu koreografije. Nadalje, jezik CL sadrži konstrukte za manipulaciju

podacima kojima je omogućeno ostvarenje dugotrajnih međudjelovanja koja imaju svojstvo očuvanja stanja, što je tipično za koncept orkestracije.

Ostvareni rasподijeljeni programi su mrežne usluge sa svojstvom očuvanja stanja (engl. *stateful web services*) i mogu imati više autonomnih primjeraka koji se izvode istovremeno. Stvaranje primjeraka i upravljanje životnim vijekom primjeraka rasподijeljenog programa ostvareno je u skladu s WSRF [61] specifikacijom. Primjeri rasподijeljenih programa imaju jedinstveni pokazatelj, koji omogućuje jednoznačnu komunikaciju pojedinog primjerka s uslugama suradnicima. Adresiranje primjeraka rasподijeljenog programa i usluga suradnika ostvareno je u skladu s normom *WS-Addressing* [21].

## 4 Razvoj prilagođen krajnjim korisnicima

Korisnike računala moguće je podijeliti u više razreda ovisno o vrsti posla kojeg obavljaju primjenom računala i o znanjima koja posjeduju o računalnim sustavima. Dva su razreda korisnika računalnih sustava na suprotnim krajevima navedene podjele, profesionalni programeri i krajnji korisnici računalnih sustava. Profesionalni programeri su osobe koje posjeduju veliku količinu znanja o računalnim sustavima, tehnikama oblikovanja programskih sustava i programskim jezicima. S druge strane, krajnji korisnici (engl. *end-users*) računalnih sustava su osobe koje ne posjeduju osnovna znanja o načinu rada računalnih sustava koje koriste, već računala koriste za izvođenje specifičnih primjenskih programa koji im pomažu u svakodnevnom radu. Krajnji korisnici su osobe koje računala koriste u slobodno vrijeme, ali i stručnjaci iz određenih znanstvenih ili poslovnih područja koji računala koriste za obavljanje poslovnih zadataka.

Područje računarstva koje istražuje metode krajnjim korisnicima prilagođenog razvoja računalnih sustava naziva se *razvoj prilagođen krajnjem korisniku* (engl. *End User Development*). Unatoč nekim postignućima u području razvoja prilagođenog krajnjim korisnicima, od trenutka kada je koncept stvoren u ranim 80-tima, proizvodi koji krajnjim korisnicima omogućuju razvoj novih programske sustava nisu široko rasprostranjeni, već su često samo dodatak komercijalnim proizvodima koji im omogućuju određene izmjene i prilagodbu funkcionalnosti primjenskih programa.

Većina korisnika računala upoznata je s osnovnim funkcionalnostima računala i sučeljima putem kojih se te funkcionalnosti koriste. Svakodnevnim unapređivanjem korisničkih sučelja ostvaruje se napredak u jednostavnosti korištenja interaktivnih programskih sustava te se krajnjim korisnicima olakšava izvođenje željenih zadataka na računalu. Unatoč napretku u načinu korištenja postojećih programskih sustava, malo je pomaka ostvareno u načinu izgradnje novih primjenskih programskih sustava za izvođenje specifičnih zadataka krajnjih korisnika. Izgradnja novih programskih sustava, kako složenih tako i onih vrlo jednostavnih, zahtijeva određenu količinu znanja o oblikovanju programskih sustava i poznavanje programskih jezika što se ne očekuje od krajnjih korisnika računalnih sustava. Upravo zbog toga je jedan od problema računalne znanosti razvoj okolina koje će omogućiti krajnjim korisnicima izgradnju vlastitih programskih sustava ili prilagođavanje postojećih programskih sustava bez potrebe za prethodnim stjecanjem znanja o tehnikama programiranja.

Unapređenjem postojećih i razvojem novih razvojnih okolina omogućit će se evolucija interaktivnih programskih sustava iz *jednostavnih za korištenje* (engl. *easy-to-use*) prema *jednostavnim za razvoj* (engl. *easy-to-develop*). Prema predviđanjima iznesenim u radu [68]

već 2005. godine broj neprofesionalnih programera u svijetu će za red veličine prerasti broj profesionalnih programera.

Motivacija za unapređenje razvoja prilagođenog krajnjim korisnicima je stalni rast tehnološkog razvoja koji je podložan *Moore-ovom zakonu*. Kapaciteti spremničkog prostora, brzina procesorskih jedinica, propusnost mrežnih kanala i druga svojstva računalnih sustava razvijaju se u skladu s *Moore-ovim zakonom*, odnosno imaju svojstva eksponencijalnog rasta kroz vrijeme. Tako se, na primjer, povećanjem spremničkog prostora i propusnosti mrežnih kanala omogućuje dohvaćanje i spremanje velike količine raznih sadržaja čak i na razne ručne uređaje poput mobilnih telefona i ručnih računala. Porastom dostupne količine sadržaja dolazi do pojave preopterećenja korisnika sadržajem (engl. *content overload*). Ta negativna pojava dovodi do toga da korisnik više vremena provodi pretražujući dostupne sadržaje nego što koristi željene sadržaje. Kako bi se izbjegla navedena pojava razvijaju se nove metode pristupa sadržajima poput raznih sustava dohvata upitima (engl. *query management*) i drugih interaktivnih sustava kojima je zajedničko posjedovanje određenog stupnja inteligencije. Unatoč napretku u navedenim područjima, želju korisnika da potpuno nadzire ponašanje sustava moguće je ostvariti samo ako krajnji korisnik ima mogućnost definiranja ponašanja, odnosno programiranja, interaktivne okoline kojom pristupa sadržaju. Ovo je vrlo bitno za proizvode koji se plasiraju na masovna tržišta, poput tržišta osobnih računala i mobilnih telefona, čiji korisnici ne posjeduju znanja o tehnikama programiranja. *Razvoj prilagođen krajnjim korisnicima* bi trebao omogućiti krajnjim korisnicima da interaktivne programske sustave takvih uređaja prilagode vlastitim potrebama, te po potrebi definiraju nove funkcionalnosti.

## 4.1 Definicija razvoja prilagođenog krajnjem korisniku

Razvoj prilagođen krajnjem korisniku podrazumijeva aktivno sudjelovanje krajnjeg korisnika u procesu razvoja programskega sustava. Na taj se način zadatke koje tradicionalno obavljaju profesionalni programeri preuzimaju krajnji korisnici sustava. Vrste zadataka koje aktivno rješavaju krajnji korisnici mogu biti u rasponu od definiranja zahtjeva svojstava programskega sustava, definiranja načina uporabe sustava i djelomičnog oblikovanja sustava, do programiranja samog sustava. Razvoj prilagođen krajnjem korisniku mora korisniku pružiti mogućnost stvaranja i izmjene programskih sustava tijekom uporabe sustava.

Jedna od definicija razvoja prilagođenog krajnjem korisniku je dana u [71]: "Razvoj prilagođen krajnjem korisniku je skup metoda, tehnika i alata koji omogućavaju korisnicima programskih sustava, koji se ponašaju kao neprofesionalni razvijatelji programskih sustava, neki oblik izgradnje ili promjene postojećih programskih produkata". Prema navedenoj

definiciji osnovne vrste aktivnosti koje obavljaju krajnji korisnici u okruženju razvoja prilagođenog krajnjim korisnicima su prilagodba i izmjena gotovih sustava.

Aktivnosti prilagodbe omogućavaju korisniku da promjenom odgovarajućih parametara odabere jednu od dostupnih vrsta ponašanja sustava. Takve aktivnosti obuhvaćene su pojmovima parametrizacije (engl. *parameterisation*), prilagođavanja (engl. *customisation*) i poosobljavanja (engl. *personalization*). Osim *parametrizacije*, primjer aktivnosti koja pripada ovoj skupini je i obilježavanje (engl. *annotation*). Obilježavanjem korisnici zapisuju bilješke vezane uz podatke i rezultate kako bi zapamtili koje operacije su nad podacima izveli i kako su došli do dobivenih rezultata, s ciljem uspješnog ponavljanja obavljenih aktivnosti.

Aktivnosti izmijene podrazumijevaju korištenje poznatih programskih paradigmi za uspostavu novih funkcionalnosti sustava. Primjeri programskih paradigmi koje se koriste su programiranje primjenom demonstracije i primjera, te grafičko programiranje, makro programiranje i skriptni jezici. Ova skupina aktivnosti više odgovara definiciji razvoja prilagođenog krajnjim korisnicima jer, za razliku od aktivnosti prilagodbe, omogućava izmijene funkcionalnosti samog programskega sustava. Ovu skupinu aktivnosti moguće je dalje podijeliti na objedinjavanje i proširenje (engl. *integration and extension*) [69].

*Objedinjavanje* označava aktivnosti koje prelaze granice prilagođavanja postojećih primjenskih programa te krajnjim korisnicima omogućava dodavanje novih funkcionalnosti postojećim primjenskim programima. Dodavanje novih funkcionalnosti obavlja se povezivanjem unaprijed definiranih komponenti primjenskog programa, ali bez poznavanja programskega kôda kojim je primjenski program ostvaren.

*Proširenja* se primjenjuju na programske sustave koji ne ostvaruju funkcionalnosti potrebne za obavljanje cijelokupnih korisničkih zahtjeva, niti sadrže komponente koje obavljaju dijelove zadataka. U takvim slučajevima potrebno je programski sustav proširiti novim funkcionalnostima, a taj postupak je složen i zahtjeva izmijene u programskom kodu.

## 4.2 Aspekti krajnjem korisniku prilagođenog razvoja programskih sustava

Razvoj prilagođen krajnjim korisnicima promatra se kroz aspekte različitih područja programskega inženjerstva. Glavni aspekti procesa razvoja programskih sustava su programske paradigme i jezici, razvojna okruženja i metode programiranja, međudjelovanje korisnik-računalo, područje primjene te organizacijski i socijalni aspekti.

### **Programske paradigme i jezici**

Različiti pristupi programiranju imaju značajan utjecaj na složenost procesa kojim krajnji korisnici pristupaju i koriste programski sustav. Postoje mnogi formalizmi u tehnikama programiranja koji su razvijeni za rješavanje problema u određenim područjima računarstva. Većinu formalizama oblikovale su i koristile iste skupine ljudi, što je često za posljedicu imalo složenost razvijenih formalizama i mogućnost njihovog korištenja samo od strane osoba koje su posjedovale znanja iz područja računarske znanosti. Svi su formalizmi zasnovani na nekoj programskoj paradigmi koja određuje model interpretiranja programa i njihovog izvođenja na računalu. Primjeri razvijenih programskih paradigmi su logičko programiranje, funkcionalno programiranje, objektno-orientirano programiranje, paralelno programiranje, tablični proračuni (engl. *spread-sheet*) i skriptni jezici. Neke od navedenih paradigmi su tijekom oblikovanja prilagođene krajnjim korisnicima, npr. tablični proračuni, dok su druge paradigme presložene i nerazumljive krajnjim korisnicima, te ih je potrebno na neki način prilagoditi njihovim mogućnostima.

### **Razvojna okruženja i metode programiranja**

Razvojna okruženja i metode programiranja potrebno je prilagoditi sposobnostima korisnika koji imaju mala znanja iz područja tehnika programiranja kako bi se i takvim korisnicima omogućilo razvijanje novih programskih produkata. Razvojna okruženja moraju korisnicima pružiti razumljivu potporu za obavljanje željenih zadataka te učinkovito predstavljanje i obradu objekata potrebnih za tu svrhu. Dodatno, krajnjim korisnicima je potrebno pružiti mogućnosti objedinjavanja dokumentiranja s procesom razvoja programskog sustava.

### **Međudjelovanje korisnik - računalo**

Nove tehnike predstavljanja i analize velike količine informacija, razvijene u području međudjelovanja čovjek-računalo (engl. *Human-Computer Interaction*), mogu poslužiti za stvaranje razvojnih okruženja jednostavnih za korištenje.

### **Područje primjena**

Svako znanstveno ili poslovno područje posjeduje vlastiti jezik opisa problema i vlastiti skup osnovnih koncepata na kojima se zasnivaju problemi iz tog područja. Stručnjacima iz određenih područja potrebno je osigurati okruženja u kojima će moći izravno manipulirati s takvim konceptima kroz skup njima poznatih apstrakcija bez potrebe za učenjem posebnih programskih jezika.

### **Organizacijski i socijalni aspekti**

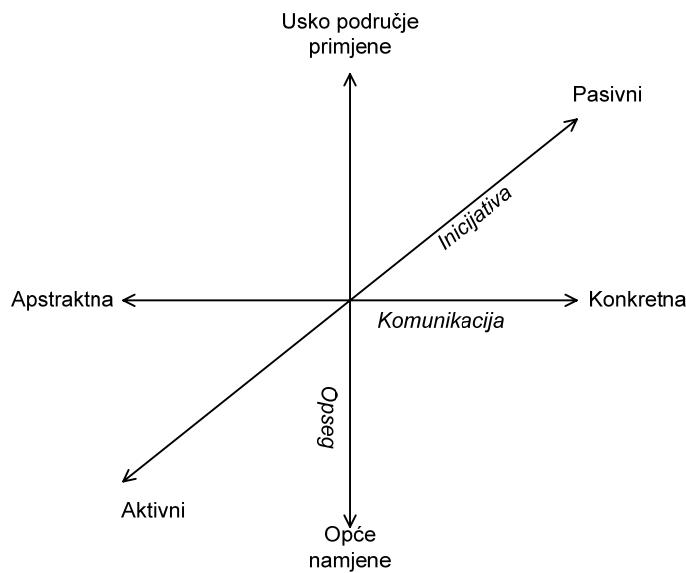
Okruženje u kojem korisnici obavljaju svoje poslove znatno utječe na način organizacije razvojnog procesa. Krajnji su korisnici programirljivih i prilagodljivih sustava najčešće

organizirani u mreže pomoći i hijerarhijske organizacije. Cilj takvih organizacija je omogućiti pomoć onima koji nisu programeri da mijenjaju primjenske programe koje koriste ili da stvaraju nove. Uobičajena posrednička uloga u takvim organizacijama zasnovana je na *naprednim korisnicima* (engl. *super-users*). To su stručnjaci iz područja primjene programskih sustava koji posjeduju određena znanja o tehnikama programiranja i čija je zadaća pomoći ostalim članovima organizacije u obavljanju njihovih zadataka.

### 4.3 Dimenzije sustava prilagođenih krajnjem korisniku

Razredba sustava prilagođenih krajnjem korisniku zasniva se na definiranju tri osnovne dimenzije sustava [70], opsega ili područja primjene sustava, načina komunikacije korisnika i sustava, te inicijative sustava. Navedene dimenzije razredbe sustava i rasponi vrijednosti dimenzija prikazani su na slici 4.1.

Razvojna okruženja namijenjena krajnjim korisnicima mogu se po pitanju opsega ili područja primjene opisati u granicama specijaliziranih i općenitih sustava. Neki sustavi su razvijeni s ciljem korištenja u uskom, specijaliziranom području primjene i namijenjeni su samo stručnjacima iz tog područja. Primjer takvih sustava su programirljivi upiti nad bazama koje sadrže podatke o proteinским strukturama. Takvi sustavi se nazivaju *sustavi prilagođeni zadacima* (engl. *task-specific systems*) ili *sustavi prilagođeni području primjene* (engl. *domain-specific systems*). Nasuprot takvim sustavima su sustavi namijenjeni krajnjim korisnicima koji su oblikovani kao alati opće namjene koji se mogu primijeniti za rješavanje širokog skupa problema.



Slika 4.1 Dimenzije sustava prilagođenih krajnjem korisniku

Druga dimenzija sustava prilagođenih krajnjem korisniku odnosi se na način komunikacije korisnika i sustava. Ova dimenzija opisuje se u rasponu od apstraktne komunikacije uporabom formalnog jezika do konkretne komunikacije uporabom prirodnog jezika. Apstraktna komunikacija najčešće se ostvaruje formalno definiranim simboličkim tekstualnim jezicima ili dijagramske jezicima. Osnovne značajke formalnih jezika su dobro definirana sintaksa i semantika jezika kojima je smanjena pojava nejednoznačnosti jezika. Nedostatak formalnih jezika je da krajnjim korisnicima nisu unaprijed poznati. Zbog toga korisnici moraju formalne jezike najprije naučiti kako bi mogli koristiti sustav. S druge strane, prirodni jezici su korisnicima unaprijed poznati, te ih korisnik ne mora učiti samo zbog potrebe komunikacije sa sustavom. Nedostatak prirodnih jezika je da nemaju formalno definirana sintaksna i semantička pravila što rezultira nejednoznačnostima jezika, te poteškoćama i pogreškama u interpretiranju jezika. Glavni kompromis koji se nameće prilikom odluke o jeziku komunikacije korisnika i sustava je između potrebe učenja formalnih jezika i mogućnosti pojave pogrešaka u interpretaciji prirodnih jezika.

*Inicijativa* sustava definira treću dimenziju razvoja prilagođenog krajnjim korisnicima. Sustav u potpunosti prepušta inicijativu korisniku i samo definira značenje instrukcija koje korisnik zadaje sustavu kako bi riješio željene zadatke. Takvi sustavi se nazivaju pasivni sustavi. Nasuprot njima su aktivni sustavi, inteligentni sustavi koji uče o korisnikovim željama na osnovi pokaznih akcija ili praćenjem rada korisnika, te nakon toga preuzimaju inicijativu i obavljaju potrebne akcije za ostvarenje korisnikovog zadatka. Između granica aktivnih i pasivnih sustava nalaze se razvojni sustavi koji korisniku omogućuju odabir jedne od ponuđenih akcija. Pri tome sustav nudi korisniku samo one akcije koje su u danom trenutku očekivane i prikladne.

Navedene dimenzije nisu u potpunosti ortogonalne. Na primjer, prirodna komunikacija korištenjem hrvatskog jezika podrazumijeva određenu inicijativu od strane sustava kojom on interpretira i razrješava nejednoznačnosti korisničkih instrukcija.

## 4.4 Programiranje prilagođeno krajnjim korisnicima

Jedna od grana područja razvoja prilagođenog krajnjem korisniku vezana je uz tehnike programiranja i programske jezike prilagođene krajnjim korisnicima i naziva se programiranje prilagođeno krajnjim korisnicima (engl. *end-user programming*).

### 4.4.1 Značajke programiranja prilagođenog krajnjim korisnicima

Tehnike programiranja i programske jezike namijenjeni krajnjim korisnicima sadrže neke osnovne značajke po kojima se razlikuju od tradicionalnih tehnika programiranja i

programskega jezika opće namjene. Osnovno svojstvo koje se želi postići za oblikovanje programskega jezika namijenjenih krajnjem korisniku je kratko vrijeme učenja jezika. Vrijeme učenja skraćuje se prilagođavanjem razvojnih okruženja okolinama u kojima korisnici rješavaju zadatke u stvarnom svijetu, te definiranjem tehnika programiranja sličnih načinima rješavanja problema u stvarnom svijetu. Tradicionalni programski jezici imaju strogo definirana leksička, sintaksna i semantička pravila prema kojima je potrebno opisati problem koji se želi riješiti. Za razliku od tradicionalnih programskih jezika koji su uglavnom tekstualni jezici, jezici namijenjeni krajnjim korisnicima često se oblikuju kao grafički jezici.

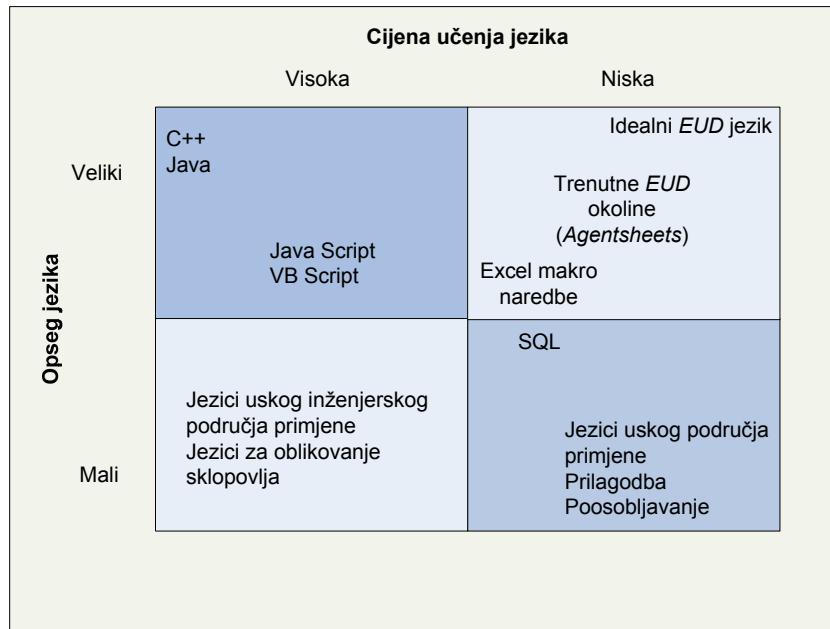
Tehnike programiranja koje koriste krajnji korisnici najčešće se oslanjaju na postojanje većeg broja programskih komponenti i na njihovu uporabu prilikom oblikovanja željene funkcionalnosti programa. U ovom okruženju, programska komponenta se odnosi na bilo koju programsku ostvarenu funkcionalnost koja se koristi za izgradnju novog programskega sustava. U okruženju skriptnih jezika programska komponenta je objekt nekog razreda čije je ostvarenje sadržano u nekoj od standardnih biblioteka komponenata. U okruženju definiranja formula u MS Excel programskom paketu, komponenta je bilo koja matematička funkcija koja se poziva tijekom pisanja formule.

### *Trošak učenja programskog jezika*

Tijekom oblikovanja jezika za komunikaciju korisnika i računala najteže je riješiti proturječnost između složenosti programskega jezika i njegove izražajnosti. Složenost jezika određena je utroškom vremena potrebnog za učenje jezika, dok je izražajnost ili opseg jezika određena skupom akcija koje je moguće izraziti naredbama jezika. Složeniji jezici mogu poslužiti za rješavanje velikog broja različitih problema, ali podrazumijevaju povećani trud i veći utrošak vremena i novca tijekom učenja jezika. Odnos opsega jezika i cijene učenja jezika koriste se za definiranje odnosa tradicionalnih programskih jezika opće namjene, programskih jezika namijenjenih krajnjem korisniku i jezika za prilagođavanje i izmjeni postojećih primjenskih programa. Navedeni odnosi prikazani su na slici 4.2.

Tradicionalni programski jezici opće namjene, poput programskih jezika C++ i Java nalaze se u kvadrantu koji sadrži jezike velikog opsega, ali i visoke cijene učenja. Jezike ove skupine koristi vrlo mali udio krajnjih korisnika, uglavnom visoko motiviranih korisnika koji imaju sposobnosti i potrebu učenja programskih jezika zbog rješavanja specifičnih znanstvenih problema. Na granici prema kvadrantu "veliki opseg - niska cijena" nalazi se većina jezika koji se smatraju programskim jezicima namijenjenim krajnjem korisniku. To su uglavnom skriptni jezici, koji su nastali pojednostavljivanjem programskih jezika opće namjene. Kvadrant "mali opseg - visoka cijena učenja" sadrži programske jezike uskog područja primjene namijenjenih rješavanju problema u složenim inženjerskim područjima poput

oblikovanja sklopovskih sustava. Ovi jezici podrazumijevaju visoku cijenu učenja, ali jednom naučeni postaju učinkovitiji za rješavanje specijaliziranih problema od programskega jezika opće namjene.



**Slika 4.2 Odnos cijene i opsega jezika prilagođenih krajnjem korisniku**

U kvadrantu "mali opseg - niska cijena" sadržani su jezici jednostavniji za učenje krajnjim korisnicima, ali ograničenog uskog područja primjene. Ova skupina jezika predstavlja granicu između programskih jezika namijenjenih krajnjem korisniku te prilagodbe i poosobljavanja gotovih primjenskih programa pa je programiranje svedeno na unos parametara u za to predviđene obrasce. Na granici prema većem opsegu nalaze se makro jezici koji su namijenjeni proširenju uredskih primjenskih programa. Primjeri takvih jezika su formule u MS Excel programskom paketu i jezici za upite nad bazama podataka.

Kvadrant "veliki opseg - niska cijena" definira idealne značajke jezika koje se teži postići tijekom oblikovanja jezika namijenjenih krajnjem korisniku. Trenutno najprilagođeniji jezici krajnjem korisniku su grafički jezici koji omogućuju stvaranje programirljivih agenata. Navedeni jezici i dalje određena ograničenja koja se odnose na potrebu učenja programskega jezika za zadavanje pravila ponašanja agentima u obliku *uvjet-akcija* pravila. Primjer razvojne okoline prilagođene krajnjem korisniku koja koristi agente je sustav *AgentSheets* [66]. Daljnje smanjenje troškova učenja moguće je postići programiranjem ponašanja agenata primjenom primjera.

### *Grafički i tekstualni programske jezici*

Jedna od podjela programskih jezika namijenjenih krajnjim korisnicima je prema načinu predstavljanja programa i načinu definiranja naredbi programa. Prema toj podjeli, programski jezici namijenjeni krajnjem korisniku dijele se na grafičke programske jezike (engl. *Visual Programming Languages - VPL*) i tekstualne programske jezike. Za razliku od većine tekstualnih programskih jezika opće namjene čija su sintaksna pravila definirana tekstualnim izrazima u obliku gramatike, sintaksna pravila grafičkih jezika određena su grafičkim izrazima.

Grafičko programiranje je način programiranja kojim se semantika izražava u više dimenzija. Primjeri dodatnih dimenzija su uporaba višedimenzionalnih grafičkih objekata, uporaba njihovih prostornih odnosa ili uporaba vremenske dimenzije za definiranje "prije-poslije" semantičkih odnosa. Definirani objekt ili relacija između objekata predstavlja najjednostavniju cjelinu jezika, istoznačnu leksičkoj jedinki tekstualnih programskih jezika. Povezivanjem više cjelina sukladno pravilima grafičkog jezika definira se grafički izraz. Primjeri grafičkih izraza su dijagrami, skice, ikone ili primjer akcija obavljenih nad grafičkim objektima. Primjeri grafičkih programskih jezika namijenjenih krajnjem korisniku su jezici tabličnih proračuna (engl. *spreadsheets*) i programiranje demonstracijom (engl. *programming by demonstration*), dok su predstavnici tekstualnih programskih jezika skriptni jezici.

### *Programiranje zasnovano na komponentama*

Programske komponente i razvoj zasnovan na programskim komponentama imaju veliki značaj u programskom inženjerstvu. Programske komponente omogućuju oblikovanje programskih sustava povezivanjem ponovno iskoristivih (engl. *reusable*) programskih modula, za razliku od izgradnje programskih sustava definiranjem cjelokupne primjenske logike sustava naredbama programskih jezika opće namjene. Programiranje zasnovano na komponentama zahtijeva samo definiranje programskog kôda kojim se opisuje veza između komponenata.

Unatoč jednostavnoj paradigmi programiranja zasnovanog na komponentama, objedinjavanje komponenata u nove programske sustave od strane krajnjih korisnika nije jednostavan proces. Za potrebe krajnjih korisnika potrebno je izgraditi učinkovite i interaktivne okoline koje predstavljaju komponente na visokoj razini apstrakcije. Takve okoline moraju korisnicima omogućiti brz i jednostavan razvoj novih programskih sustava, prilikom kojeg korisnici definiraju funkcionalnost sustava na visokim razinama, bez potrebe za poznavanjem tehničkih detalja. Razvojna okolina mora krajnjim korisnicima pružiti mogućnosti grafičkog oblikovanja i razvoja, a sav potreban programski kôd generirati

automatski na osnovi grafičke reprezentacije oblikovanog sustava. Jedan od preuvjeta izgradnje opisanih razvojnih okolina je razvoj grafičkih programskih jezika.

Osnovna prepreka krajnjim korisnicima tijekom izgradnje programskih sustava povezivanjem komponenti je preuvjet poznavanja programskog sučelja komponenti. Korisnici moraju znati koje su operacije definirane sučeljima različitih komponenti i kako se one moraju pozvati da bi se komponente povezale u sustav. Osnovni model objedinjavanja programskih komponenti zasniva se na definiranju jedinstvenih, normiranih sučelja svih komponenti sustava. Tako definirana sučelja osiguravaju mogućnost povezivanja različitih komponenti i visok stupanj prilagodljivosti procesa povezivanja komponenti. Negativna posljedica definiranja općenitih sučelja je nastanak općenitih naziva operacija sučelja. Općeniti nazivi operacija sučelja za povezivanje komponenti nemaju jednoznačna i krajnjim korisnicima razumljiva značenja u različitim područjima primjene.

#### 4.4.2 Primjeri programskih jezika namijenjenih krajnjim korisnicima

Tri opće prihvaćene skupine programskih jezika namijenjenih krajnjim korisnicima su jezici tabličnih proračuna, programiranje demonstracijom i skriptni jezici.

##### *Jezici tabličnih proračuna*

Jezici tabličnih proračuna (engl. *spreadsheet languages*) pripadaju najrasprostranjenijim programskim jezicima prilagođenim krajnjim korisnicima. Krajnji korisnici koriste jezike tabličnih proračuna za obradu i prezentaciju finansijskih podataka i drugih tipova tabličnih podataka. Kako je osnovna namjena ovih jezika obrada finansijskih podataka, oni se smatraju specijaliziranim programskim jezicima.

Osnovna paradigma jezika tabličnih proračuna je jednostavna, program se sastoji od pravokutnih ćelija (engl. *cells*), koje sadrže vrijednosti i formule. Formule određuju vrijednosti ćelija na osnovi konstanti sadržanih u formuli, ugrađenih funkcija i vrijednosti drugih ćelija. Prema opisanoj paradigmi korisnik oblikuje programe navođenjem formula unutar pojedinih ćelija. Nakon što korisnik promijeni vrijednost neke ćelije, formule se automatski izračunavaju i izračunati rezultat prikazuje se u ćeliji.

Programska paradigma tabličnih proračuna predmet je istraživanja kojima je cilj proširiti njenu uporabu u drugim područjima primjene, kao što su obrada matrica, vizualizacija rezultata znanstvenih istraživanja, te eventualno stvoriti programski jezik opće namjene zasnovan na jezicima tabličnih proračuna. Nekoliko je razloga zašto su jezici tabličnih proračuna zanimljivi kao programska paradigma. Sustavi tabličnih proračuna su vrlo rašireni i korisnici ne razmišljaju o jezicima tabličnih proračuna kao o programskim jezicima već ih

smatraju prirodnim načinom korištenja računala u rješavanju svakodnevnih poslovnih zadataka. Nadalje, mnogo programa napisanih jezicima tabličnih proračuna pokazuju visoki stupanj paralelnosti. Paralelnost se najviše očituje istovremenim izračunavanjem vrijednosti celija.

Rezultati dosadašnjih istraživanja na području jezika tabličnih proračuna pokazuju da se nije znatno napredovalo u poopćavanju spomenutog razreda jezika. Neki razvijeni jezici [73] posjeduju svojstva klasičnih proceduralnih jezika s grafičkim sučeljem sustava tabličnih proračuna, no implicitni paralelizam je uglavnom izgubljen. Neznatan uspjeh u prethodnim istraživanjima posljedica je nekih osnovnih svojstava jezika tabličnih proračuna koja se razlikuju od svojstava programskih jezika opće namjene.

Najznačajnije svojstvo jezika tabličnih proračuna je geometrijska priroda tablica koja je u suprotnosti s tekstualnim osobinama većine modernih programskih jezika opće namjene. U tabličnim proračunima ćelije pojedine tablice naslovljavaju podatke iz drugih ćelija primjenom relativnog pomaka u odnosu na svoju poziciju u tablici. Takav način naslovljavanja je nepoznat u većini raširenih programskih jezika opće namjene i jedino ima smisla u okruženju adresiranja elemenata polja. No takvo adresiranje elemenata polja nije rašireno i rijetki programski jezici ga ostvaruju. U slučajevima drugih podatkovnih struktura relativno adresiranje bi zahtijevalo od programera poznavanje načina smještaja podatkovnih struktura u memoriji, a upravo je jedan od glavnih ciljeva viših programskih jezika sakrivanje organizacije memorije od korisnika.

Interaktivna priroda jezika tabličnih proračuna je sljedeća bitna razlika u odnosu na programske jezike opće namjene. Prilikom izmjene vrijednosti bilo koje ćelije ponovno se izračunavaju vrijednosti svih ćelija čija vrijednost ovisi o vrijednosti promijenjene ćelije i rezultati se prikazuju putem grafičkog sučelja. Održavanje i prikazivanje trenutne vrijednosti svih ćelija jedno je od glavnih svojstava sustava tabličnih proračuna. Nasuprot tome, programski jezici opće namjene prikazuju samo krajnji rezultat.

Zbog geometrijske i interaktivne prirode jezika tabličnih proračuna koja je izravna posljedica grafičkog korisničkog sučelja teško je smatrati da će se jezici tabličnih proračuna razviti i postati programski jezici opće namjene. Uporaba jezika tabličnih proračuna kao programskih jezika namijenjenih krajnjim korisnicima ostat će ograničena na primjenu u obavljanju financijskih i sličnih zadataka koji se svode na obradu tabličnih podataka.

### *Programiranje demonstracijom*

Programiranje demonstracijom zasniva se na dva načela. Prvo načelo je *programiranje zasnovano na primjerima* (engl. *example-based programming*), tehniku programiranja koja koristi primjere programa tijekom razvoja programskog sustava. Prednost ovog načina

programiranja je što krajnjim korisnicima omogućuje korištenje konkretnih primjera za oblikovanje programa u odnosu na korištenje apstraktnih konstrukata programskih jezika. Drugo načelo je *programiranje posredstvom korisničkog sučelja* (engl. *programming in the user interface*), tehnika programiranja u kojoj su naredbe programa jednake naredbama koje bi korisnik zadavao sustavu. Prednost ovog načina programiranja je što korisnik gradi program na osnovi naredbi s kojima je već upoznat i u smislu funkcionalnosti i u smislu njihovog oblika.

Navedena načela sadrže neke poznate probleme. Programiranje zasnovano na primjerima nije jednoznačno, jer skup primjera navedenih za oblikovanje programa može definirati beskonačno mnogo programa. Problem programiranja posredstvom korisničkog sučelja je da se za oblikovanje programa koriste sučelja prilagođena korisnicima. Ponašanja korisnika i način rada računala znatno se razlikuju i računalu razumljive programe za neke zadatke nije moguće ostvariti s gledišta korisničkog sučelja. Unatoč tim problemima, ove ideje moguće je uspješno primijeniti u različitim područjima primjene i omogućuju osobama bez znanja i iskustva u tehnikama programiranja izgradnju korisnih programa.

### **Pygmalion**

Prvi rezultat istraživanja iz područja programiranja zasnovanog na demonstraciji bio je sustav *Pygmalion*. Autor sustava David Smith je tijekom njegova stvaranja želio oblikovati okolinu za programiranje koja će poticati kreativno razmišljanje. On je identificirao različite aspekte kreativnog razmišljanja i zaključio da programski sustavi trebaju omogućiti vizualizaciju i analogiju kao glavne aspekte kreativnog razmišljanja. Samo oblikovanje *Pygmalion* sustava inspirirano je jednostavnošću korištenja programa za obradu teksta, posebno u usporedbi s programskim jezicima.

Uspoređujući obradu teksta s tehnikama programiranja, Smith je uočio da su programi za obradu teksta slični programskim jezicima namijenjenim za usko područje primjene. U programima za obradu teksta naredbe se zadaju interaktivno, dok se u programskim jezicima naredbe zapisuju u obliku programa koji se naknadno prevodi i izvodi. Smith je definirao proširenje programa za obradu teksta kojim je omogućio stvaranje zapisa interaktivno zadanih operacija. Tako ostvaren zapis slijeda operacija programa za obradu teksta sličan je zapisu naredbi programa napisanog u programskom jeziku i omogućuje jednostavno ponavljanje izvođenja operacija. Ako se takav programski sustav ne ograniči na obradu teksta, već se njegova primjena proširi i na druge podatkovne strukture te se skup operacija proširi s aritmetičkim, uvjetnim operacijama i potprogramima, onda se dobije sustav koji ima sve mogućnosti programskog jezika opće namjene.

*Pygmalion* je razvijen primjenom navedenih zaključaka. Smith je razvio posebno korisničko sučelje koje sadrži sve operacije korištene u programskom jeziku. Razvijeno sučelje je prvo sučelje koje uvodi koncept *ikona* koje se koriste kao način predstavljanja varijabli, kazaljki, funkcija i ostalih konstrukta programskog jezika.

### **Tipkovničke makro naredbe**

Tipkovničke makro naredbe (engl. *keyboard macros*) su najjednostavniji i najčešći primjer uspješnog programiranja zasnovanog na demonstraciji, za koje je najrašireniji primjer makro naredbi dostupnih u MS Office programskom paketu. Tipkovničke makro naredbe omogućuju korisnicima demonstraciju slijeda pritisaka tipki tipkovnice i akcija mišem. Sustav snimanja makro naredbi pamti slijed akcija korisnika, te nakon toga stvara program koji ponavlja snimljene akcije. Takvi programi ne sadrže uvjetne naredbe upravljanja tijekom izvođenja programa, a jedina naredba petlje je ponavljanje cijelog makro programa. Tipkovničke makro naredbe postigle su značajan praktičan uspjeh i prisutne su u gotovo svim programima za obradu teksta, programima za tablične proračune te u samim operacijskim sustavima.

Iako su tipkovničke makro naredbe ograničene po svojim mogućnostima, one su primjer mnogih pozitivnih svojstava programiranja zasnovanog na demonstraciji. Prvo, korisnik gradi program od komponenti koje su mu dostupne putem poznatog korisničkog sučelja. Drugo, korisnici specificiraju detaljno ponašanje programa unutar okruženja u kojem se izvodi. Treće, sučelje za oblikovanje programa je vrlo jednostavno i najčešće se sastoji od samo tri naredbe za pokretanje i zaustavljanje snimanja akcija te za izvođenje snimljenog niza akcija. Slijedeća prednost je što se korisniku omogućuje stvaranje programa bez potrebe da uopće poznaje strukturu snimljenih programa. Posljedica toga je da korisnik ne mora učiti kako čitati zapis programa i nije opterećen pravilima zapisa programa tijekom procesa oblikovanja programa.

### **Skriptni jezici**

Skriptni jezici (engl. *script languages*), poput jezika Perl, Python, Rexx, Tcl, JavaScript i VB Script, definiraju drugačiji stil programiranja od klasičnih programske jezike opće namjene.

Skriptni jezici prepostavljaju da unaprijed postoji skup komponenti čija je funkcionalnost ostvarena uporabom drugih programske jezike. Skriptni jezici nisu namijenjeni za pisanje cjelovite funkcionalnosti primjenskih programa, već su namijenjeni za povezivanje postojećih programske komponenti. Na primjer, jezik VB Script koristi se kako bi se povezao skup kontrola korisničkog sučelja nekog Web primjenskog sustava. Skriptni jezici se često koriste kako bi se proširile funkcionalnosti dostupnih komponenti, ali vrlo rijetko koriste složene algoritme i strukture podataka. Funkcionalnosti čije ostvarenje podrazumijeva uporabu

složenih algoritama i struktura podataka ostvarene su komponentama koje su unaprijed dostupne. Skriptni jezici se zbog opisane pretpostavke često nazivaju i *jezici povezivanja* (engl. *glue languages*) ili *jezici objedinjavanja sustava* (engl. *system integration languages*).

Kako bi se pojednostavio postupak povezivanja komponenti, skriptni jezici najčešće ne razlikuju tipove podataka već se svi podaci predstavljaju na isti način i međusobno su zamjenjivi. Na primjer, varijabla definirana u jeziku VB Script određenom trenutku sadrži tekstualnu vrijednost, ali se naknadno toj istoj varijabli pridruži cijelobrojna vrijednost. Skriptni jezici su najčešće orijentirani prema zapisu podataka u obliku teksta, jer tekst omogućava jedinstveni zapis različitih tipova podataka.

Skriptni jezici bez tipova podataka pojednostavljaju međusobno povezivanje raznorodnih komponenti. Ne postoje unaprijed definirana ograničenja o tome kako će se koja komponenta koristiti te su sve komponente i vrijednosti predstavljene na jedinstven način. Zbog toga je moguće bilo koju komponentu ili vrijednost koristiti u bilo kojoj situaciji, odnosno moguće je komponentu oblikovanu za određenu namjenu iskoristiti za neku potpuno drugu namjenu koju programer, koji je izgradio komponentu, nije mogao predvidjeti.

Jezici sa strogo definiranim tipovima podataka ograničavaju ponovnu iskoristivost programskog kôda. Tipovi podataka omogućavaju programerima stvaranje velikog broja nesukladnih sučelja programskih cjelina. Svako sučelje definira objekt točno određenog tipa i prevoditelji onemogućavaju uporabu bilo kojeg drugog tipa podataka s tim sučeljem. Programer koji koristi takvo sučelje mora napisati programski kôd koji će prilagoditi druge tipove podataka sučelju. To nadalje dovodi do potrebe za ponovnim prevođenjem dijela ili cijelog programa.

```
button .b -text Hello! -font {Times 16} -command {puts hello}
```

Slika 4.3 Primjer naredbe skriptnog jezika

Kao ilustracija prednosti jezika bez tipova podataka na slici 4.3 prikazan je primjer napisan u jeziku Tcl [72]. Prikazana naredba stvara novi gumb korisničkog sučelja s naslovom "Hello!" napisan fontom *Times*, veličine 16 točaka koji ispisuje poruku "hello" nakon što korisnik aktivira kontrolu. U ovoj naredbi se naizmjence koriste različite strukture podataka i naredbe jezika Tcl. Ime naredbe za stvaranje gumba određeno je pokazateljem *button*, a kontrola korisničkog sučelja na kojoj se stvara gumb određena je pokazateljem *.b*. Različita imena svojstava gumba određena su pokazateljima *-text*, *-font* i *-command*. Osim toga, koriste se jednostavne tekstualne konstante *Hello!* i *hello* i unaprijed definirane vrijednosti za naziv

oblika pisma *Times 16*, kojima je određeno ime vrste pisma i veličina pisma u točkama. Svi se navedeni programski elementi u jeziku Tcl jednoznačno opisuju primjenom teksta. U ovom primjeru, definirana svojstva navode se proizvoljnim redoslijedom, a preostalim nedefiniranim svojstvima pridružuju se pretpostavljene (engl. *default*) vrijednosti.

Kako bi se navedeni primjer ostvario u nekom od programskih jezika opće namjene koji razlikuju tipove podataka potrebno je napisati više naredbi, npr. u Javi je potrebno 7 redaka kôda, dok je za ostvarenje u C++ programskom jeziku, uporabom MFC biblioteke, potrebno napisati oko 25 redaka programskog kôda. Samo definiranje vrste pisma stvorenog gumba korištenjem MFC biblioteke zahtjeva pisanje nekoliko redaka kôda i prikazano je na slici 4.4.

```
CFont *fontPtr = new CFont();
    fontPtr->CreateFont(16, 0, 0, 0, 700, 0, 0, 0, ANSI_CHARSET,
    OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
    DEFAULT_PITCH|FF_DONTCARE, "Times New Roman");
buttonPtr->SetFont(fontPtr);
```

Slika 4.4 Primjer C++ programa

Većina prikazanog programskog kôda je posljedica strogo definiranih tipova programskog jezika C++. Kako bi se definirala vrijednost pisma stvorenog gumba potrebno je nad stvorenim objektom pozvati metodu *SetFont* kojoj se kao parametar mora predati pokazivač na razred *CFont*. To nadalje uzrokuje prethodno stvaranje primjerka razreda tipa *CFont* i postavljanje potrebnih parametara pisma pozivom metode *CreateFont*. Metoda *CreateFont* ima strogo definirano sučelje koje prihvata 14 parametara točno određenog tipa koji svi moraju biti navedeni za razliku od prethodno prikazanog primjera u kojem su navedeni samo željeni parametri dok su ostalima implicitno dodijeljene pretpostavljene vrijednosti.

Nedostatak potpore za različite tipove podataka u skriptnim jezicima uzrokuje neprepoznavanje pogrešaka. Međutim, praksa pokazuje da su skriptni jezici jednako sigurni kao i programski jezici opće namjene. Na primjer, u primjeru prikazanom na slici 4.3 definiranje veličine pisma necjelobrojnim vrijednostima uzrokovalo bi pogrešku prilikom izvođenja programa. Razlika je što skriptni jezici obavljaju provjeru ispravnosti i otkrivanje pogrešaka u najkasnijem mogućem trenutku, odnosno u trenutku kada se vrijednost varijable koristi. Jezici sa strogo definiranim tipovima podataka omogućuju provjeru ispravnosti programa u trenutku prevođenja izvornog programa, pa su time poboljšana radna svojstva programa, jer se izbjegava otkrivanje pogrešaka za vrijeme izvođenja programa. No poboljšanje učinkovitosti rezultira ograničenjima nametnutim na uporabu podataka u programu što rezultira većom količinom i manjom prilagodljivosti programskog kôda.

Još jedna ključna razlika između skriptnih i programskega jezika je način prevođenja izvornog programa. Programi napisani u skriptnim jezicima uglavnom se interpretiraju, dok se programi napisani u programskega jezicima najčešće prevode. Time skriptni jezici pružaju korisniku mogućnost učestalog mijenjanja izvornog programa za vrijeme razvoja eliminirajući trošak vremena potrebnog za prevođenje. Interpretatori također čine primjenske programe prilagodljivijima, jer dopuštaju korisnicima izmjenu programa tijekom izvođenja.

Jedna od posljedica interpretiranja izvornih programa napisanih skriptnim jezicima je njihova manja učinkovitost od programa napisanih programskega jezicima. Uzrok manje učinkovitosti skriptnih jezika je uporaba komponenti koje su odabrane prema potrebnim funkcionalnostima i jednostavnosti korištenja, a ne prema učinkovitosti izvođenja na konkretnim sklopovskim arhitekturama. Na primjer, skriptni jezici će u većini slučajeva koristiti tekstualne vrijednosti promjenjivih duljina, dok će programske jezici u istim situacijama koristiti binarne vrijednosti koje će odgovarati jednoj strojnoj riječi procesora. Međutim, učinkovitost programa napisanih skriptnim jezicima u većini slučajeva nije kritičan čimbenik. Primjenski programi napisani skriptnim jezicima su općenito manji od onih napisanih programskega jezicima i ukupna radna svojstva programa ovise o radnim svojstvima komponenata koje se koriste u programu, a koje su napisane u programskega jezicima.

Skriptni jezici su jezici više razine od programskega jezika, jer jedna naredba skriptnih jezika u prosjeku obavlja više strojnih instrukcija od naredbe programskega jezika. Tipična naredba skriptnih programskega jezika prevodi se u stotinu ili tisuću strojnih naredbi, za razliku od tipične naredbe programskega jezika koja se prevodi u prosječno 5 strojnih instrukcija. Jedan od uzroka većeg broja strojnih naredbi prevedenih programa napisanih skriptnim jezicima je uporaba interpretatora koji su općenito manje učinkoviti od kompilatora.

Zbog navedenih svojstava, skriptni jezici su vrlo pogodni za brzi razvoj primjenskih programa koji se zasnivaju na povezivanju postojećih komponenti. Kako komponenta u ovom okruženju predstavlja bilo koju programsku jedinku koja ima definirano programsko sučelje ona se odnosi i na mrežne usluge. Zbog toga su skriptni jezici pogodni i kao jezici kompozicije mrežnih usluga namijenjene krajnjim korisnicima. U slijedećem poglavljiju opisan je programski jezik kompozicije mrežnih usluga namijenjen krajnjim korisnicima koji je oblikovan kao skriptni jezik.

## 5 SSCL korisnički jezik za kompoziciju mrežnih usluga

Pregled postojećih programskega jezika za kompoziciju mrežnih usluga iznesen je u poglavju 3. Iako se jezici kompozicije mrežnih usluga razlikuju prema načinu definicije kompozicije mrežnih usluga, zajednička im je zasnovanost na XML-u. Tako oblikovani jezici imaju složena sintaksna pravila, a programi pisani u takvim jezicima su vrlo opsežni i teško čitljivi. Zbog toga je izgradnja kompozicije mrežnih usluga primjenom navedenih jezika vrlo zahtjevna za programere, te je dugotrajna i podložna pogreškama. Kako bi se olakšala kompozicija mrežnih usluga korištenjem jezika zasnovanih na XML-u razvijen je veći broj alata koji omogućuju oblikovanje kompozicije mrežnih usluga definiranjem grafičkog prikaza dijagrama tijeka izvođenja i tijeka podataka. No i tako oblikovani alati ne skrivaju složenost protokola i tehnologija mrežnih usluga kao što su detalji WSDL opisa mrežne usluge, strukture SOAP poruka i općenito složene strukture podataka opisanih jezikom XML koje se razmjenjuju između mrežnih usluga.

Zbog navedenih nedostataka razvijen je jezik SSCL (*Simple Service Composition Language*), korisnički jezik za kompoziciju mrežnih usluga. Jezik SSCL namijenjen je neprofesionalnim programerima i krajnjim korisnicima računalnih sustava. Korisnički jezik SSCL je skriptni jezik i ima mali skup naredbi, jednostavna sintaksna i semantička pravila te ga se zbog toga jednostavno i brzo nauči koristiti. Jezik SSCL namijenjen je oblikovanju kompozicije usluga definirane programskim modelom zasnovanim na uslugama (engl. *Service Oriented Programming Model, SOPM*).

### 5.1 Programska model zasnovan na uslugama

Programski jezik SSCL razvijen je u skladu s programskim modelom zasnovanom na uslugama (*Service Oriented Programming Model - SOPM*) [79]. Programska model zasnovan na uslugama je rezultat istraživanja na tehnološkom projektu "Okrilje javnog informacijskog sustava". Programska sustav izgrađen prema načelima programskog modela zasnovanog na uslugama sastoji se od primjenskih mrežnih usluga, raspodijeljenih programa i mrežnih usluga za suradnju i natjecanje. Primjenske mrežne usluge i mrežne usluge za suradnju i natjecanje su unaprijed dostupne mrežne usluge razvijene od strane profesionalnih programera, dok raspodijeljene programe pišu krajnji korisnici u programskom jeziku SSCL.

### 5.1.1 Primjenske mrežne usluge

Primjenske mrežne usluge (engl. *application services*) su unaprijed razvijene i dostupne putem globalne mreže Internet. Ove mrežne usluge mogu biti jednostavne mrežne usluge, ostvarene nekim od objektno-orientiranih ili komponentnih tehnologija, ili složene mrežne usluge, ostvarene kompozicijom mrežnih usluga. Primjenske mrežne usluge ostvaruju dijelove funkcionalnosti koje korisnik želi povezati u jedinstven programski sustav. Krajnji korisnik ne mora znati ništa o tehnologiji ostvarenja primjenske mrežne usluge, već samo mora dohvatiti WSDL opis sučelja primjenske usluge. Na osnovi dohvaćenog WSDL opisa korisnik poziva operacije primjenske mrežne usluge iz raspodijeljenih programa.

### 5.1.2 Raspodijeljeni programi

Raspodijeljene programe (engl. *distributed programs*) pišu krajnji korisnici u jeziku SSCL. Raspodijeljeni programi povezuju funkcionalnosti primjenskih mrežnih usluga u složenije funkcionalnosti. Nova, složena funkcionalnost ostvaruje se definiranjem koordinacijske logike primjenom raspodijeljenih programa. Koordinacijska logika ostvaruje se primjenom SSCL naredbi za definiranje tijeka izvođenja programa, naredbama poziva mrežnih usluga za suradnju i natjecanje, te naredbama poziva operacija primjenskih mrežnih usluga.

### 5.1.3 Mrežne usluge za suradnju i natjecanje

Mrežne usluge za suradnju i natjecanje sadrže skupinu mrežnih usluga s očuvanjem stanja (engl. *stateful web services*) koje ostvaruju mehanizme sinkronizacije i komunikacije raspodijeljenih programa. Međusobnom sinkronizacijom i komunikacijom raspodijeljenih programa omogućeno je definiranje raspodijeljene koordinacijske logike primjenskog sustava zasnovanog na uslugama.

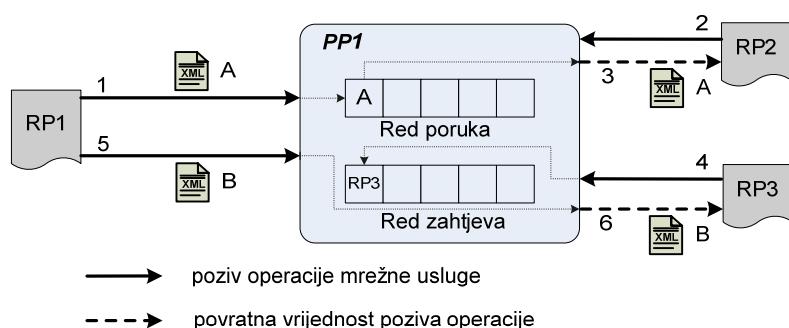
Mrežne usluge za suradnju i natjecanje sastoje se od usluge poštanskog pretinca (engl. *mailbox*), usluga binarnog i općeg semafora (engl. *binary and counting semaphore*), te usluge usmjernika događaja (engl. *event-channel*). Poštanski pretinac, binarni i opći semafor su mehanizmi čija je funkcionalnost jednaka istoimenim mehanizmima jezgre operacijskih sustava, ali su ostvareni kao mrežne usluge čime je omogućena njihova uporaba u raznorodnoj raspodijeljenoj okolini. Usmjernik događaja je složeni komunikacijski mehanizam zasnovan na objava-preplata modelu komunikacije.

#### *Poštanski pretinac*

Poštanski pretinac (engl. *mailbox*) je mehanizam suradnje i natjecanja kojim se ostvaruje asinkrona i postojana komunikacija između raspodijeljenih programa. Poruke koje se

razmjenjuju su proizvoljni XML dokumenti čime je ostvarena komunikacija neovisna o računalnom okrilju, operacijskom sustavu ili tehnologiji korištenoj za programsko ostvarenje.

Poštanski pretinac sastoji se od reda poruka i reda zahtjeva. Red zahtjeva sadrži pokazatelje raspodijeljenih programa koji žele čitati poruku iz poštanskog pretinca koji ne sadrži niti jednu poruku u redu poruka. Red poruka se koristi za spremanje pristiglih poruka ako ne postoji niti jedan zapis u redu zahtjeva. Primjer korištenja poštanskog pretinca kao mehanizma komunikacije raspodijeljenih programa prikazan je na slici 5.1. Prikazani sustav se sastoji od 3 raspodijeljena programa i jednog primjerkra poštanskog pretinca *PP1*. Raspodijeljeni program RP1 šalje poruku A u obliku XML dokumenta (1). Kako je u tom trenutku red zahtjeva prazan, pristigla poruka se spremi u red poruka. U sljedećem trenutku raspodijeljeni program RP2 šalje zahtjev za dohvata poruke iz poštanskog pretinca (2). Kako red poruka u tom trenutku sadrži poruku A, ona se šalje raspodijeljenom programu RP2 (3) i briše se iz reda poruka. Tijekom slanja sljedećeg zahtjeva za dohvata poruke (4) red poruka je prazan, te se pokazatelj raspodijeljenog programa RP3 zapisuje u red zahtjeva. U programskom ostvarenju, pokazatelj raspodijeljenog programa sastoji se od WSDL opisa sučelja povratnog poziva (engl. *callback*) raspodijeljenog programa i adrese raspodijeljenog programa definirane u skladu s normom *WS-Addressing*. Nakon što raspodijeljeni program RP1 pošalje novu poruku u poštanski pretinac (5), dohvaća se zapis iz reda zahtjeva te se raspodijeljenom programu RP3 šalje poruka B (6).



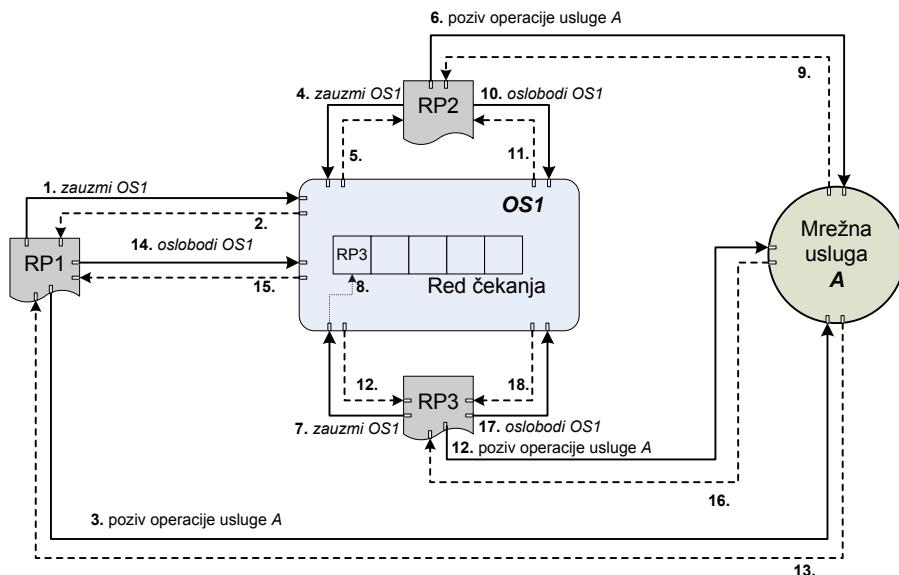
**Slika 5.1 Primjer uporabe poštanskog pretinca kao mehanizma komunikacije raspodijeljenih programa**

U programskom modelu zasnovanom na uslugama poštanski pretinac se primjenjuje za definiranje tijeka izvođenja raspodijeljene koordinacijske logike primjenskog sustava ili tijeka podataka primjenskog sustava. Primjena poštanskog pretinca ovisi o sadržaju XML dokumenata koje razmjenjuju raspodijeljeni programi.

### *Binarni i opći semafori*

Mrežne usluge binarnih i općih semafora ostvaruju funkcionalnosti istoimenih mehanizama operacijskih sustava. Dok se binarni i opći semafori operacijskih sustava koriste za

sinkronizaciju i međusobno isključivanje procesa operacijskog sustava, istoimene mrežne usluge koriste se za sinkronizaciju i međusobno isključivanje raspodijeljenih programa koji se izvode u otvorenoj raspodijeljenoj okolini.



**Slika 5.2 Primjer uporabe općeg semafora za ograničavanje pristupa primjenskoj mrežnoj usluzi**

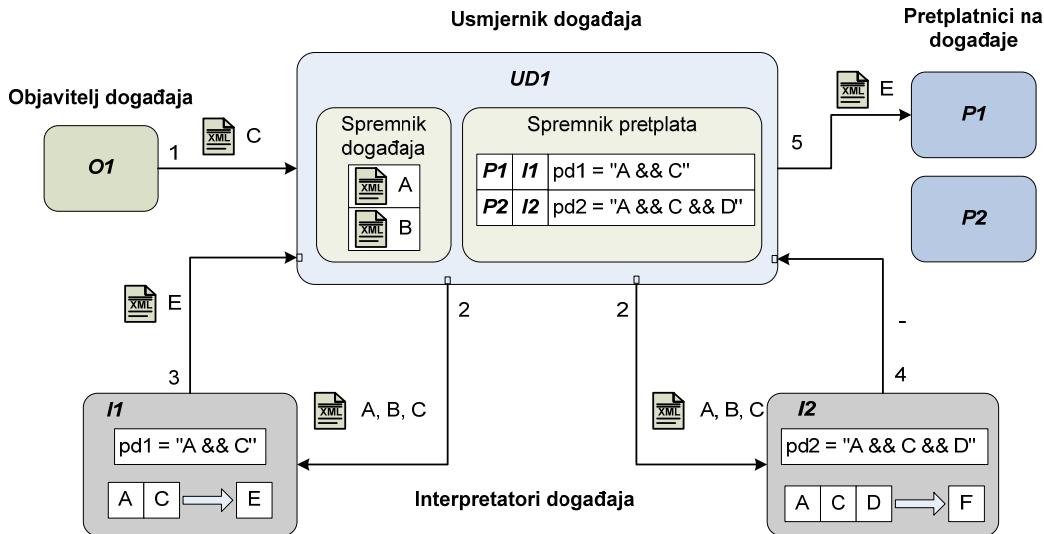
Opći semafor ima definirani kapacitet kojim je određeno koliko ga raspodijeljenih programa istodobno može zauzeti. Kapacitet semafora navodi se tijekom stvaranja primjera mrežne usluge semafora. Binarni semafor je poseban slučaj općeg semafora. Binarni semafor u jednom trenutku može zauzeti samo jedan raspodijeljeni program i njegova funkcionalnost je jednaka općem semaforu kapaciteta jedan. Opći semafori se koriste kao mehanizmi kojima se ograničava pristup računalnom sredstvu, odnosno mrežnoj usluzi. Na slici 5.2 prikazana je uporaba općeg semafora za zaštitu pristupa mrežnoj usluzi A. Pristup mrežnoj usluzi je potrebno ograničiti tako da joj istovremeno mogu pristupati samo dva raspodijeljena programa. Kako na prikazanom primjeru postoje tri raspodijeljena programa RP1, RP2 i RP3 koji pokušavaju pristupiti mrežnoj usluzi A, koristi se opći semafor OS1 kapaciteta dva. Svi raspodijeljeni programi prije pristupa usluzi A moraju zauzeti semafor OS1 te ga nakon uporabe usluge moraju osloboditi. Na prikazanom primjeru raspodijeljeni program RP1 prvo šalje zahtjev za zauzimanjem semafora OS1 (1). Semafor OS1 vraća potvrdu o uspješnom zauzimanju semafora (2) pa raspodijeljeni program nastavlja s izvođenjem i poziva željenu operaciju usluge A (3). Nakon toga, raspodijeljeni program RP2 također uspješno zauzima semafor (4, 5) i poziva operaciju usluge A (6). U tom je trenutku kapacitet semafora OS1 zauzet i kada raspodijeljeni program RP3 pošalje zahtjev za zauzimanjem semafora OS1 (7) njegov pokazatelj se spremi u red čekanja (8), gdje ostaje sve dok jedan raspodijeljeni program ne pozove operaciju oslobođanja semafora. Pokazatelj

raspodijeljenog programa koji se spremo u red čekanja sastoji se od WSDL opisa sučelja dojave raspodijeljenog programa i adrese raspodijeljenog programa zapisane prema normi *WS-Addressing*. Tek nakon što se izvrši operacija mrežne usluge (9) koju je pozvao raspodijeljeni program RP2 i nakon što on oslobodi semafor (10, 11), dohvata se prvi zapis iz reda čekanja. Prvi zapis u redu čekanja je pokazatelj raspodijeljenog programa RP3 te se raspodijeljenom programu RP3 šalje potvrda o uspješnom zauzimanju semafora (12). Nakon toga, raspodijeljeni program RP3 nastavlja s izvođenjem i poziva operaciju mrežne usluge A. Nakon završetka izvođenja operacije mrežne usluge A raspodijeljeni programi RP1 i RP3 oslobađaju semafor OS1 (13-18).

### *Usmjernik događaja*

Usmjernik događaja (engl. *event-channel*) je napredni komunikacijski mehanizam, zasnovan na objava-preplata (engl. *publish-subscribe*) komunikacijskom modelu, koji pruža napredne mogućnosti tumačenja događaja. Usmjernik događaja koristi se za izgradnju raspodijeljenih sustava poticanih događajima (engl. *event-driven systems*), raspodijeljenih sustava zasnovanih na dokumentima (engl. *document-centric systems*), mreža zasnovanih na sadržaju (engl. *content-based networks*), te na semantici zasnovanih sustava.

Arhitektura komunikacijskog modela zasnovanog na objava-preplata mehanizmu sastoji se od objavitelja, preplatnika na događaj, usmjernika i interpretatora događaja. Usmjernik događaja sastoji se od spremnika događaja i spremnika preplata. Spremnik događaja sadrži događaje generirane od strane objavitelja, a spremnik preplata služi za spremanje zapisa o preplati. Svaki se zapis o preplati sastoji od pokazatelja preplatnika, pokazatelja vanjskog interpretatora događaja i preplatničkog dokumenta. Objavitelj događaja je raspodijeljeni program koji generira događaje u obliku XML dokumenata. Objavljeni se događaji spremaju u spremnik događaja. Preplatnici su raspodijeljeni programi koji se pretplaćuju na događaje određenog sadržaja. Odabir događaja koji zanimaju preplatnika određen je preplatničkim dokumentom i vanjskim interpretatorom događaja. Vanjski interpretatori ostvaruju mehanizme za parsiranje preplatničkog dokumenta i skupa događaja. Osim toga, vanjski interpretatori ostvaruju funkcionalnosti ispitivanja sadržaja skupa događaja prema kriterijima zadanim preplatničkim dokumentom i prema kriterijima ugrađenim u programsku logiku interpretatora. Ako trenutni skup objavljenih događaja zadovoljava zadane kriterije, interpretator vraća pozitivan odgovor usmjerniku događaja te oblikuje poruku s informacijama o događajima koju usmjernik prosljeđuje pripadnom preplatniku.



Slika 5.3 Primjer korištenja usmjernika događaja

Primjer usmjernika događaja prikazan na slici 5.3 sadrži dva zapisa o pretplati u spremniku preplata. Tijekom pretplate raspodijeljenog programa *P1* naveden je interpretator događaja *I1*, a kriterij odabira događaja zadan je pretplatničkim dokumentom *pd1*. Kriterij odabira definiran pretplatničkim dokumentom *pd1* je pojava događaja *A* i događaja *B*. Drugi zapis spremljen je nakon pretplate raspodijeljenog programa *P2* i sadrži pokazatelj vanjskog interpretatora *I2*, te pretplatnički dokument *pd2*. Kriterij odabira događaja definiran pretplatničkim dokumentom *pd2* je pojava događaja *A i C i D*. Vanjski interpretator *I1* na osnovi objavljenih događaja, nakon što je zadovoljen kriterij pretplatničkog dokumenta, oblikuje poruku *E*, a interpretator *I2* oblikuje poruku *F*.

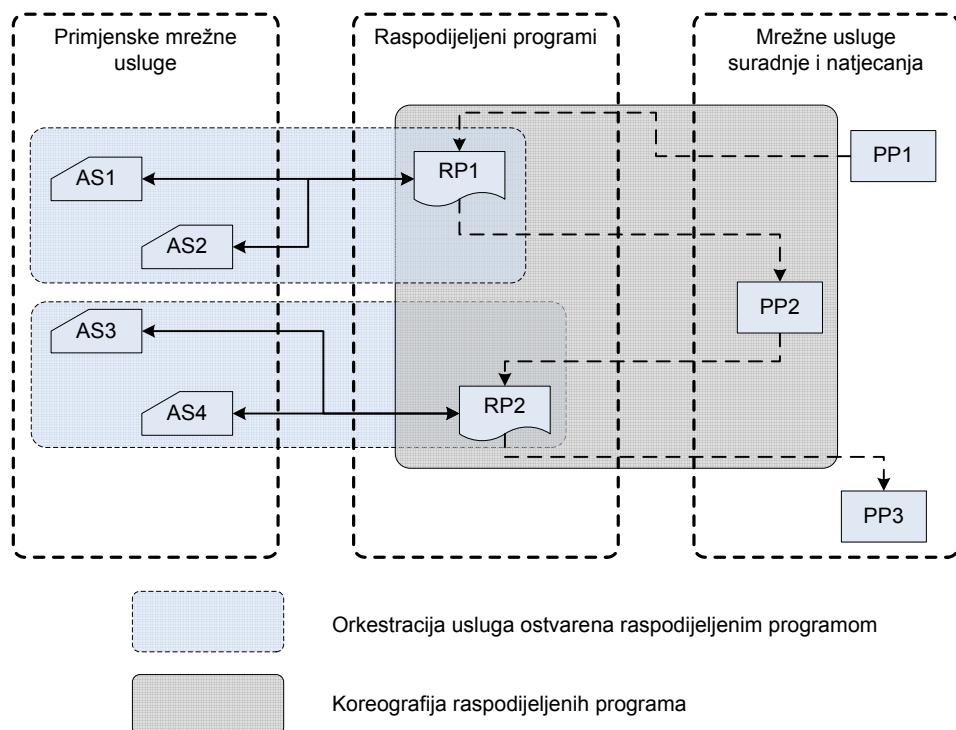
Ako u stanju sustava prikazanom na slici 5.3 raspodijeljeni program *O* objavi događaj *C* (1), spremnik događaja će sadržavati događaje *A*, *B* i *C*. Nakon objave događaja, cijeli se skup događaja šalje svim interpretatorima navedenim u zapisima spremnika preplata. Nakon primitka skupa događaja kriterij odabira događaja interpretatora *I1* će biti zadovoljen i on će oblikovati poruku *E* koju šalje usmjerniku događaja (3). Kriterij odabira interpretatora *I2* nije zadovoljen i on usmjerniku događaja šalje negativan odgovor na skup primljenih događaja (4). Usmjernik događaja će stvorenju poruku *E* proslijediti raspodijeljenom programu *P1* (5).

#### 5.1.4 Ostvarenje složene mrežne usluge

Primjer ostvarenja složene mrežne usluge koja koristi funkcionalnosti četiri primjenske mrežne usluge prikazan je na slici 5.4. Prikazani primjer ostvaruje raspodijeljenu koordinacijsku logiku primjenom dva raspodijeljena programa i tri poštanska pretinca.

Raspodijeljeni program *RP1* ostvaruje koordinacijsku logiku kompozicije primjenskih usluga *AS1* i *AS2* definiranjem redoslijeda pozivanja njihovih operacija. Raspodijeljeni program *RP2*

ostvaruje koordinacijsku logiku potrebnu za kompoziciju primjenskih usluga AS3 i AS4. Koordinacijska logika kompozicije svih primjenskih mrežnih usluga definirana je redoslijedom izvođenja raspodijeljenih programa RP1 i RP2. Redoslijed izvođenja raspodijeljenih programa RP1 i RP2 usklađuje se uporabom poštanskih pretinaca.



**Slika 5.4 Primjer ostvarenja složene mrežne usluge**

Kompozicija usluga prema programskom modelu zasnovanom na uslugama ostvaruje se definiranjem koordinacijske logike primjenom raspodijeljenih programa. Kompozicija ostvarena primjenom raspodijeljenih programa omogućuje definiranje hibridnog modela kompozicije koji objedinjuje postupke orkestracije i koreografije. Raspodijeljeni program definira centralizirani proces kojim se ostvaruje orkestracija mrežnih usluga suradnika. Na slici 5.4 raspodijeljeni program RP1 definira orkestraciju mrežnih usluga suradnika AS1 i AS2. Za oblikovanje kompozicije usluga s jednostavnom koordinacijskom logikom, moguće je cijelu kompoziciju opisati orkestracijom definiranom jednim raspodijeljenim programom. Složena kompozicija usluga oblikuje se primjenom više raspodijeljenih programa objedinjenih u jedinstvenu kompoziciju usluga primjenom usluga suradnje i natjecanja. Usluge suradnje i natjecanja omogućuju definiranje raspodijeljene koordinacijske logike, odnosno koreografije raspodijeljenih programi. Objedinjavanjem raspodijeljenih programa primjenom usluga suradnje i natjecanja stvorena je hibridna kompozicija usluga primjenom postupaka orkestracije i koreografije.

Raspodjelom koordinacijske logike ubrzava se izvođenje složene usluge, jer se više raspodijeljenih programa istovremeno izvodi na više čvorova. Osim ubrzanja izvođenja, raspodjelom koordinacijske logike omogućeno je pojednostavljenje skupa naredbi programskog jezika SSCL. Naime, neke naredbe upravljanja tijekom izvođenja programa su izostavljane iz definicije SSCL jezika. Primjer naredbe upravljanja tijekom izvođenja koja nije definirana jezikom SSCL je naredba *flow* programskog jezika BPEL koja pokreće izvođenje dvije paralelne grane u grafu izvođenja. Paralelno izvođenje više grana grafa izvođenja moguće je ostvariti definiranjem zasebnog raspodijeljenog programa za svaku granu i dodavanjem naredbi za upravljanje nekom od usluga suradnje i natjecanja na početak i na kraj paralelnih grane.

## 5.2 Jezik kompozicije mrežnih usluga namijenjen krajnjim korisnicima

Programski jezik SSCL razvijen je s ciljem jednostavnog oblikovanja raspodijeljenih programa, odnosno koordinacijske logike kompozicije mrežnih usluga. Izvodljivi raspodijeljeni programi napisani su u jeziku CL (*Coopetition Language*). U radu [77] ostvaren je raspodijeljni interpretator CL programa koji omogućuje izvođenje raspodijeljenih programa u raznorodnim računalnim okolinama. Jezik CL je na XML-u zasnovan jezik i kao takav nije prikladan za neprofesionalne programere i krajnje korisnike računalnih sustava.

Osnovna razlika jezika SSCL u odnosu na jezik CL i ostale jezike za kompoziciju mrežnih usluga je da jezik SSCL nije zasnovan na jeziku XML već je tekstualni programski jezik. Jezik SSCL osmišljen je kao jednostavan jezik koji će omogućiti krajnjim korisnicima jednostavno i brzo definiranje kompozicije mrežnih usluga.

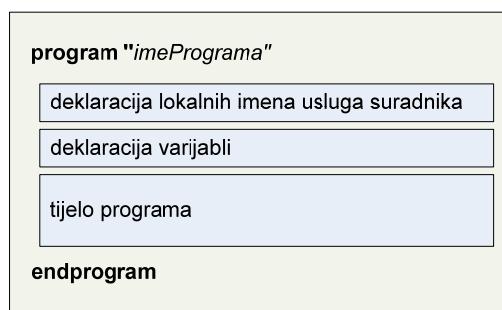
Razmatranjem postojećih rezultata u području tehnika programiranja i programskih jezika namijenjenih krajnjim korisnicima te zahtijevanim osobinama jezika kompozicije mrežnih usluga odabran je model skriptnog jezika kao osnova za razvoj jezika SSCL.

Kako je jezik SSCL namijenjen oblikovanju kompozicije mrežnih usluga, svojstvo skriptnih jezika da se koriste za povezivanje postojećih programskih komponenti je samo po sebi zadovoljeno. Skup naredbi jezika SSCL sadrži samo naredbe definicije tijeka izvođenja poziva mrežnih usluga, te njihovu međusobnu sinkronizaciju i komunikaciju. Sva funkcionalnost programa pisanih jezikom SSCL ostvarena je mrežnim uslugama, za koje se prepostavlja da su unaprijed razvijene i da su na raspolaganju krajnjim korisnicima. Korisnik naslovljava postojeće usluge na osnovi njihova WSDL opisa i povezuje njihove operacije stvarajući složenu funkcionalnost potrebnu za ostvarenje željenog cilja.

U skladu sa svojstvom skriptnih programskih jezika da ne razlikuju tipove podataka, sve varijable deklarirane SSCL programskim jezikom su istog, općenitog tipa i sadrže samo tekstualne podatke. Kako je jezik SSCL namijenjen kompoziciji mrežnih usluga, deklarirane varijable se u većini slučajeva koriste za definiranje tijeka podataka između poziva različitih mrežnih usluga, odnosno za spremanje ulaznih i izlaznih parametara poziva operacija mrežnih usluga. Ulazni i izlazni parametri operacija mrežnih usluga zapisani su u XML obliku definiranom WSDL opisom mrežnih usluga. Kako je XML posebno strukturirani tekstualni jezik, tekstualni tip podataka je prikladan za spremanje sadržaja varijabli jezika SSCL. Jedinstveni način zapisa podataka omogućuje da ista varijabla u različitim vremenskim razdobljima sadrži različite tipove podataka. Na primjer, ista varijabla može sadržavati složene podatke u XML formatu koji sadrže rezultat poziva operacije mrežne usluge, ali se ista varijabla može nakon toga uporabiti kao uvjetna varijabla naredbe grananja te sadržavati jednostavnu logičku vrijednost *true* ili *false*.

### 5.3 Struktura SSCL programa

Osnovna struktura raspodijeljenog programa napisanog u jeziku SSCL prikazana je na slici 5.6. Raspodijeljeni programi započinju ključnom riječi *program* nakon koje se navodi ime programa. Raspodijeljeni program završava ključnom riječi *endprogram*. Raspodijeljeni program sastoji se od bloka deklaracije lokalnih imena usluga suradnika, bloka deklaracije varijabli i bloka naredbi tijela programa. Detaljna struktura raspodijeljenog programa i pojedinih naredbi jezika SSCL formalno je opisana gramatikom prikazanom u dodatku B.



Slika 5.5 Struktura raspodijeljenog programa napisanog u jeziku SSCL

Struktura jezika detaljnije je prikazana u primjeru na slici 5.6. Nakon ključne riječi *program* i definiranja imena raspodijeljenog progama "Primjer", navodi se blok deklaracija lokalnih imena usluga suradnika. Naredbe bloka deklaracije lokalnih imena mrežnih usluga suradnika započinju ključnom riječi *service* nakon koje se navodi lokalno ime mrežne usluge i stvarni URL usluge. Na primjeru su prikazane tri naredbe deklaracije lokalnih imena mrežne usluge. Nakon bloka deklaracije lokalnih imena usluga slijedi blok deklaracije varijabli. Blok

deklaracije varijabli sastoje se od niza naredbi koje započinju ključnom riječi *variable* nakon koje se navode pokazatelji varijabli.

Nakon deklaracija lokalnih imena usluga i varijabli slijedi tijelo programa koje se sastoje od naredbi poziva mrežnih usluga, naredbi upravljanja tijekom programa i naredbi pridruživanja. Na prikazanom primjeru prva naredba u tijelu programa je naredba pridruživanja kojom se varijabli *condition* pridružuje vrijednost *true*. Nakon toga se ta varijabla koristi kao uvjetna varijabla naredbe petlje. Naredba petlje je složena naredba koja započinje ključnom riječi *while*. Nakon ključne riječi *while* navodi se uvjetna varijabla, iza koje slijedi blok naredbi koje se ponavljaju sve dok je uvjet ponavljanja zadovoljen. Naredba petlje završava ključnom riječi *endwhile*.

```
program "Primjer"

service trazilica, "http://api.google.com/search/beta2"
service prikaz, "http://ris.zemris.fer.hr/OutputService/OutputService.asmx"
service provjera, "http://ris.zemris.fer.hr/CheckingService/Service1.asmx"

variable ulaz, izlaz
variable uvjet, rezultat

uvjet = "true"

while uvjet
    getmessage "192.168.0.1", "PP1", ulaz
    invoke trazilica, "DoGoogleSearch", ulaz, izlaz
    invoke provjera, "CheckSearchResult", izlaz, rezultat

    if rezultat
        uvjet = "false"
    endif

endwhile

invoke prikaz, "ShowResult", izlaz

endprogram
```

Slika 5.6 Primjer programa napisanog u korisničkom jeziku SSCL

U navedenom primjeru se blok naredbi petlje sastoji od naredbe čitanja poruke iz poštanskog pretinca, naredbi poziva operacija mrežnih usluga i naredbe uvjetnog grananja. Prva naredba unutar *while* petlje je primjer ugrađene naredbe poziva operacije mrežne usluge poštanskog pretinca. Naredbom *getmessage* dohvaća se poruka iz primjerka poštanskog pretinca PP1, na računalu s adresom 192.168.0.1 i dohvaćena vrijednost se spremi u varijablu *ulaz*. Nakon toga se naredbom *invoke* poziva operacija "DoGoogleSearch" mrežne usluge određene lokalnim imenom *trazilica* i kao ulazni parametar se predaje varijabla *ulaz*, a rezultat operacije se spremi u varijablu *izlaz*. Lokalno ime *trazilica* je prvom naredbom deklaracije mrežne usluge povezano sa stvarnim URL-om mrežne usluge "<http://api.google.com/search/beta2>". Nakon poziva još jedne mrežne usluge navedena je naredba uvjetnog grananja koja počinje ključnom riječi *if*, nakon koje se navodi uvjetna

varijabla *rezultat*, blok naredbi i konačno naredba grananja završava *endif* ključnom riječi. Prikazani primjer završava još jednom naredbom poziva operacije mrežne usluge *prikaz*.

## 5.4 Naredbe programskog jezika SSCL

Skup naredbi SSCL programskog jezika dijeli se na četiri osnovna podskupa, naredbe deklaracije, naredba poziva mrežnih usluga, naredbe upravljanja tijekom i ugrađene naredbe za uporabu usluga suradnje i natjecanja. Naredbe za uporabu usluga suradnje i natjecanja dijele se na naredbe upravljanja binarnim i općim semaforima, naredbe upravljanja poštanskim pretincima i naredbe upravljanja usmjernikom događaja.

### 5.4.1 Naredbe deklaracije

Naredbe deklaracije sastoje se od naredbi deklaracije varijabli i naredbi deklaracije lokalnih imena mrežnih usluga suradnika. Deklaracija varijabli ostvaruje se jednostavnom naredbom koja se sastoji od ključne riječi *variable* i liste imena varijabli koje se deklariraju. Kako se u jeziku SSCL ne razlikuju tipovi podataka, sve deklarirane varijable sadrže tekstualne vrijednosti. Spremanjem samo tekstualnih vrijednosti u variable nije znatno ograničena upotrebljivost jezika za kompoziciju mrežnih usluga, jer se s mrežnim uslugama komunicira XML porukama koje se šalju kao tekst. Samo deklarirane varijable moguće je koristiti kao parametre pri pozivu mrežnih usluga i kao uvjetne varijable naredbi upravljanja tijekom izvođenja programa.

Deklaracija mrežnih usluga naredbom *service* omogućuje definiranje lokalnih imena mrežnih usluga suradnika u SSCL programu i povezivanje lokalnih imena usluga s odgovarajućim WSDL opisom mrežne usluge. Time je korisniku omogućeno definiranje kratkih imena koja se koristite za naslovaljavanje mrežnih usluga suradnika umjesto dugačkih i nepreglednih URL-ova kojima se mrežna usluga jedinstveno identificira na mreži Internet. Lokalna imena usluga suradnika deklariraju se za usluge bez očuvanja stanja i usluge s očuvanjem stanja (engl. *stateful services*). Za deklariranje lokalnih imena usluga s očuvanjem stanja potrebno je osim URL-a usluge navesti i pokazatelj primjera usluge suradnika (engl. *service instance identifier*). Uporabom kratkih imena povećava se preglednost programskog kôda i smanjuje se vjerojatnost pogreške tijekom pisanja programa. Osim toga, za deklarirana lokalna imena usluga unaprijed se dohvaća WSDL datoteka s opisom mrežne usluge što optimira proces prevodenja.

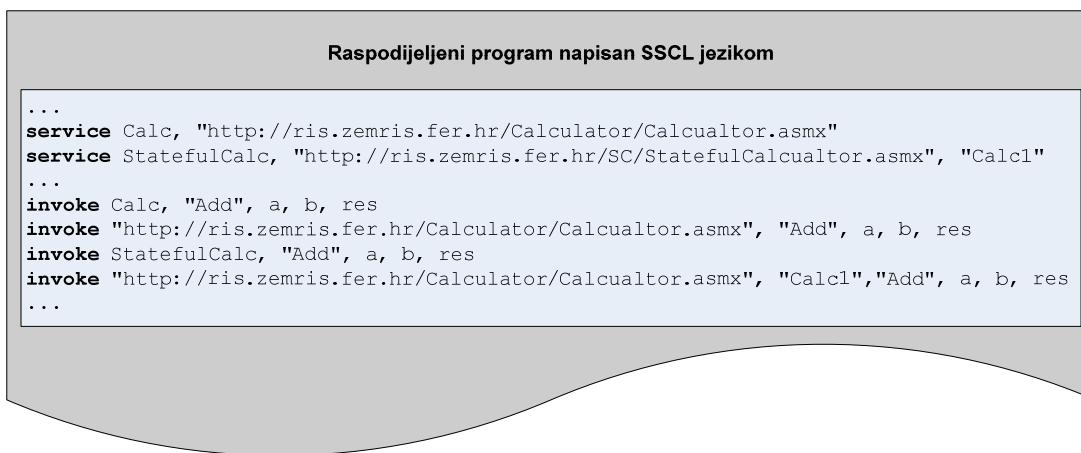
Za razliku od varijabli, koje se obavezno moraju deklarirati kako bi se koristile u ostatku programa, lokalna imena usluga ne moraju se deklarirati za sve mrežne usluge suradnike. Naredba poziva mrežnih usluga može, osim lokalnog imena usluge, kao parametar primati i

izvorni URL mrežne usluge. No za tako naslovljene mrežne usluge nije moguće unaprijed dohvatiti WSDL opis usluge, već se tijekom prevođenja naredbe poziva mrežne usluge posebno dohvaća WSDL opis.

#### 5.4.2 Naredba poziva mrežnih usluga

Kako je SSCL jezik za kompoziciju mrežnih usluga, središnja naredba jezika je naredba poziva mrežnih usluga *invoke*. Naredba *invoke* je naredba za sinkroni poziv operacije mrežne usluge suradnika. Naredbom *invoke* moguće je pozvati operacije mrežnih usluga bez očuvanja stanja i operacije mrežnih usluga sa očuvanjem stanja. Parametri naredbe poziva operacije usluge bez očuvanja stanja su pokazatelj mrežne usluge suradnika, pokazatelj operacije mrežne usluge koja se želi pozvati, varijable koje se šalju kao ulazni parametri operacije, te varijabla u koju se spremi rezultat operacije. Dodatni parametar naredbe za poziv operacije usluge s očuvanjem stanja je pokazatelj primjerka usluge.

Naredbu *invoke* moguće je napisati na nekoliko načina koji se razlikuju prema načinu naslovljavanja usluge suradnika i vrsti usluge suradnika. Pokazatelj usluge suradnika mora jedinstveno naslovljavati mrežnu uslugu unutar jednog SSCL programa. Pokazatelj mrežne usluge može biti lokalno ime mrežne usluge suradnika, prethodno definirano naredbom *service*. No ukoliko nije definirano lokalno ime mrežne usluge, kao pokazatelj mrežne usluge bez očuvanja stanja koristiti se URL usluge, odnosno kao pokazatelj usluge s očuvanjem stanja koristi se URL usluge s očuvanjem stanja i pokazatelj primjerka usluge.



**Slika 5.7 Mogućnosti naslovljavanja mrežnih usluga naredbom *invoke***

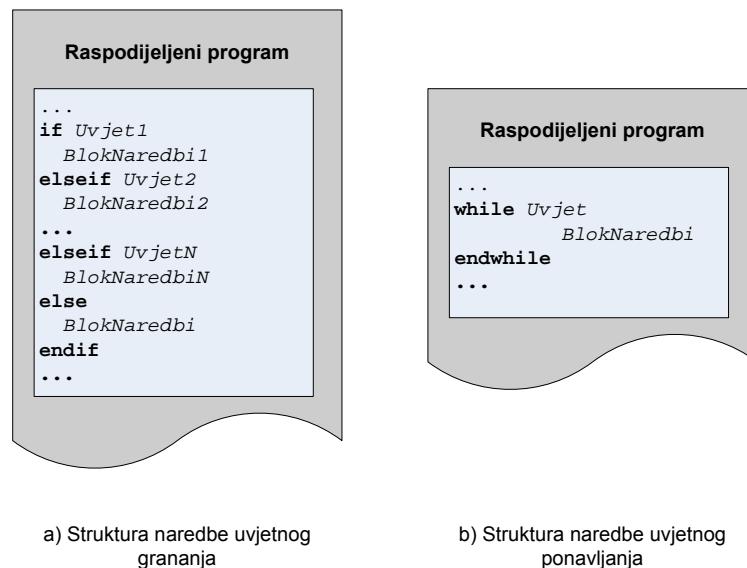
Programski odsječak prikazan na slici 5.7 prikazuje dva načina poziva operacije *Add* iste mrežne usluge bez očuvanja stanja i dva načina poziva operacije *Add* istog primjerka usluge sa očuvanjem stanja. U prvoj naredbi *invoke* mrežna usluga poziva se na osnovi prethodno definiranog lokalnog imena mrežne usluge, dok se u drugoj naredbi mrežna usluga poziva na osnovi URL-a mrežne usluge. U trećoj naredbi se poziva operacija *Add*, primjerka mrežne

usluge određenog lokalnim imenom *StatefulCalc*, dok se u četvrtoj naredbi poziva operacija *Add*, primjerka usluge naslovljenog URL-om usluge i pokazateljem primjerka *Calc1*.

### 5.4.3 Naredbe upravljanja tijekom izvođenja programa

Jezik SSCL sadrži dvije naredbe upravljanja tijekom izvođenja: naredbu petlje *while* i *if-then-else* naredbu uvjetnog grananja. Struktura naredbe uvjetnog grananja prikazana je na slici 5.8a. Naredba počinje ključnom riječi *if* nakon koje se navodi izraz *Uvjet1* kojim se određuje uvjet izvođenja bloka naredbi *BlokNaredbi1*. Nakon toga slijedi ključna riječ *elseif* te uvjetni izraz *Uvjet2* koji određuje uvjet izvođenja bloka naredbi *BlokNaredbi2*. Nakon proizvoljnog broja ponavljanja *elseif* bloka moguće je navesti *else* blok koji se izvodi ako niti jedan od prethodno navedenih uvjetnih izraza nije zadovoljen. Blok *else* počinje ključnom riječi *else* iza koje slijedi blok naredbi *BlokNaredbi*. Cjelokupna *else* naredba završava ključnom riječi *endif*. Struktura naredbe uvjetnog ponavljanja prikazana je na slici 5.8b Naredba počinje ključnom riječi *while*, nakon koje slijedi uvjet grananja *Uvjet* i blok naredbi koje se ponavljaju dok je uvjet ispunjen. Blok naredbi petlje završava ključnom riječi *endwhile*.

Uvjetni izraz naredbe grananja i naredbe petlje moguće je definirati na dva načina, uvjetnom varijablom ili pozivom uvjetne mrežne usluge. Varijabla se specificira pokazateljem variable, a poziv uvjetne mrežne usluge specificira se na isti način kao i poziv usluge suradnika naredbom *invoke*. Jedina razlika između poziva usluge suradnika naredbom *invoke* i poziva uvjetne usluge je da se tijekom poziva uvjetne usluge navode samo ulazni parametri operacije.



a) Struktura naredbe uvjetnog grananja

b) Struktura naredbe uvjetnog ponavljanja

Slika 5.8 Struktura naredbi upravljanja tijekom izvođenja programa

#### 5.4.4 Naredbe za uporabu usluge poštanskog pretinca

Poštanski pretinac (engl. *mailbox*) je mrežna usluga koja se koristi kao mehanizam za komunikaciju i sinkronizaciju raspodijeljenih programa i opisana je u odjeljku 5.1.3. Skup naredbi SSCL jezika namijenjen za korištenje mrežne usluge poštanskog pretinca omogućuje jednostavno stvaranje i uništavanje primjeraka (engl. *instance*) poštanskih pretinaca, te slanje i primanje poruka iz poštanskog pretinca. Skup naredbi za rukovanje poštanskim pretincima opisan je u tablici 5.1.

**Tablica 5-1 Naredbe za uporabu usluge poštanskog pretinca**

Naredba	Objašnjenje
<b><i>createmailbox "address", "mbID"</i></b>	Naredba za stvaranje primjerka poštanskog pretinca s pokazateljem <i>mbID</i> na računalu s adresom <i>address</i> .
<b><i>destroymailbox "address", "mbID"</i></b>	Naredba za uništavanje primjerka poštanskog pretinca s pokazateljem <i>mbID</i> na računalu s adresom <i>address</i> .
<b><i>getmessage "address", "mbID", var</i></b>	Naredba za čitanje poruke iz poštanskog pretinca s pokazateljem <i>mbID</i> na računalu s adresom <i>address</i> . Poruka se spremi u SSCL varijablu <i>var</i> .
<b><i>putmessage "address", "mbID", var</i></b>	Naredba za slanje poruke u poštanski pretinac s pokazateljem <i>mbID</i> na računalu s adresom <i>address</i> . Sadržaj poruke se nalazi u SSCL varijabli <i>var</i> .

#### 5.4.5 Naredbe za uporabu usluga općeg i binarnog semafora

Usluge binarnih i općih semafora omogućuju sinkronizaciju i ostvarenje međusobnog isključivanja raspodijeljenih programa. Opis usluga binarnog i općeg semafora sadržan je u odjeljku 5.1.3. Skup naredbi za korištenje općeg i binarnog semafora sastoji se od naredbi za stvaranje i uništavanje primjeraka usluge općeg i binarnog semafora, te naredbi zauzimanja i oslobođanja semafora. Opis svih naredbi za uporabu usluga binarnih i općih semafora sadržan je u tablici 5.2.

Istovjetne naredbe za binarne i opće semafore iste su strukture. Jedina iznimka je naredba stvaranja primjerka usluge općeg semafora koja, za razliku od naredbe stvaranja primjerka binarnog semafora, ima dodatni parametar kojim se definira kapacitet stvorenog primjerka općeg semafora.

Sve naredbe korištenja usluga binarnih i općih semafora su neblokirajuće naredbe, osim naredbe zauzimanja semafora koja blokira tijek izvođenja raspodijeljenog programa do uspješnog zauzimanja naslovljenog semafora.

Tablica 5-2 Naredbe za uporabu usluga binarnog i općeg semafora

Naredba	Objašnjenje
<b><i>createBinarySemaphore "address", "semID"</i></b>	Naredba za stvaranje primjera binarnog i općeg semafora s pokazateljem <i>semID</i> na računalu s adresom <i>address</i> . Tijekom stvaranja općeg semafora kao parametar se navodi i kapacitet semafora ( <i>capacity</i> ).
<b><i>createCountingSemaphore "address", "semID", capacity</i></b>	
<b><i>destroyBinarySemaphore "address", "semID"</i></b>	Naredba za uništavanje primjera binarnog i općeg semafora s pokazateljem <i>semID</i> na računalu s adresom <i>address</i> .
<b><i>destroyCountingSemaphore "address", "semID"</i></b>	
<b><i>obtainBinarySemaphore "address", "semID"</i></b>	Naredba za zauzimanje binarnog i općeg semafora s pokazateljem <i>SemID</i> na računalu s adresom <i>address</i> .
<b><i>obtainCountingSemaphore "address", "semID"</i></b>	
<b><i>releaseBinarySemaphore "address", "semID"</i></b>	Naredba za oslobađanje binarnog i općeg semafora s pokazateljem <i>SemID</i> na računalu s adresom <i>address</i> .
<b><i>releaseCountingSemaphore "address", "semID"</i></b>	

#### 5.4.6 Naredbe za uporabu usluge usmjernika događaja

Programski jezik SSCL sadrži skup naredbi za uporabu usluge usmjernika događaja. Usluga usmjernika događaja opisana je u odjeljku 5.1.3. Osim naredbi za stvaranje i uništavanje primjera usmjernika događaja, te naredbe *publish* za objavu događaja i naredbe *subscribe* za pretplatu na događaje čija namjena je opisana u odjeljku 5.1.3., definirane su naredba *republish* za ponovnu objavu događaja, naredba *unpublish* za povlačenje objavljenog događaja i naredba *unsubscribe* za prekid pretplate na događaj. Sve naredbe za korištenje usluge usmjernika događaja, njihova struktura i opis prikazane su u tablici 5.3.

**Tablica 5-3 Naredbe za uporabu usmjernika događaja**

Naredba	Objašnjenje
<b>createEventchannel</b> "address", "echID"	Naredba za stvaranje primjera usmjernika događaja s pokazateljem <i>echID</i> na računalu s adresom <i>address</i> .
<b>destroyEventchannel</b> "address", "echID"	Naredba za uništavanje primjera usmjernika događaja s pokazateljem <i>echID</i> na računalu s adresom <i>address</i> .
<b>publish</b> "address", "echID", <i>eType</i> , <i>eDoc</i> , <i>eID</i>	Naredba za objavu događaja tipa <i>eType</i> , čiji sadržaj je zapisan u varijabli <i>eDoc</i> . Objavljeni se događaj spremu u primjerak usmjernika događaja s pokazateljem <i>echID</i> na računalu s adresom <i>address</i> . Rezultat operacije je pokazatelj događaja <i>eID</i> .
<b>unpublish</b> "address", "echID", <i>eID</i> , <i>res</i>	Naredba za poništavanje prethodno objavljenog događaja određenog pokazateljem događaja <i>eID</i> i spremljenog u primjerak usmjernika događaja s pokazateljem <i>echID</i> na računalu s adresom <i>address</i> .
<b>republish</b> "address", "echID", <i>eType</i> , <i>eID</i> , <i>eDoc</i> , <i>newID</i>	Naredba za ponovnu objavu prethodno objavljenog događaja.
<b>subscribe</b> "address", "echID", <i>interEPR</i> , <i>cb</i> , <i>subDoc</i> , <i>subID</i>	Naredba za pretplatu na događaje objavljene u primjerku usmjernika događaja <i>echID</i> , na adresi <i>address</i> . Interpretator koji obrađuje događaje na koje se pretplatio korisnik određen je pokazateljem <i>interEPR</i> . <i>subDoc</i> je pretplatnički dokument koji sadrži korisnički definirane uvjete obrade događaja. Varijabla <i>cb</i> sadrži adresu raspodijeljenog programa i WSDL opis sučelja dojave raspodijeljenog programa. Rezultat operacije je pokazatelj pretplate <i>subID</i> .
<b>unsubscribe</b> "address", "echID", <i>subID</i> , <i>res</i>	Naredba za ukidanje pretplate definirane pokazateljem pretplate <i>subID</i> , objavljene u primjerku usmjernika događaja <i>echID</i> , na adresi <i>address</i> . Rezultat operacije se spremu u varijablu <i>res</i> .

## 5.5 Izražajnost SSCL korisničkog jezika

Kako je SSCL programski jezik za kompoziciju mrežnih usluga zasnovanih na oblikovanju procesa, njegova izražajnost moguće je ocijeniti istom metodom kojom se određuje izražajnost jezika za oblikovanje *tijeka rada* (engl. *workflow languages*).

Jezici za oblikovanje tijeka rada definiraju redoslijed izvođenja aktivnosti koje povezane čine cjelokupni radni tijek poslovnog procesa. Izražajnost jezika za oblikovanje tijeka rada ocjenjuje se brojem uzoraka tijeka izvođenja aktivnosti koje su ponuđene jezikom. Mnogi uzorci tijeka izvođenja koji su česti u stvarnim poslovnim procesima nisu izravno ponuđeni

konstruktima jezika, ali ih je moguće ostvariti kombinacijom više ponuđenih konstrukata. U takvim slučajevima izražajnost jezika se ocjenjuje na osnovu složenosti postupka ostvarenja uzorka kombinacijom jezičnih konstrukata. U radu [62] autori su ispitali postojeće sustave za oblikovanje i izvođenje tijeka rada poslovnih procesa i odredili 20 uzoraka tijeka izvođenja koji su izravno ili neizravno ponuđeni u većini sustava. Proučavanjem tih 20 uzoraka tijeka izvođenja prepoznato je desetak uzoraka koji su primjenjivi na kompoziciju mrežnih usluga i na osnovi njih je ocijenjena izražajnost programskog jezika SSCL. Odabrani uzorci tijeka izvođenja mrežnih usluga podijeljeni su na osnovne uzorce i napredne uzorce.

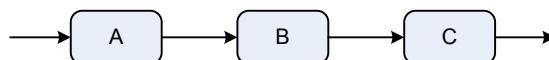
### 5.5.1 Osnovni uzorci tijeka izvođenja

Osnovni uzorci tijeka izvođenja koriste se za ostvarenje osnovnih modela uređenja aktivnosti u poslovnim procesima. Osnovne uzorce čine uzorak slijednog poziva mrežnih usluga, uzorak isključivog grananja, uzorak petlje i uzorak paralelnog poziva mrežnih usluga.

#### *Uzorak slijednog poziva mrežnih usluga*

Slijed poziva mrežnih usluga definira tijek izvođenja u kojem završetak pozvane operacije jedne mrežne usluge pokreće izvođenje operacije sljedeće usluge u nizu. Drugi nazivi koji se koriste za ovaj uzorak su slijedno usmjeravanje (engl. *sequential routing*) i serijsko usmjeravanje (engl. *serial routing*).

Primjer uporabe uzorka slijednog poziva mrežnih usluga u poslovnom procesu je operacija izdavanja računa koju je moguće izvesti tek nakon što je izvedena operacija plaćanja traženog iznosa ostvarena nekom drugom uslugom.

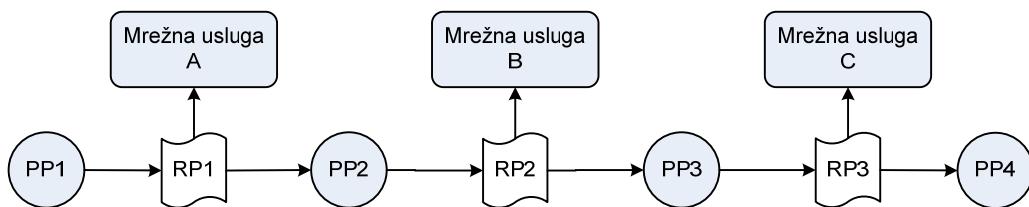


**Slika 5.9 Uzorak slijednog poziva mrežnih usluga**

Uzorak slijednog poziva mrežnih usluga moguće je izravno ostvariti jednim raspodijeljenim programom nizom *invoke* naredbi. No osim takvog centraliziranog ostvarenja, uzorak slijeda moguće je ostvariti primjenom više raspodijeljenih programa koji se međusobno sinkroniziraju uporabom nekih od opisanih sinkronizacijskih mehanizama.

Primjer na slici 5.10 prikazuje raspodijeljeno ostvarenje uzorka slijeda poziva mrežnih usluga prikazanog na slici 5.9, koje se sastoji od tri raspodijeljena programa koji se sinkroniziraju uporabom poštanskih pretinaca. Uzorak se počinje izvoditi nakon što program RP1 pročita poruku iz poštanskog pretinca PP1. Nakon čitanja poruke iz poštanskog pretinca poziva se operacija mrežne usluge A i nakon što je izvedena operacija, raspodijeljeni program RP2 šalje poruku u poštanski pretinac PP2. Izvođenje raspodijeljenog programa RP2 je blokirano

sve dok ne pročita poruku iz poštanskog pretinca PP2. Nakon što je poruka pročitana, izvođenje programa se nastavlja i poziva se mrežna usluga B, odnosno izvodi se aktivnost B iz slijeda aktivnosti. Izvođenje raspodijeljenog programa RP3 je blokirano operacijom čitanja poruke iz poštanskog pretinca PP3. Tek nakon što raspodijeljeni program RP2 pošalje poruku u poštanski pretinac PP3, raspodijeljeni program RP3 nastavlja s izvođenjem i poziva operaciju mrežne usluge C. Nakon što se izvede operacija usluge C, RP3 šalje poruku u poštanski pretinac PP4 i time je izvođenje uzorka slijednog poziva usluga završeno.



**Slika 5.10 Raspodijeljeno ostvarenje uzorka slijednog poziva mrežnih usluga**

Ostvarenje raspodijeljenih programa, napisanih u programskom jeziku SSCL, prikazano je u tablici 5.4. Svi programi na početku izvođenja čitaju podatak iz ulaznog poštanskog pretinca, pozivaju određene operacije mrežne usluge i nakon toga šalju poruku u izlazni poštanski pretinac.

**Tablica 5-4 Ispis raspodijeljenih programa uzorka slijeda aktivnosti**

Ispis programa RP1	Ispis programa RP2
<pre> program RP1 service A, " http://www.pie.fer.hr/ServiceA" variable a, b, result, input getmessage "IP1", "PP1", input invoke A, "Operation", a, b, result putmessage "IP2", "PP2", result endprogram </pre>	<pre> program RP2 service B, " http://www.pie.fer.hr/ServiceB" variable input, result getmessage "IP2", "PP2", input invoke B, "Operation", input, result putmessage "IP3", "PP3", result endprogram </pre>
<b>Ispis programa RP3</b>	
<pre> program RP1 service C, " http://www.pie.fer.hr/ServiceC" variable input, result getmessage "IP3", "PP3", input invoke C, "Operation", input, result putmessage "IP4", "PP4", result endprogram </pre>	

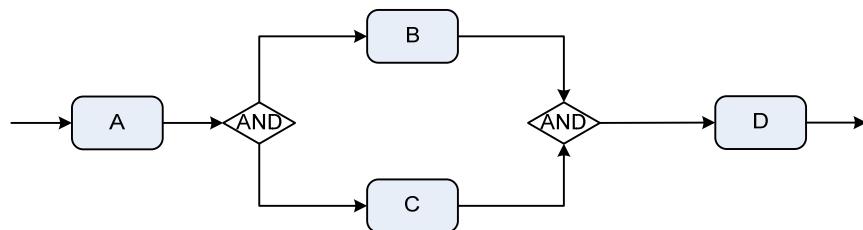
### Paralelno grananje i spajanje

Uzorak paralelnog grananja definira točku u tijeku izvođenja nakon koje se počinju paralelno izvoditi dvije grane tijeka izvođenja. Paralelne grane opisuju se proizvoljnim uzorkom tijeka izvođenja. Nakon završetka izvođenja obje paralelne grane tijeka izvođenja, nastavlja se izvođenje aktivnosti izlazne grane uzorka. Točka u kojoj se paralelne grane spajaju i izvođenje se nastavlja jednom granom naziva se točka sinkronizacije.

Uzorak paralelnog grananja još se označava nazivima I-granjanje, paralelno usmjeravanje ili *fork* granjanje. Naziv *fork* granjanje potekao je od imena naredbe stvaranja paralelnih procesa

u programskom jeziku C koji je postao općeprihvaćeni naziv stvaranja paralelnih procesa u programskim jezicima.

Primjer uzorka paralelnog grananja i spajanja je uzorak aktivnosti procesa potraživanja osiguranja. Nakon što osiguravajuća kuća primi zahtjev za isplatom osiguranja pokreću se dva paralelna procesa, provjera police osiguranja podnositelja zahtjeva i procjena stvarne štete dobra koje je osigurano. Tek nakon završetka paralelno pokrenutih procesa moguće je izvršiti aktivnost procjene zahtjeva za isplatu osiguranja.

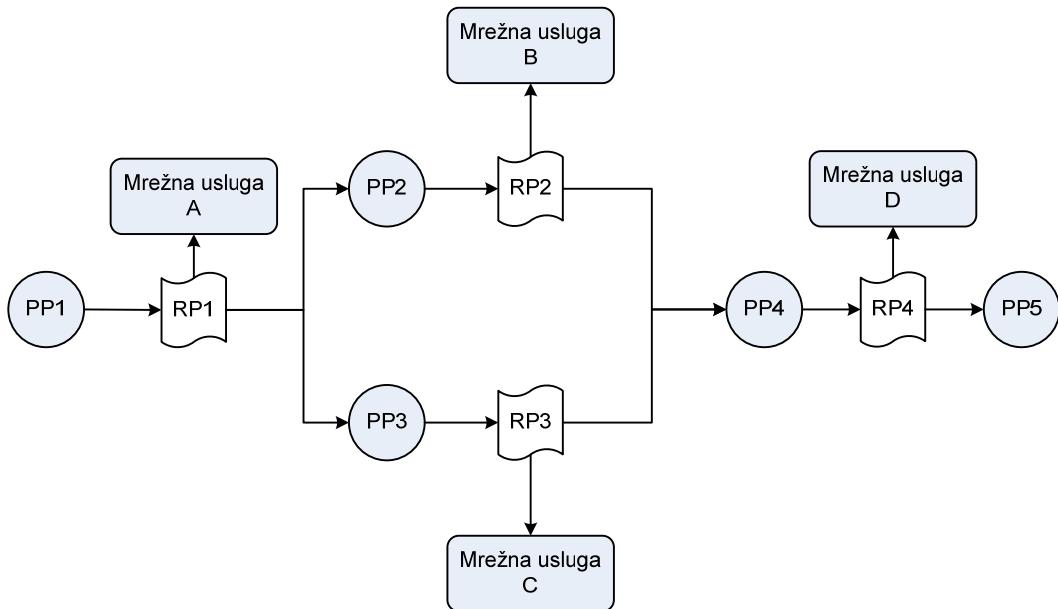


Slika 5.11 Uzorak paralelnog grananja i spajanja

Primjer na slici 5.11 prikazuje uzorak koji se sastoji od 4 aktivnosti od kojih se prvo izvodi aktivnost A, a nakon njenog završetka paralelno se izvode aktivnosti B i C. Nakon završetka izvođenja aktivnosti B i C, uzorak se nastavlja izvođenjem aktivnosti D. Prikazani primjer nije moguće ostvariti jednim raspodijeljenim programom, jer ne postoji naredba u jeziku SSCL kojom se stvaraju paralelni procesi.

Opisani uzorak moguće je ostvariti s četiri raspodijeljena programa prikazana na slici 5.12. Raspodijeljeni programi pozivaju jednu od mrežnih usluga koje obavljaju aktivnosti A, B, C i D, a njihova sinkronizacija se ostvaruje uporabom pet poštanskih pretinaca.

Raspodijeljeni programi započinju naredbom čitanja poruke iz poštanskog pretinca koja blokira nastavak izvođenja programa sve dok se ne pročita poruka iz poštanskog pretinca. Kako bi se pokrenulo paralelno izvođenje raspodijeljenih programa RP2 i RP3, raspodijeljeni program RP1 šalje dvije poruke, jednu u poštanski pretinac PP2 i jednu u poštanski pretinac PP3. Nakon poziva odgovarajućih mrežnih usluga, raspodijeljeni programi RP2 i RP3 šalju poruku u ulazni poštanski pretinac raspodijeljenog programa RP4. Raspodijeljeni program RP4 poziva mrežnu uslugu D tek nakon što pročita dvije poruke iz ulaznog poštanskog pretinca, odnosno nakon što završi izvođenje raspodijeljenih programa RP2 i RP3. Ostvarenja raspodijeljenih programa kojima je oblikovan uzorak paralelnog grananja i spajanja prikazana su u tablici 5.5.



Slika 5.12 Raspodijeljeno ostvarenje uzorka paralelnog grananja i spajanja

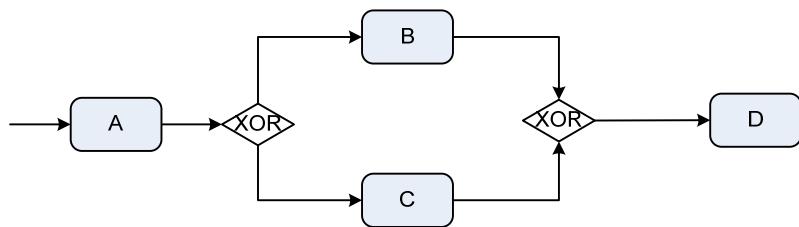
Tablica 5-5 Ispis raspodijeljenih programa uzorka paralelnog grananja i spajanja

Ispis programa RP1	Ispis programa RP2
<pre> program RP1 service A, "http://www.pie.fer.hr/serviceA" variable input, result getmessage "IP1", "PP1", input invoke A, "Operation1", input, result putmessage "IP2", "PP2", result putmessage "IP3", "PP3", result endprogram </pre>	<pre> program RP2 service B, "http://www.pie.fer.hr/serviceB" variable input, result getmessage "IP2", "PP2", input invoke B, "Process", input, result putmessage "IP4", "PP4", result endprogram </pre>
Ispis programa RP3	Ispis programa RP4
<pre> program RP3 service C, "http://www.pie.fer.hr/serviceC" variable input, result getmessage "IP3", "PP3", input invoke C, "Process", input, result putmessage "IP4", "PP4", result endprogram </pre>	<pre> program RP4 service D, "http://www.pie.fer.hr/serviceD" variable x, y, result getmessage "IP4", "PP4", x getmessage "IP4", "PP4", y invoke D, "Operation1", x, y, result putmessage "IP5", "PP5", result endprogram </pre>

### Isključivo grananje i spajanje

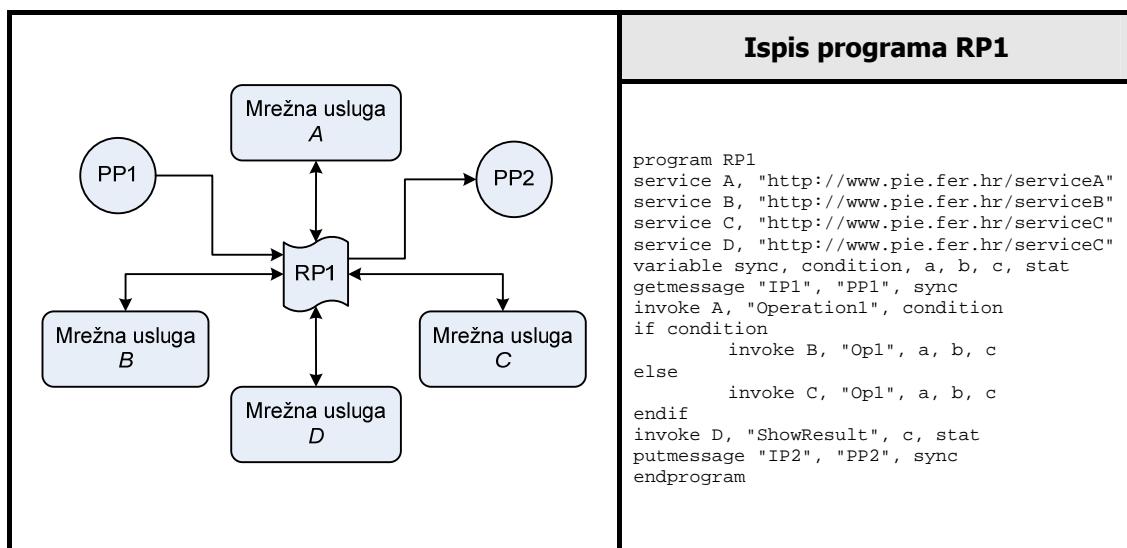
Isključivo grananje određuje točku u tijeku rada u kojoj se, na osnovi upravljačkih podataka, odabire jedna od mogućih grana izvođenja. Nakon izvođenja jedne od mogućih grana tijeka rada, izvođenje se nastavlja u točki spajanja izvođenjem aktivnosti izlazne grane. Primjer na slici 5.13 prikazuje uzorak isključivog grananja u kojem se nakon izvođenja aktivnosti A izvodi ili grana koja sadrži aktivnost B ili grana koja sadrži aktivnost C. Nakon završetka

izvođenja odabrane grane, izvođenje se nastavlja u točki spajanja pokretanjem izlazne grane uzorka, odnosno izvođenjem aktivnosti D.



Slika 5.13 Uzorak isključivog grananja i sinkronizacije

Uzorak isključivog grananja još se naziva i XOR grananje, uvjetno grananje, uvjetno usmjeravanje i točka odluke. Točka spajanja još se naziva i XOR spajanje i asinkrono spajanje.



Slika 5.14 Centralizirano ostvarenje uzorka isključivog grananja i sinkronizacije

Primjer procesa procjene osiguranja kojim je opisan prethodni uzorak može poslužiti i kao primjer isključivog grananja. Nakon što je izvedena aktivnost procjene zahtjeva za isplatom osiguranja, na osnovi rezultata procjene zahtjeva izvodi se ili aktivnost isplate određenog iznosa ili aktivnost slanja rješenja korisniku o neprihvaćanju zahtjeva. Nakon što se izvede jedna od dvije moguće aktivnosti, izvodi se aktivnost arhiviranja primljenog i obrađenog zahtjeva.

Za razliku od uzorka paralelnog grananja koji nije izravno ponuđen u programskom jeziku SSCL, uzorak isključivog grananja izravno je ponuđen uporabom naredbe *if-then--else*. Moguće centralizirano rješenje uporabom jednog raspodijeljenog programa prikazano je na slici 5.14 koja sadrži ispis raspodijeljenog programa RP1. Nakon čitanja poruke iz poštanskog pretinca PP1, poziva se operacija *Operation1* mrežne usluge A. Rezultat poziva spremi se u varijablu *condition* koja postaje uvjetna varijabla naredbe grananja. Ovisno o vrijednosti

variabile *condition* izvodi se poziv mrežne usluge *B* ili *C*. Nakon završetka jedne od grana, poziva se mrežna usluga *D*. Izvođenje raspodijeljenog programa završava slanjem poruke u poštanski pretinac *PP2*.

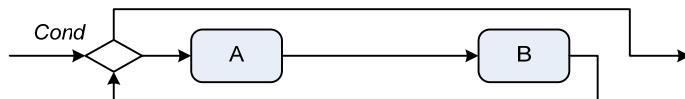
**Tablica 5-6 Ispis raspodijeljenih programa uzorka isključivog grananja i spajanja**

Ispis programa RP1	Ispis programa RP2
<pre>program RP1 service A, "http://www.pie.fer.hr/serviceA" variable result, input, condition getmessage "IP1", "PP1", input invoke A, "Operation1", input, result invoke A, "Operation2", input, condition if condition     putmessage "IP2", "PP2", result else     putmessage "IP3", "PP3", result endif endprogram</pre>	<pre>program RP2 service B, "http://www.pie.fer.hr/serviceB" variable input, result getmessage "IP2", "PP2", input invoke B, input, result putmessage "IP4", "PP4", result endprogram</pre>
Ispis programa RP3	Ispis programa RP4
<pre>program RP3 service C,"http://www.pie.fer.hr/serviceC" variable input, result getmessage "IP3", "PP3", input invoke C, input, result putmessage "IP4", "PP4", result endprogram</pre>	<pre>program RP4 service D, "http://www.pie.fer.hr/serviceD" variable input, result getmessage "IP4", "PP4", input invoke F, input, result putmessage "IP5", "PP5", result endprogram</pre>

Raspodijeljeno ostvarenje uzorka isključivog grananja i sinkronizacije moguće je ostvariti na sličan način kao i prethodno opisani uzorak paralelnog grananja i spajanja koji je prikazan na slici 5.12. Jedina nužna promjena u programskom ostvarenju uzorka je promjena raspodijeljenih programa RP1 i RP4 tako da RP1 ne šalje svaki put poruke u oba poštanska pretinca paralelnih grana, već ovisno o stanju uvjetne variabile šalje poruku u samo jedan poštanski pretinac. Kako se u ovom uzorku uvek izvodi samo jedna od paralelnih grana, raspodijeljeni program RP4 mora pročitati samo jednu poruku iz poštanskog pretinca PP4 i nakon toga može nastaviti svoje izvođenje pozivanjem mrežne usluge D. Ispis svih programa prikazan je u tablici 5.6.

### *Uzorak petlje*

Uzorak petlje (engl. *arbitrary cycles*) definira točku u tijeku izvođenja programa nakon koje se slijed poziva mrežnih usluga ponavlja određen broj puta. Na slici 5.15 prikazan je uzorak petlje u kojem se ponavlja slijed poziva mrežnih usluga A i B. Broj ponavljanja poziva mrežnih usluga A i B ovisi o vrijednosti upravljačke variabile *Cond*.



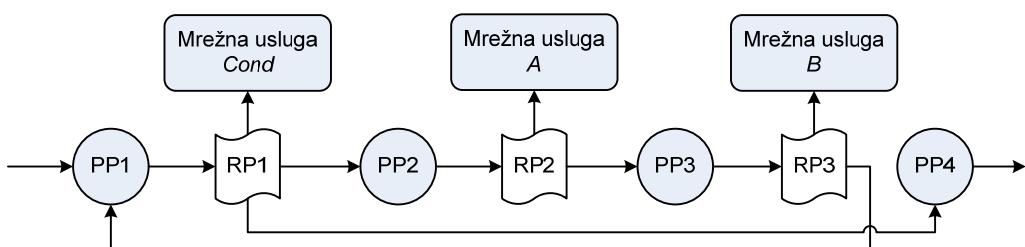
Slika 5.15 Uzorak petlje

Prikazani uzorak moguće je ostvariti centralizirano jednim raspodijeljenim programom uporabom naredbe *while*. Raspodijeljeni program kojim je ostvaren uzorak prikazan je u tablici 5.7. Slijed poziva mrežnih usluga *A* i *B* napisan je kao blok naredbi unutar uvjetne petlje. U svakom ponavljanju se, nakon slijeda poziva mrežnih usluga, ponovno izračunava vrijednost upravljačke varijable *condition*. Na osnovi izračunate vrijednosti donosi se odluka o ponovnom izvođenju bloka petlje ili o nastavku izvođenja raspodijeljenog programa, što je u ovom slučaju slanje poruke u poštanski pretinac PP2.

Tablica 5-7 Ispis programa centraliziranog ostvarenja uzorka petlje

Ispis programa RP1
<pre> program RP1 service A, "http://www.pie.fer.hr/serviceA" service B, "http://www.pie.fer.hr/serviceB" service Cond, "http://www.pie.fer.hr/Condition"  variable msg, result, status, condition  getmessage "IP1", "PP1", msg invoke A, "InitializeSystem", msg, condition  while condition     invoke A, "Calculate", msg, status     invoke B, "ProcessData", status, result     getmessage "IP1", "PP1", msg     invoke Cond, "Evaluate", msg, condition endwhile  putmessage "IP1", PP2, result endprogram </pre>

Osim centraliziranog ostvarenja, moguće je oblikovati i raspodijeljeno ostvarenje uzorka petlje. Jedno od mogućih raspodijeljenih ostvarenja prikazano je na slici 5.17. Prikazano raspodijeljeno ostvarenje sastoji se od tri raspodijeljena programa i četiri poštanska pretinca za međusobnu sinkronizaciju i komunikaciju raspodijeljenih programa.



Slika 5.16 Raspodijeljeno ostvarenje uzorka petlje

Raspodijeljeni program RP1 čita poruku iz poštanskog pretinca PP1. Vrijednost pročitane poruke šalje se kao parametar pri pozivu mrežne usluge *Cond*. Rezultat poziva mrežne usluge spremu se u uvjetnu varijablu *condition*. Vrijednost uvjetne variabile određuje da li se izvodi slijed aktivnosti petlje ili se izvođenje petlje prekida. Slijed aktivnosti petlje, odnosno pozivi mrežnih usluga A i B, pokreću se slanjem poruke u poštanski pretinac PP2, a izvođenje petlje prekida se slanjem poruke u poštanski pretinac PP4. Raspodijeljeni programi RP2 i RP3 imaju jednostavnu logiku i samo čitaju poruku iz ulaznog poštanskog pretinca, pozivaju određenu mrežnu uslugu i šalju rezultat u izlazni poštanski pretinac. Ispisi raspodijeljenih programa korištenih za ostvarenje uzorka petlje prikazani su u tablici 5.8.

**Tablica 5-8 Ispis raspodijeljenih programa uzorka petlje**

<b>Ispis programa RP1</b>
<pre>program RP1 service Cond, "http://www.pie.fer.hr/Condition" variable condition, msg  while "true"     getmessage "IP1", "PP1", msg     invoke Cond, "Evaluate", msg, condition     if condition         putmessage "IP2", "PP2", msg     else         putmessage "IP4", "PP4", msg     endif endwhile endprogram</pre>
<b>Ispis programa RP2</b>
<pre>program RP2 service A, "http://www.pie.fer.hr/serviceA" variable msg, result while "true"     getmessage "IP2", "PP2", msg     invoke A, "Calculate", msg, result     putmessage "IP3", "PP3", result endprogram</pre>
<b>Ispis programa RP3</b>
<pre>program RP3 service B, "http://www.pie.fer.hr/serviceB" variable msg, result while "true"     getmessage "IP3", "PP3", msg     invoke B, "ProcessData", msg, result     putmessage "IP1", "PP1", result endprogram</pre>

### 5.5.2 Napredni uzorci tijeka izvođenja

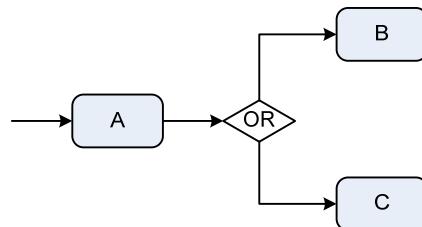
Napredne uzorke tijeka izvođenja čine nekoliko uzoraka višestrukog grananja te uzorak neuređenog slijednog izvođenja. Uzorci višestrukog grana razlikuju se prema načinu spajanja paralelnih grana u jedinstvenu granu kojom se nastavlja tijek izvođenja. Napredni uzorci nemaju potporu za izravno ostvarenje u većini sustava za upravljanje tijekom rada i

pripadajućim jezicima za opis tijeka rada. Međutim, navedeni uzorci su vrlo česti u svakodnevnim poslovnim procesima i zbog toga je vrlo značajno omogućiti njihovo ostvarenje. Svi napredni uzorci grananja i sinkronizacije ostvaruju se primjenom većeg broja raspodijeljenih programa i niti jedan uzorak nije moguće izravno ostvariti jednim raspodijeljenim programom napisanim SSCL programskim jezikom.

### *Općeniti uzorak višestrukog grananja i spajanja*

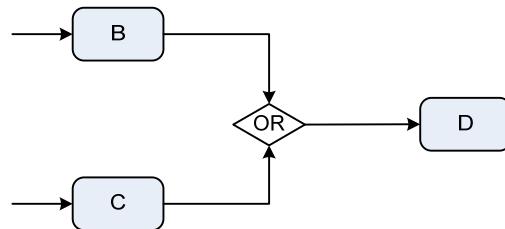
Prethodno opisani uzorak isključivog grananja i sinkronizacija podrazumijeva da se od više mogućih grana izvođenja uvijek odabere samo jedna grana. Često je u stvarnim situacijama potrebno omogućiti odabir više grana koje će se paralelno izvršiti, ali ne nužno svih mogućih grana kao što je bio slučaj kod uzorka paralelnog grananja i spajanja.

Višestruko grnanje je točka u programu u kojoj se, ovisno o stanju upravljačke varijable, odabire jedna ili više paralelnih grana izvođenja. Na slici 5.17 prikazan je najjednostavniji primjer višestrukog grananja u kojem se nakon obavljanja aktivnosti A, dolazi u točku višestrukog grananja gdje se na osnovi stanja upravljačke varijable odlučuje da li će se izvršiti aktivnost A, aktivnost B ili obje aktivnosti istovremeno. Za razliku od paralelnog grananja i isključivog grananja, kod uzorka višestrukog grananja ne može se tijekom oblikovanja programa znati koliko grana će postati aktivno nakon točke višestrukog grananja. Paralelno grnanje i isključivo grnanje su specijalani slučajevi višestrukog grananja.



Slika 5.17 Uzorak višestrukog grananja

Za razliku od paralelnog grananja i isključivog grananja, gdje je spajanje mogućih grana tijeka izvođenja jedinstveno definirano, u slučaju višestrukog spajanja postoji više načina spajanja mogućih grana izvođenja u jednu granu kojom se nastavlja izvođenje nakon grananja. Slika 5.18 prikazuje najjednostavniji uzorak višestrukog spajanja u kojem se dvije paralelne grane spajaju u jednu izlaznu granu. Ali već kod ovog najjednostavnijeg uzorka postavlja se pitanje da li se u točki spajanja treba čekati na završetak aktivnosti B, aktivnosti C ili na završetak obje aktivnosti.

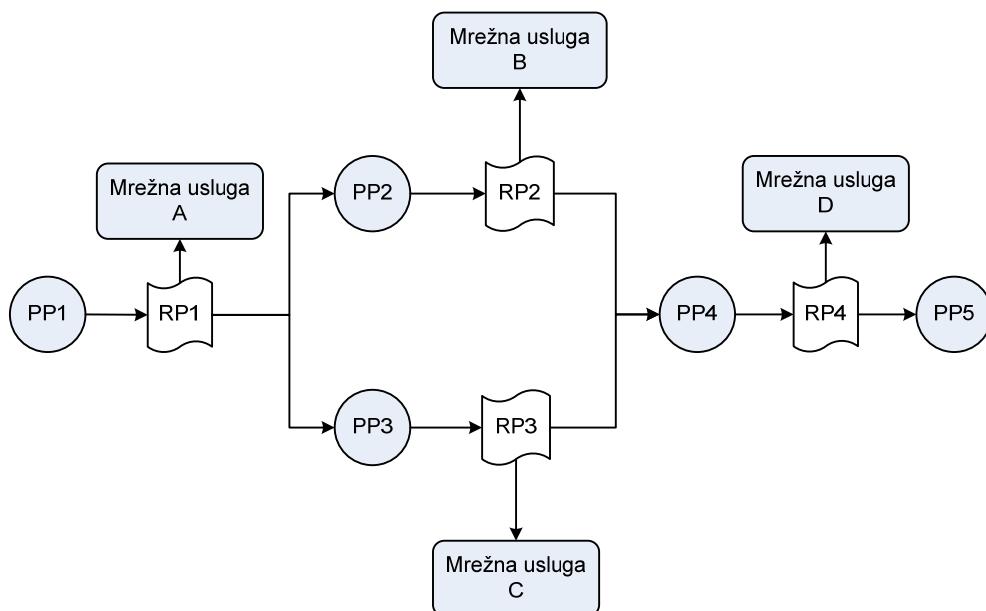


Slika 5.18 Uzorak višestrukog spajanja

Načini spajanja razlikuju se prema broju paralelnih grana na čiji završetak izvođenja se čeka prije nego se započne izvoditi aktivnost izlazne grane. Prema broju grana na koje se čeka u točki spajanja razlikuju se uzorci višestrukog spajanja, sinkronizacijskog spajanja, dekrementirajuće sinkronizacije i n-od-m spajanje.

### Višestruko spajanje

Uzorak višestrukog spajanja (engl. *multiple merge*) definira točku u tijeku rada u kojoj se izvođenje izlazne grane uzorka pokreće nakon završetka izvođenja svake pokrenute paralelne grane.



Slika 5.19 Ostvarenje uzorka višestrukog grananja i višestrukog spajanja

Uzorak spajanja paralelnih grana nastalih višestrukim grananjem moguće je ostvariti sustavom prikazanim na slici 5.19. Topologija programskog sustava je ista kao i za uzorce paralelnog i isključivog grananja, ali je razlika u ostvarenju raspodijeljenog programa RP1, kojim se pokreće izvođenje paralelnih grana i raspodijeljenog programa RP4, kojim se paralelne grane stapaju u jednu izlaznu granu.

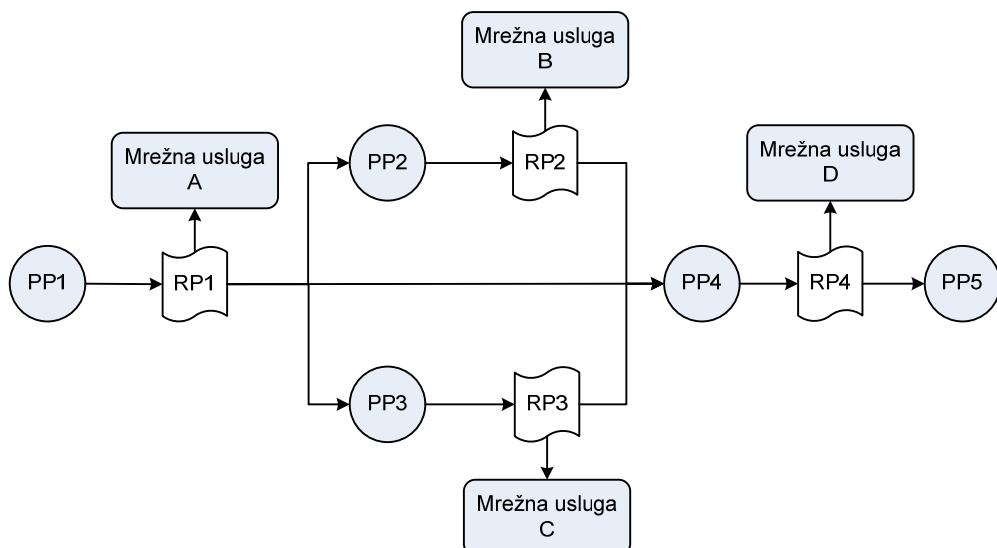
Raspodijeljeni program RP1, ovisno o stanju uvjetnih varijabli sustava, pokreće izvođenje jedne ili obje paralelne grane u kojima se pozivaju mrežne usluge B, odnosno C, slanjem poruke u odgovarajući poštanski pretinac. Raspodijeljeni program RP4 ostvaruje višestruko spajanje grana tako što za svaku pročitanu poruku iz poštanskog pretinca PP4 poziva mrežnu uslugu D i pošalje odgovarajuću poruku u izlazni poštanski pretinac PP5. Kako raspodijeljeni programi RP2 i RP3 šalju poruke u poštanski pretinac PP4 tek nakon poziva odgovarajuće mrežne usluge, raspodijeljeni program RP4 će po završetku izvođenja bilo koje od paralelnih grana pozvati mrežnu uslugu D i time ostvariti logiku višestrukog spajanja. Ispis programa prikazan je u tablici 5.9.

Tablica 5-9 Ispis programa kojima je ostvaren uzorak višestrukog grananja i spajanja

Ispis programa RP1	Ispis programa RP3
<pre>program RP1 service A, "http://www.pie.fer.hr/serviceA" variable input, result, cond1, cond2 getmessage "IP1", "PP1", input invoke A, "Process", input, result invoke A, "Evaluate1", input, cond1 invoke A, "Evaluate2", input, cond2 if cond1     putmessage "IP2", "PP2", result endif if cond2     putmessage "IP3", "PP3", result endif endprogram</pre>	<pre>program RP3 service C, "http://www.pie.fer.hr/serviceC" variable input, result getmessage "IP3", "PP3", input invoke C, "Process", input, result putmessage "IP4", "PP4", result endprogram</pre>
Ispis programa RP2	Ispis programa RP4
<pre>program RP2 service B, "http://www.pie.fer.hr/serviceB" variable input, result getmessage "IP2", "PP2", input invoke B, "Calculate", input, result putmessage "IP4", "PP4", result endprogram</pre>	<pre>program RP4 service D, "http://www.pie.fer.hr/serviceD" variable condition, input, result condition = "true" while condition     getmessage "IP4", "PP4", input     invoke D, "Compute", input, result     putmessage "IP5", "PP5", result enwhile endprogram</pre>

### Sinkronizacijsko spajanje

Za razliku od višestrukog spajanja, sinkronizacijsko spajanje (engl. *synchronizing merge*) definira spajanje u kojem se čeka na završetak svih paralelnih grana koje su pokrenute i tek nakon završetka svih aktivnih grana nastavlja izvođenje izlazne grane. Za primjer na slici 5.18 to znači da ako je pokrenuta samo jedna grana, onda je sinkronizacijsko spajanje jednako spajanju nakon isključivog grananja, a ako su pokrenute obje grane onda je sinkronizacijsko spajanje jednako spajanju nakon paralelnog grananja. U općenitom slučaju, kada postoji više od dvije paralelne grane, sinkronizacijsko spajanje potrebno je ostvariti tako da se iz točke višestrukog grananja u točku višestrukog spajanja pošalje broj paralelnih grana koje će se izvesti.



**Slika 5.20 Raspodijeljeno ostvarenje uzorka višestrukog grananja i sinkronizacijskog spajanja**

Slika 5.20 prikazuje ostvarenje uzorka višestrukog grananja i sinkronizacijskog spajanja za primjer sa slike 5.18, odnosno samo za dvije paralelne grane. Prikazano ostvarenje uzorka je općenito i moguće ga je primijeniti na uzorak s proizvoljnim brojem paralelnih grana. Raspodijeljeni program RP1 nakon čitanja poruke iz ulaznog poštanskog pretinca PP1 i poziva mrežne usluge A provjerava stanje upravljačkih varijabli. Na osnovi vrijednosti upravljačkih varijabli program donosi odluku o broju paralelnih grana koje će se pokrenuti i nakon toga šalje u poštanski pretinac PP4 poruku koja sadrži broj paralelnih grana koje će se pokrenuti. Nakon toga se pokreće izvođenje određenog broja paralelnih grana slanjem poruka u odgovarajuće poštanske pretinice. Raspodijeljeni program RP4 najprije iz poštanskog pretinca PP4 čita poruku koja sadrži broj pokrenutih paralelnih grana. Nakon toga u *while* petlji izvodi naredbu čitanja poruke iz poštanskog pretinca PP4 i naredbu poziva operacije *CreateMsg* usluge D. *While* petlja se izvodi po jednom za svaku pokrenutu granu.

Kako SSCL jezik ne definira naredbu petlje s unaprijed određenim brojem ponavljanja, za ostvarenje petlje s određenim brojem ponavljanja koristi se *while* petlja i operacija *ForEach* usluge *D*. Parametar operacije *ForEach* je broj ponavljanja *count*. Operacija *ForEach* vraća logičku vrijednost *true* ako je ulazni parametar veći od nule i smanjuje ga za jedan. Nakon što raspodijeljeni program *RP4* primi izlazne poruke iz svih pokrenutih paralelnih grana izvođenje uzorka završava pozivom operacije *Process* usluge *D* i slanjem rezultata izvođenja operacije u izlazni poštanski pretinac *PP5*. Ispis raspodijeljenih programa prikazan je u tablici 5.10.

Tablica 5-10 Ispis raspodijeljenih programa uzorka sinkronizacijskog spajanja

Ispis programa RP1	Ispis programa RP2
<pre>program RP1 service A, "http://www.pie.fer.hr/serviceA" variable input, result, cond1, cond2, count getmessage "IP1", "PP1", input invoke A, "Process", input, result invoke A, "Evaluate1", input, cond1 invoke A, "Evaluate2", input, cond2 invoke A, "Count", input, count  putmessage "IP4", "PP4", count if cond1     putmessage "IP2", "PP2", result endif if cond2     putmessage "IP3", "PP3", result endif endprogram</pre>	<pre>program RP2 service B, "http://www.pie.fer.hr/serviceB" variable input, result getmessage "IP2", "PP2", input invoke B, "Calculate", input, result putmessage "IP4", "PP4", result endprogram</pre>
Ispis programa RP3	Ispis programa RP4
<pre>program RP3 service C, "http://www.pie.fer.hr/serviceC" variable input, result getmessage "IP3", "PP3", input invoke C, "Process", input, result putmessage "IP4", "PP4", result endprogram</pre>	<pre>program RP4 service D, "http://www.pie.fer.hr/serviceD" variable input, result, temp, count getmessage "IP4", "PP4", count  while D, "ForEach", count     getmessage "IP4", "PP4", temp     invoke D, "CreateMsg", temp, input, input endwhile  invoke D, "Process", input, result putmessage "IP5", "PP5", result endprogram</pre>

### Dekrementirajuća sinkronizacija

Uzorak dekrementirajuće sinkronizacije (engl. *discriminator*) definira točku u procesu u kojoj se čeka na završetak izvođenja jedne od ulaznih paralelnih grana. Nakon što jedna od ulaznih grana završi, pokreće se izvođenje izlazne grane. Nakon pokretanja izlazne grane čeka se završetak preostalih aktivnih ulaznih grana, ali prilikom završetka pojedine ulazne grane ne pokreće se ponovno izvođenje izlazne grane.

Primjer korištenja višestrukog grananja i dekrementirajuće sinkronizacije je napredno pretraživanje baze podataka. Kako bi se ubrzalo složeno pretraživanje baza podataka, u isto

vrijeme se poziva više mrežnih usluga kojima se pretražuju nezavisne preslike baze podataka. Prvi odgovor koji stigne se prihvata i proces se nastavlja izvoditi, dok se ostali odgovori zanemaruju. Raspodijeljeno ostvarenje uzorka višestrukog grananja i dekrementirajuće sinkronizacije iste je topologije kao i ostvarenje uzorka višestrukog grananja i sinkronizacijskog spajanja prikazano na slici 5.20. Razlika je u ostvarenju raspodijeljenih programa, a njihov ispis je naveden u tablici 5.11.

**Tablica 5-11 Ispis raspodijeljenih programa uzorka dekrementirajuće sinkronizacije**

Ispis programa RP1	Ispis programa RP2
<pre>program RP1 service A, "http://www.pie.fer.hr/serviceA" variable input, result, cond1, cond2, count getmessage "IP1", "PP1", input invoke A, "Process", input, result invoke A, "Evaluate1", input, cond1 invoke A, "Evaluate2", input, cond2 invoke A, "Count", input, count  putmessage "IP4", "PP4", count if cond1     putmessage "IP2", "PP2", result endif if cond2     putmessage "IP3", "PP3", result endif endprogram</pre>	<pre>program RP2 service B, "http://www.pie.fer.hr/serviceB" variable input, result getmessage "IP2", "PP2", input invoke B, "Calculate", input, result putmessage "IP4", "PP4", result endprogram</pre>
Ispis programa RP3	Ispis programa RP4
<pre>program RP3 service C, "http://www.pie.fer.hr/serviceC" variable input, result getmessage "IP3", "PP3", input invoke C, "Process", input, result putmessage "IP4", "PP4", result endprogram</pre>	<pre>program RP4 service D, "http://www.pie.fer.hr/serviceD" variable input, result, temp, count  getmessage "IP4", "PP4", count getmessage "IP4", "PP4", input invoke D, "Process", input, result putmesagge "IP5", "PP5", result  while D, "ForEach", count     getmessage "IP4", "PP4", temp endwhile endprogram</pre>

### N od M spajanje

Dekrementirajuća sinkronizacija se jednostavno može generalizirati na *n-od-m spajanje* kojim se čeka na završetak *n* od ukupno *m* ulaznih grana i nakon toga započinje izvođenje izlazne grane, odnosno poziva mrežnu uslugu D. Razlika u odnosu na prethodno opisano ostvarenje uzorka dekrementirajuće sinkronizacije je jedino u ostvarenju raspodijeljenih programa RP1 i RP4, a ispis spomenutih raspodijeljenih programa je prikazan u tablici 5.12.

Raspodijeljeni program RP1 šalje dvije poruke u poštanski pretinac PP4. Prva poruka sadrži informaciju o broju grana (*n*) koje moraju završiti prije nego se nastavi izvođenje uzorka, odnosno prije nego se pozove operacija *Process* usluge D. Druga poruka sadrži broj grana koje se naknadno moraju čekati (*m-n*), odnosno razliku između ukupnog broja pokrenutih grana i broja grana na čiji završetak izvođenja se čeka.

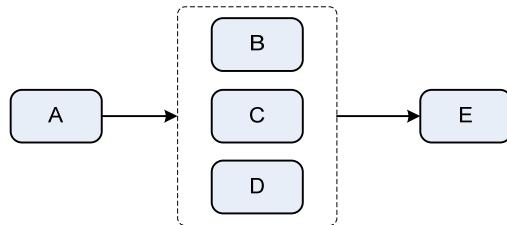
Raspodijeljeni program RP4 čita iz poštanskog pretinca dvije poruke koje je poslao raspodijeljeni program RP1. Nakon toga čeka na završetak  $n$  grana. Rezultat izvođenja svake od  $n$  grana spaja se u jedinstvenu poruku pozivom pomoćne operacije *CreateMsg* mrežne usluge D. Poruka koja sadrži rezultate izvođenja svih  $n$  grana predaje se kao ulazni parametar operaciji *Process* mrežne usluge D, čijim pozivom se pokreće nastavak složene usluge. Nakon toga, raspodijeljeni program čeka na završetak preostalih pokrenutih grana uzorka. Broj grana na koje se čeka određen je drugom porukom pročitanom iz poštanskog pretinca PP4. Na završetak se čeka čitanjem odgovarajućeg broja poruka iz poštanskog pretinca PP4.

Tablica 5-12 Djelomično ostvarenje uzorka višestrukog grananja i  $n$ -od- $m$  spajanja

Ispis programa RP1	Ispis programa RP4
<pre> program RP1 service A, "http://www.pie.fer.hr/serviceA" variable input, result, cond1, cond2 variable count1, count2 getmessage "IP1", "PP1", input invoke A, "Process", input, result invoke A, "Evaluate1", input, cond1 invoke A, "Evaluate2", input, cond2 invoke A, "Count1", input, count1 invoke A, "Count2", input, count2  putmessage "IP4", "PP4", count1 putmessage "IP4", "PP4", count2 if cond1     putmessage "IP2", "PP2", result endif if cond2     putmessage "IP3", "PP3", result endif endprogram </pre>	<pre> program RP4 service D, "http://www.pie.fer.hr/serviceD" variable input, result, temp, count1, count2 getmessage "IP4", "PP4", count1 getmessage "IP4", "PP4", count2 while D, "ForEach", count1     getmessage "IP4", "PP4", temp     invoke D, "CreateMsg", temp, input endwhile  invoke D, "Process", input, result putmessage "IP5", "PP5", result  while D, "ForEach", count2     getmessage "IP4", "PP4", temp endwhile endprogram </pre>

### Neuređeno slijedno izvođenje

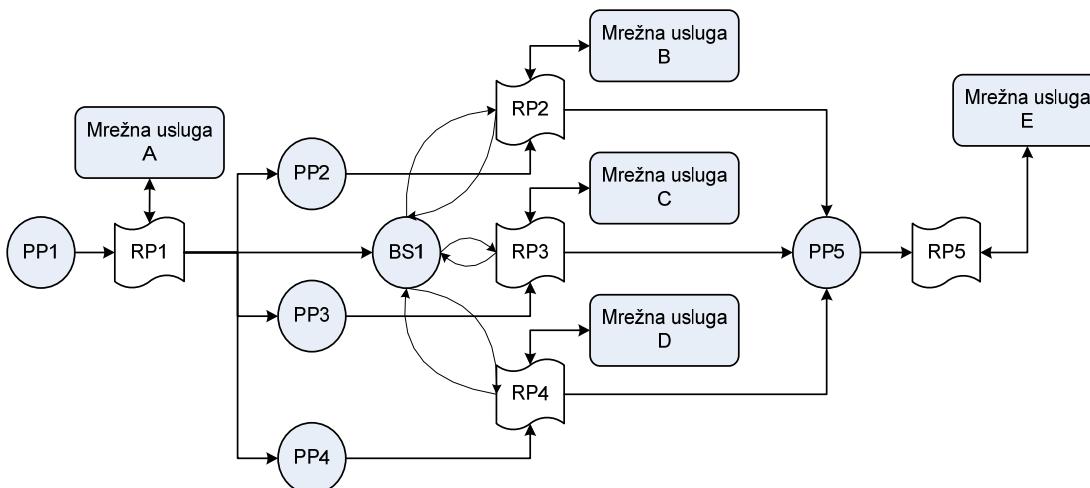
Uzorak neuređenog slijednog izvođenja (engl. *interleaved parallel routing, unordered sequence*) definira se skupom uzoraka tijeka izvođenja koji se izvode proizvoljnim redoslijedom. Bilo koji uzorak iz skupa izvodi se točno jedanput. Primjer uzorka slijednog neuređenog izvođenja prikazan je na slici 5.21. Skup uzoraka koji se izvode proizvoljnim redoslijedom su jednostavnii uzorci i sastoje se od poziva jedne mrežne usluge. Nakon što se izvede poziv mrežne usluge A, izvođenje programa se nastavlja slijednjim izvođenjem poziva mrežnih usluga B, C i D proizvoljnim redoslijedom. Nakon što se izvedu pozivi sve tri usluge, izvođenje se nastavlja pozivom usluge E.



Slika 5.21 Uzorak neuređenog slijednog izvođenja

Primjer uporabe uzorka neuređenog slijednog izvođenja je operacija obračuna kamata i operacija izračuna troška kreditne kartice koje banka izvodi na kraju svakog mjeseca. Na kraju svake godine, banka izvodi dvije operacije *dodaj\_iznos\_kamate* i *trošak\_kreditne\_kartice* kojima mijenja stanje računa svakog klijenta. Navedene operacije moguće je izvesti proizvoljnim redoslijedom, ali se moraju izvesti samo jednom i ne mogu se izvesti u isto vrijeme jer obje operacije mijenjaju isti račun.

Uzorak slijednog neuređenog obilaska paralelnih grana moguće je ostvariti raspodijeljenim sustavom prikazanim na slici 5.22. Predloženo ostvarenje u potpunosti omogućuje proizvoljan redoslijed izvođenja paralelnih grana, jer se raspodijeljeni programi RP3, RP4 i RP5 natječe za zauzimanje binarnog semafora BS1. Redoslijed zauzimanja binarnog semafora nije moguće odrediti za vrijeme oblikovanja sustava i zbog toga se raspodijeljeni programi izvode u neuređenom slijedu.



Slika 5.22 Raspodijeljeno ostvarenje uzorka neuređenog slijednog izvođenja

Ispis raspodijeljenih programa prikazan je u tablici 5.13. Zbog jednostavnosti je pretpostavljeno da raspodijeljeni program RP1 ne može pročitati novu poruku prije nego što se izvede ostatak uzorka, odnosno dok ne završi izvođenje raspodijeljenog programa RP5. Ovo ograničenje je moguće jednostavno riješiti dodavanjem još jednog binarnog semafora, ali to nema utjecaja na uzorak pa je zbog preglednosti izostavljeno.

**Tablica 5-13 Programsko ostvarenje uzorka neuređenog slijednog izvođenja**

<b>Ispis programa RP1</b>	<b>Ispis programa RP2</b>
<pre>program RP1 service A, "http://www.pie.fer.hr/serviceA" variable result, condition condition = "true" while condition     getmessage "IP1", "PP1"     invoke A, "GetData", result     putmessage "IP2", "PP2", result     putmessage "IP3", "PP3", result     putmessage "IP4", "PP4", result endwhile endprogram</pre>	<pre>program RP2 service B, "http://www.pie.fer.hr/serviceB" variable input, result, condition condition = "true" while condition     getmessage "IP2", "PP2", input     obtainBinarySemaphore "IP1", "BS1"     invoke B, "Process", input, result     putmessage "IP5", "PP5", result     releaseBinarySemaphore "IP1", "BS1" endprogram</pre>
<b>Ispis programa RP3</b>	<b>Ispis programa RP4</b>
<pre>program RP3 service C, "http://www.pie.fer.hr/serviceC" variable input, result, condition condition = "true" while condition     getmessage "IP3", "PP3", input     obtainBinarySemaphore "IP1", "BS1"     invoke C, "Process", input, result     putmessage "IP5", "PP5", result     releaseBinarySemaphore "IP1", "BS1" endprogram</pre>	<pre>program RP4 service D, "http://www.pie.fer.hr/serviceD" variable input, result, condition condition = "true" while condition     getmessage "IP4", "PP4", input     obtainBinarySemaphore "IP1", "BS1"     invoke D, "Process", input, result     putmessage "IP5", "PP5", result     releaseBinarySemaphore "IP1", "BS1" endprogram</pre>
<b>Ispis programa RP5</b>	
<pre>program RP5 service E, "http://www.pie.fer.hr/serviceE" variable condition, a, b, c, result condition = "true" while condition     getmessage "IP5", "PP5", a     getmessage "IP5", "PP5", b     getmessage "IP5", "PP5", c     invoke E, "Process", a, b, c, result endwhile endprogram</pre>	

## 6 Raspodijeljeni prevoditelj jezika SSCL

Korisnički jezik SSCL je u usporedbi s drugim jezicima za kompoziciju mrežnih usluga oblikovan kao programski jezik visoke razine. Kao i ostale programe pisane u programskim jezicima visoke razine, programe pisane u jeziku SSCL potrebno je prevesti u izvodljivi program. U sklopu ovog rada ostvaren je raspodijeljeni prevoditelj programa napisanih u korisničkom jeziku SSCL u programe napisane u jeziku CL.

Jezik CL je na XML-u zasnovan jezik za oblikovanje raspodijeljenih programa i izgradnju kompozicije mrežnih usluga prema programskom modelu zasnovanom na uslugama. Raspodijeljeni programi napisani u jeziku CL su potpune programske cjeline koje je moguće izvesti u raspodijeljenom sustavu za izvođenje raspodijeljenih programa [77].

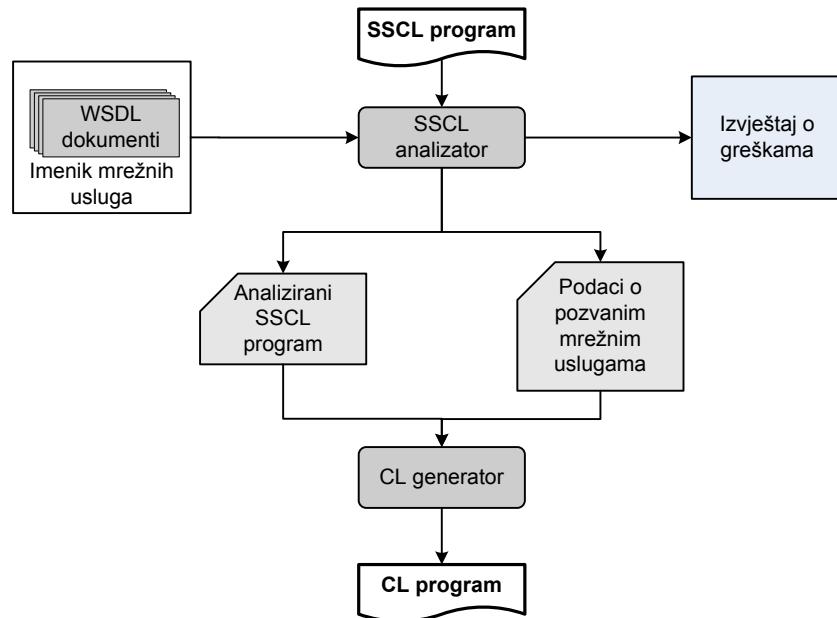
Raspodijeljeni sustav prevođenja programa napisanih u jeziku SSCL oblikovan je prema načelima arhitekture zasnovane na uslugama i sastoji se od samostalnih prevoditelja SSCL programa i raspoređivača programa. Prevoditelj raspodijeljenih programa ostvaruje proces analize izvornog raspodijeljenog programa napisanog u SSCL jeziku i proces generiranja ciljnog programa napisanog u jeziku CL. Raspoređivač programa je ulazna točka sustava za prevođenje i ostvaruje proces privremenog spremanja raspodijeljenih programa te njihovog raspoređivanja na raspoložive prevoditelje SSCL programa.

Sustav prevođenja je programski ostvaren u .Net razvojnoj okolini uporabom *Web Services* tehnologija. Tako oblikovani sustav pogodan je za izvođenje u raznorodnim raspodijeljenim okolinama.

### 6.1 Proces prevođenja

Proces prevođenja sastoji se od analize izvornog SSCL programa i generiranja ciljnog CL programa. Analizator SSCL programa kao ulaz prima izvorni SSCL program, provjerava njegovu ispravnost, dohvaća opise mrežnih usluga suradnika i ispisuje informacije o pogreškama u programu. Ako u analiziranom programu nije pronađena pogreška, analizator stvara interni hijerarhijski zapis ispravnog SSCL programa i dodatne strukture podataka s opisima mrežnih usluga suradnika SSCL programa. Stvoreni zapis analiziranog SSCL programa prilagođen je procesu generiranja ciljnog CL programa, a strukture podataka s opisima mrežnih usluga suradnika omogućuju jednostavan pristup podacima iz WSDL dokumenata nužnim za generiranje CL programa.

Generator CL programa povezuje generirani interni zapis SSCL programa s podacima o sučeljima mrežnih usluga te generira ciljni CL program. Osnovni model prevođenja prikazan je na slici 6.1.



Slika 6.1 Osnovni model procesa prevodenja SSCL programa

### 6.1.1 Analiza izvornog SSCL programa

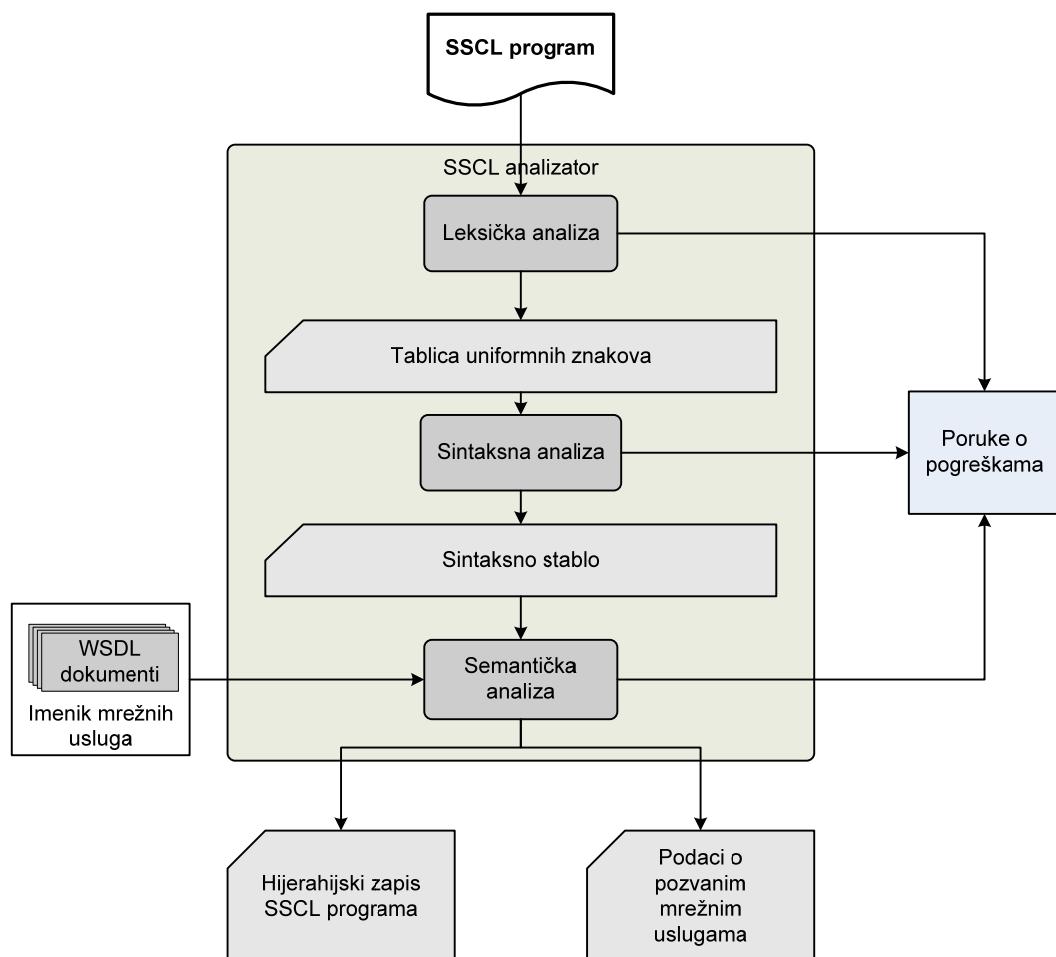
Proces analize izvornog SSCL programa sastoji se od tri faze prikazane na slici 6.2, leksičke analize, sintaksne analize i semantičke analize. Sve tri faze analize programa ostvarene su kao zaseban prolaz jezičnog procesora. Osnovna zadaća analizatora je provjera leksičke, sintaksne i semantičke ispravnosti izvornog programa i transformacija izvornog zapisa programa u strukture podataka potrebne za učinkovito generiranje ciljnog CL programa. Osnovne strukture podataka koje stvara analizator SSCL programa su unutarnji zapis analiziranog izvornog programa i strukture podataka koje sadrže WSDL opise mrežnih usluga naslovljenih u izvornom SSCL programu.

#### *Leksička analiza*

Tijekom leksičke analize znakovi izvornog SSCL programa grupiraju se u leksičke jedinice i provjeravaju se leksička pravila jezika. Ulaz u leksički analizator je izvorni SSCL program, a izlaz je tablica uniformnih znakova. Tablica uniformnih znakova sadrži slijed elemenata koji se sastoji od izvornog zapisa leksičke jedinice i razreda leksičke jedinice.

Leksički analizator slijedno učitava znak po znak izvornog programa, izbacuje nepotrebne znakove, a preostale znakove grupira u leksičke jedinice. Nepotrebni znakovi koji se izbacuju tijekom leksičke analize su razmaci, tabulatori i oznake kraja reda. Grupiranje znakova ulaznog programa u leksičke jedinice provodi se na osnovni leksičkih pravila zadanih

regularnim izrazima. Formalna definicija leksičkih pravila jezika SSCL prikazana je u dodatku A. Središnji algoritam leksičkog analizatora je algoritam određivanja razreda leksičke jedinice. Razredi leksičkih jedinki koje prepoznaje leksički analizator jezika SSCL su ključne riječi, operatori, specijalni znakovi, pokazatelje i konstante.



Slika 6.2 Proces analize izvornog SSCL programa

Ključne riječi jezika SSCL su leksičke jedinice koje tijekom sintaksne analize jedinstveno određuju početak i kraj sintaksne cjeline jezika SSCL. Ključne riječi jezika SSCL navedene su u tablici 6.1. Pokazatelji u jeziku SSCL koriste se za imena varijabli i lokalna imena usluga. Pokazatelji su leksičke jedinice koje se sastoje od niza slova i brojeva koji nužno mora započeti slovom. Prema definiciji jezika SSCL, svi podaci su predstavljeni istim tipom teksutalnih podataka. Iz tog razloga, jedine konstante koje se definiraju u SSCL programu su tekstualne konstante koje su u izvornom programu ograničene znakovima navoda.

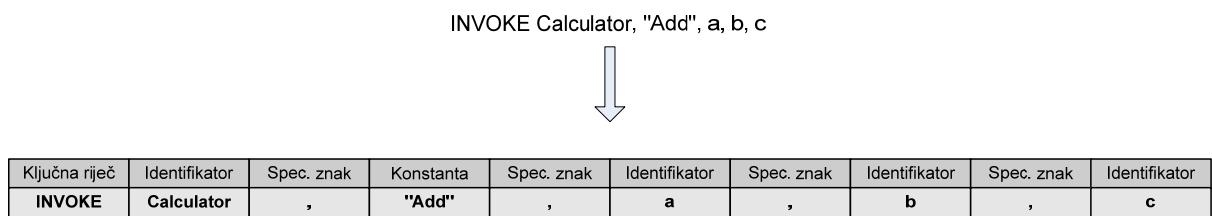
Jedini operator definiran u jeziku SSCL je operator pridruživanja ' = ' koji se koristi za pridruživanje vrijednosti varijablama. Jezik SSCL definira specijalni znak ' , ' koji se koristi za razdvajanje niza parametara naredbi jezika SSCL i specijalni znak ' " ' koji se koristi za ograničavanje konstanti.

**Tablica 6-1 Ključne riječi jezika SSCL**

program	destroycountingsemaphore
endprogram	obtaincountingsemaphore
service	releasecountingsemaphore
invoke	createmailbox
if	destroymailbox
elseif	putmessage
else	getmessage
endif	createeventchannel
while	destroyeventchannel
endwhile	subscribe
createbinarysemaphore	unsubscribe
destroybinarysemaphore	publish
obtainbinarysemaphore	republish
releasebinarysemaphore	unpublish
createcountingsemaphore	variable

Ako leksičku jedinku nije moguće svrstati niti u jedan razred leksičkih jedinki onda leksički analizator javlja poruku o pogrešci. Poruka o pogrešci sadrži niz izvornog programa koji nije bilo moguće svrstati u niti jedan razred, te informacije o mjestu na kojem je pogreška pronađena u izvornom programu.

Na slici 6.3 prikazan je isječak programa napisanog u jeziku SSCL i rezultat leksičke analize tog isječka programa. Prikazani isječak programa sadrži naredbu za poziv operacije *Add* mrežne usluge *Calculator*. Tijekom leksičke analize prepoznaje se niz leksičkih jedinki od kojih je prva jedinka ključna riječ *invoke*, nakon koje slijedi niz pokazatelja i konstanti odvojenih specijalnim znakom ','.



**Slika 6.3 Primjer leksičke analize**

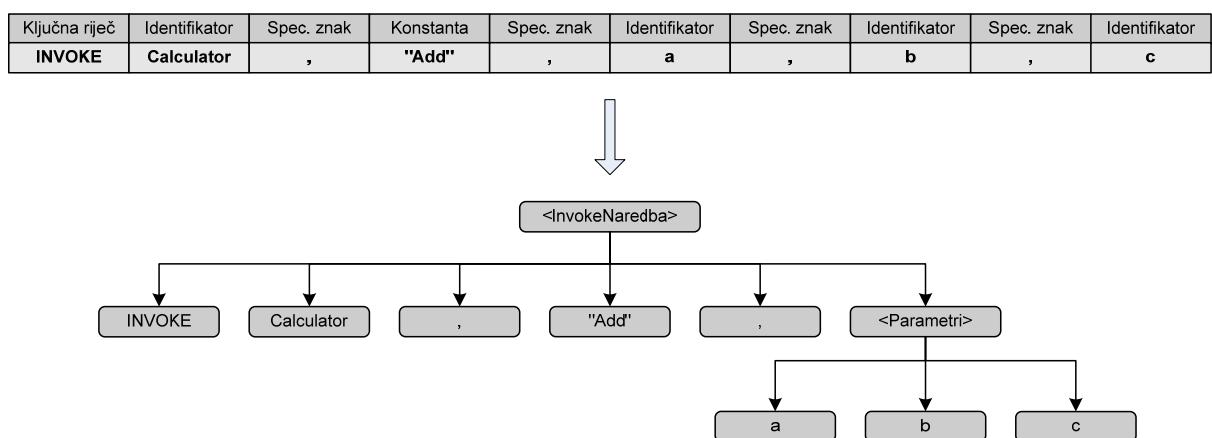
### Sintaksna analiza

Ulaz u sintaksni analizator je tablica uniformnih znakova stvorena tijekom leksičke analize. Sintaksni analizator grupira leksičke jedinke u sintaksne cjeline na osnovi sintaksnih pravila

jezika SSCL, opisanih kontekstno neovisnom gramatikom. Gramatika koja opisuje sintaksna pravila jezika SSCL prikazan je u dodatku B. Grupiranjem leksičkih jedinki u sintaksne cjeline, sintaksni analizator gradi hijerarhijsku strukturu izvornog programa koja se spremi u obliku sintaksnog stabla. Sintaksno stablo je izlazna podatkovna struktura sintaksnog analizatora.

Ako niz leksičkih jedinki nije u skladu sa sintaksnim pravilima jezika, onda sintaksni analizator generira poruku o pogrešci koja sadrži zapis leksičke jedinke koja je uzrokovala pogrešku i informacije o položaju leksičke jedinke u izvornom programu. Poruka o pogrešci sadrži i informacije o očekivanoj leksičkoj jedinci na mjestu pogreške.

Sve naredbe jezika SSCL su sintaksne cjeline jedinstveno određene ključnim riječima. Naredbe jezika SSCL dijele se na jednostavne i složene naredbe. Jednostavne naredbe su određene ključnom riječi koja se nalazi na početku naredbe, dok su složene naredbe ograničene s ključnom riječi na početku i na kraju naredbe. Tijekom sintaksne analize jednostavne naredbe, analizator na osnovi pročitane ključne riječi određuje koliko se leksičkih jedinki grupira u sintaksnu cjelinu. Osim broja leksičkih jedinki, sintaksni analizator provjerava razrede leksičkih jedinki koje se grupiraju u naredbu. Primjer jednostavne naredbe je naredba poziva mrežne usluge koja započinje ključnom riječi *invoke*. Tijekom sintaksne analize složene naredbe analizator grupira sve sintaksne cjeline između početne i završne ključne riječi u sintaksnu strukturu složene naredbe. Primjer složene naredbe jezika SSCL je naredba uvjetnog ponavljanja koja sadrži sve naredbe koje se u izvornom programu nalaze između ključnih riječi *while* i *endwhile*.



Slika 6.4 Primjer sintaksne analize

Jedina naredba jezika SSCL koja ne započinje ključnom riječi je naredba pridruživanja. Naredba pridruživanja na prvom mjestu sadrži pokazatelj varijable, nakon kojeg slijedi

operator pridruživanja, a nakon operatora pridruživanja slijedi pokazatelj varijable ili konstanta.

Primjer procesa sintaksne analize jednostavne naredbe za poziv mrežne usluge prikazan je na slici 6.4. Slika prikazuje ulazni niz leksičkih jedinki i sintaksno stablo generirano procesom sintaksne analize.

### Semantička analiza

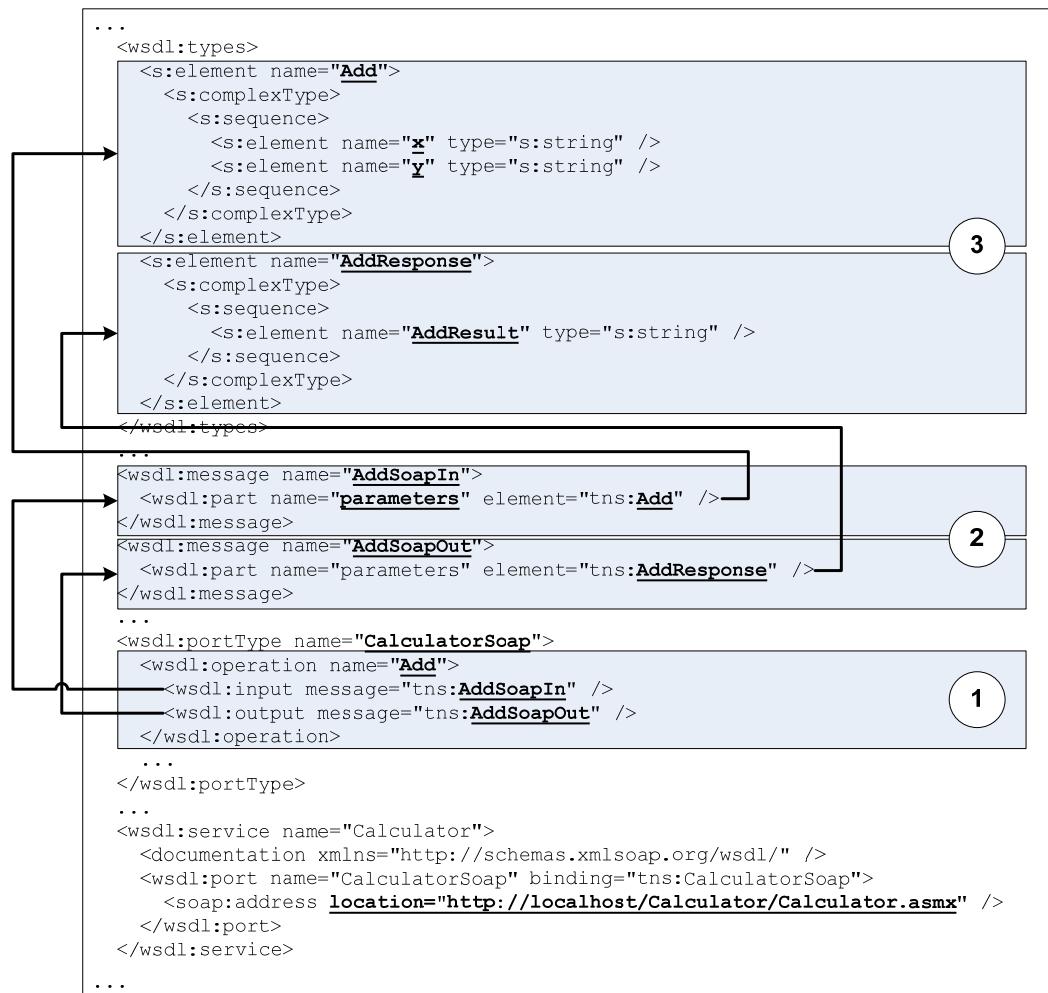
Zbog jednostavnosti SSCL jezika i malog broja naredbi za operacije s podacima, semantička pravila jezika su vrlo jednostavna. Semantička pravila se svode na provjere da li su varijable korištene u programu prethodno deklarirane i da li su korišteni pokazatelji lokalnih imena mrežnih usluga definirani naredbama *service*. Složenija provjera tijekom semantičke analize je provjera semantike naredbe poziva operacije mrežne usluge. Tijekom semantičke analize naredbe *invoke* potrebno je dohvatiti WSDL opis sučelja mrežne usluge i provjeriti da li su parametri naredbe ispravno napisani.

Semantička analiza ispravnosti provodi se nad sintaksnim stablom. Izlaz iz procesa semantičke analize je sintaksno stablo prošireno semantičkim svojstvima. Osim proširenja sintaksnog stabla, dio podataka prikupljen tijekom procesa semantičke analize spremi se u dodatne strukture podataka koje koristi generator ciljnog programa.

Semantička analiza opisana je na primjeru naredbe *invoke* čija je sintaksna struktura prikazana na slici 6.4. Na osnovi pokazatelja mrežne usluge *Calculator*, semantički analizator dohvaća WSDL opis mrežne usluge. Sažetak WSDL dokumenta s opisom mrežne usluge *Calculator* prikazan je na slici 6.5. Prikazani sažetak sadrži samo one dijelove WSDL dokumenta koji se koriste tijekom semantičke analize navedene naredbe, dok su ostali dijelovi zbog preglednosti izbačeni.

Prvi korak analize je provjera da li WSDL dokument sadrži definiranu operaciju "Add" čije je ime navedeno u naredbi poziva mrežne usluge. Ovaj dio analize prikazan je korakom (1) na slici 6.5. Ako je operacija s traženim imenom dio sučelja mrežne usluge, onda se provjerava da li broj parametara operacije odgovara broju parametara naredbe *invoke*. Ako broj parametara naredbe *invoke* ne odgovara broju parametara definiranih WSDL dokumentom, onda se generira poruka o semantičkoj pogrešci. Određivanje broja parametara operacije opisane u WSDL dokumentu obavlja se u dva koraka. U prvom koraku (2) se na osnovi imena ulazne i izlazne SOAP poruke operacije pronalazi definicija poruka. Struktura ulazne i izlazne SOAP poruke određena je tipom elementa koji je naveden unutar definicije poruke. U drugom koraku (3) se na osnovi imena elementa pronalazi XML Schema kojom je definiran tip elementa. Konačno, na osnovi XML Scheme određuje se broj ulaznih i izlaznih parametara operacije mrežne usluge. U navedenom primjeru, operacija *Add* ima dva ulazna parametra i

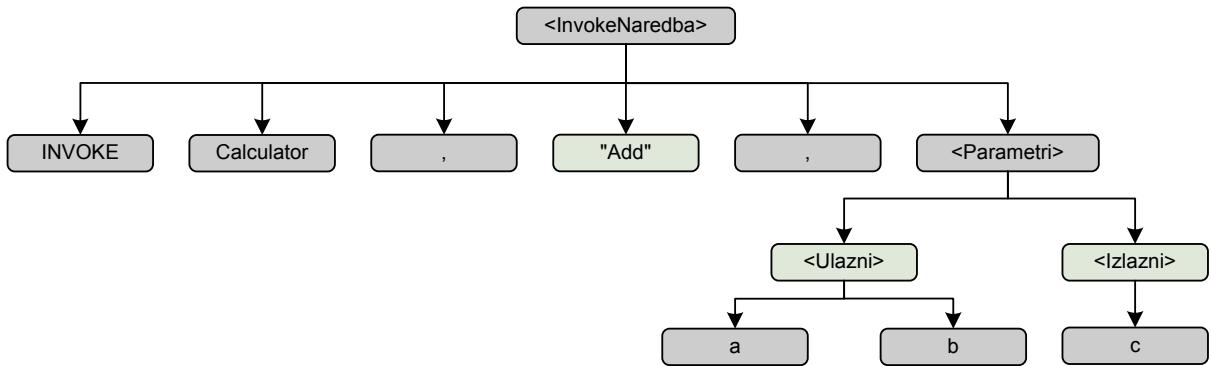
jedan izlazni. Stoga, ako naredba *invoke* sadrži tri parametra onda je semantički ispravna, u protivnom se dojavljuje poruka o pogrešci.



Slika 6.5 Parsiranje WSDL dokumenta tijekom semantičke analize INVOKE naredbe

Nakon provjere WSDL dokumenta, semantički analizator proširuje sintaksno stablo naredbe *invoke* i dijeli listu parametara na listu ulaznih parametara i listu izlaznih parametara.

Na slici 6.6. prikazan je primjer hijerarhijskog stabla proširenog semantičkim svojstvima. Prikazano stablo je generirano iz sintaksnog stabla naredbe *invoke* prikazanog na slici 6.4. Osim prikazanog stabla, izlaz procesa semantičke analize su i strukture podataka koje sadrže WSDL opise mrežnih usluga spremljene u strukture podataka prilagođene procesu generiranja ciljnog programa. Navedene strukture podataka omogućuju generatoru ciljnog programa jednostavan i učinkovit pristup dijelovima WSDL opisa usluga potrebnim u procesu generiranja kôda.

Slika 6.6 Stablo naredbe *invoke* prošireno semantičkim svojstvima

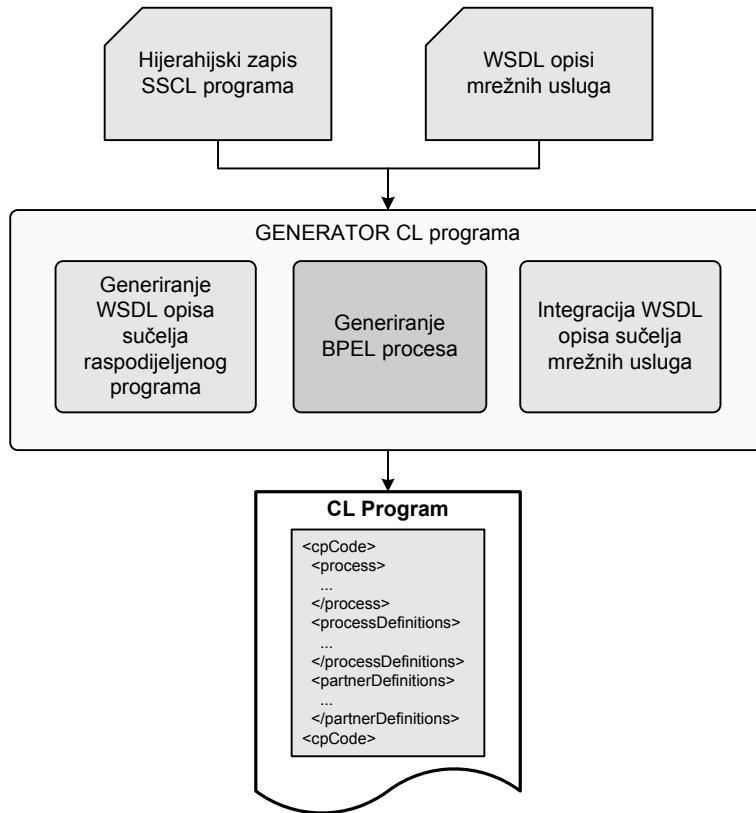
### 6.1.2 Generiranje ciljnog CL programa

Nakon provjere ispravnosti izvornog programa i stvaranja unutarnjih struktura podataka koje sadrže hijerarhijski zapis izvornog programa i WSDL opise sučelja mrežnih usluga suradnika, generator ciljnog programa generira program napisan u jeziku CL.

Ciljni jezik CL, opisan u odjeljku 3.4.6, sastoji se od tri cjeline, opisa BPEL procesa, WSDL opisa sučelja raspodijeljenog programa i WSDL opisa sučelja mrežnih usluga suradnika. U skladu sa strukturom CL jezika, proces generiranja CL programa podijeljen je u tri funkcionalne cjeline prikazane na slici 6.7, generiranje BPEL procesa, generiranje opisa sučelja raspodijeljenog programa i objedinjavanje opisa sučelja mrežnih usluga suradnika.

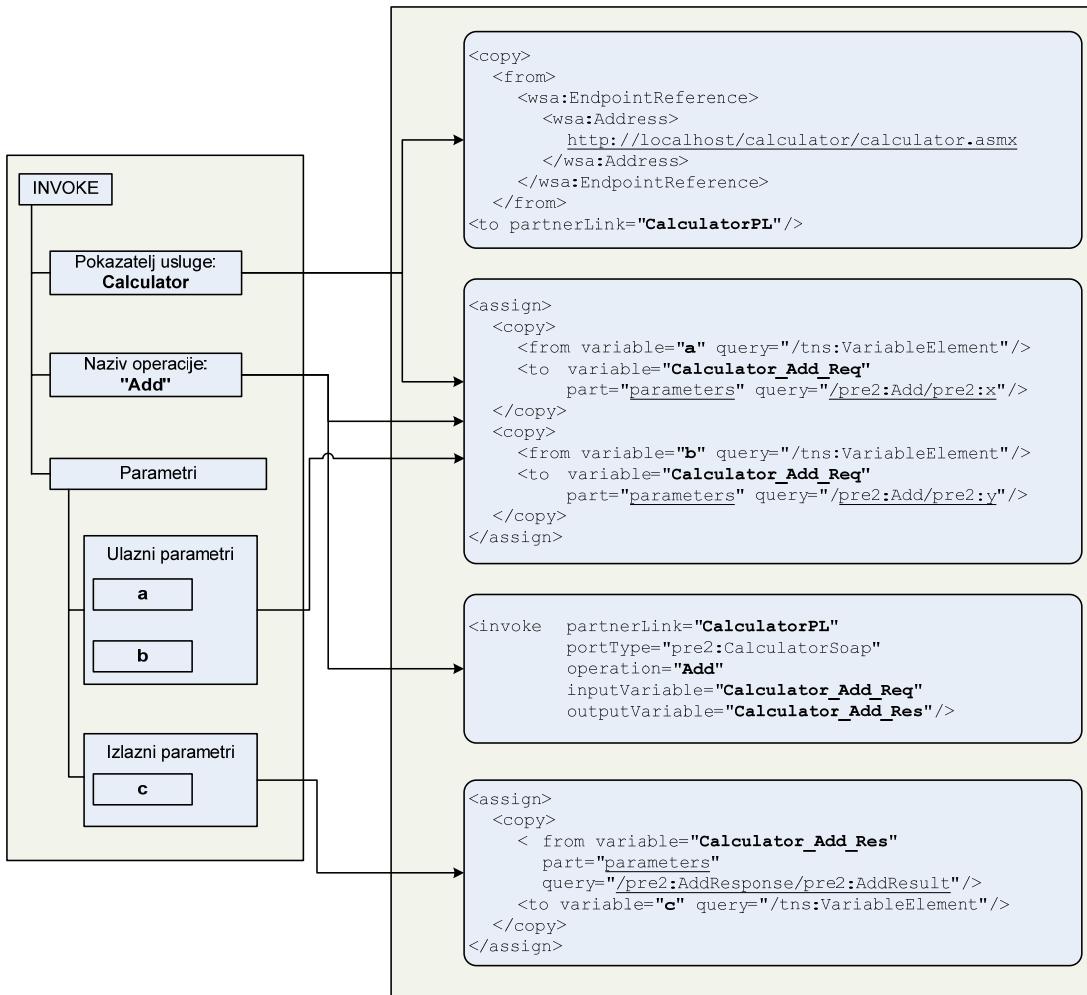
#### Generiranje BPEL procesa

Središnji proces generiranja CL programa je generiranje BPEL procesa kojim se interni zapis naredbi izvornog SSCL programa prevodi u naredbe jezika BPEL. Generirani slijed naredbi jezika BPEL sačinjava *<process>* blok CL programa. Najsloženiji dio generiranja BPEL procesa je generiranje kôda za naredbu *invoke* SSCL programa. Tijekom prevođenja naredbe *invoke* generira se nekoliko blokova naredbi jezika BPEL. Osim toga, potrebno je parsirati WSDL opis pozivane mrežne usluge i dohvatiti podatke potrebne za ispravno generiranje BPEL naredbi.



Slika 6.7 Proces generiranja ciljnog programa

Primjer generiranja BPEL kôda za naredbu *invoke* prikazan je na slici 6.8. Ulaz u postupak generiranja je hijerarhijska struktura generirana tijekom semantičke analize i WSDL opis mrežne usluge *Calculator*. WSDL opis je izostavljen sa slike zbog preglednosti, a svi podaci iz WSDL dokumenta ugrađeni u BPEL kôd prikazani su podvučeno. Generirani BPEL kôd čine četiri bloka naredbi. U prvom se bloku, na osnovi imena mrežne usluge, inicijalizira *partnerLink* s fizičkom adresom mrežne usluge. Nakon toga se ulazni parametri zapisuju u SOAP poruku čija je struktura određena WSDL opisom mrežne usluge. Treći blok je naredba BPEL jezika *<invoke>* kojom se poziva mrežna usluga na adresi definiranu u prvom koraku. Konačno, u posljednjem bloku se iz odgovora, primljenog u obliku SOAP poruke, kopira rezultat operacije u varijablu izvornog programa koja je navedena kao izlazni parametar naredbe *invoke*.

Slika 6.8 Primjer generiranja BPEL kôda za naredbu *invoke*

### Generiranje opisa sučelja raspodijeljenog programa

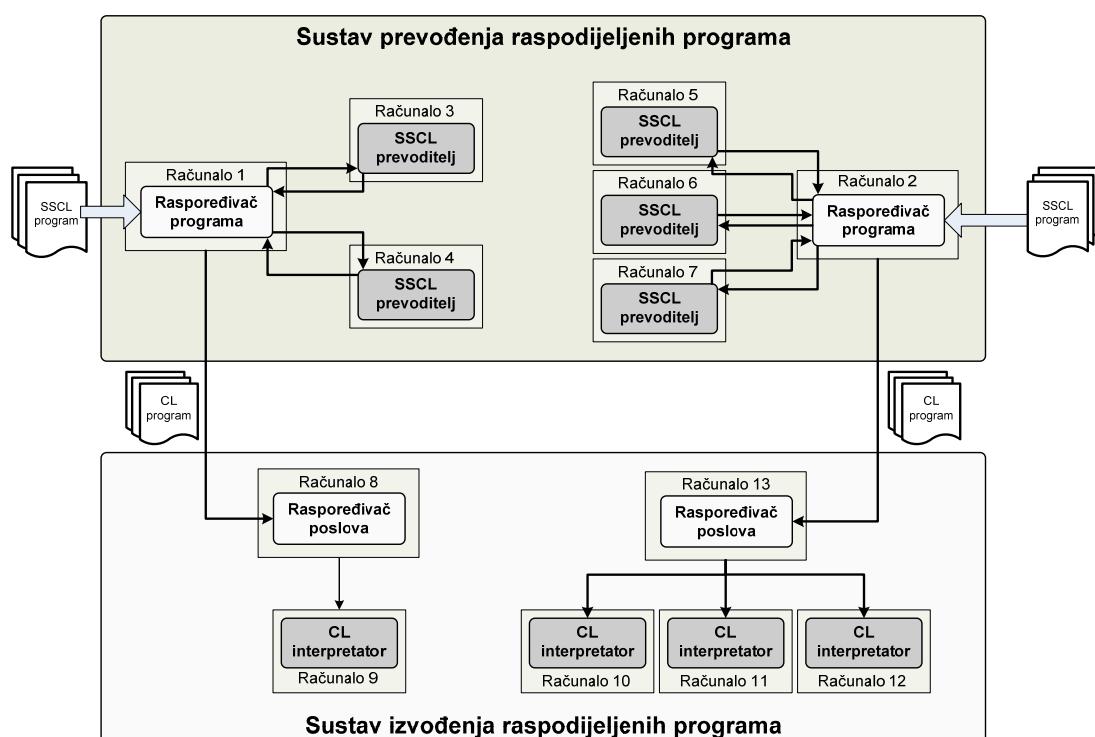
Opis sučelja raspodijeljenog programa generira se u obliku WSDL dokumenta. WSDL opis sučelja raspodijeljenog programa sadrži sve informacije potrebne drugim mrežnim uslugama koje žele komunicirati s raspodijeljenim programom. Ne postoji naredbe SSCL jezika kojima krajnji korisnik direktno definira operacije koje se dodaju u sučelje mrežne usluge. Zbog toga su jedine operacije definirane u sučelju raspodijeljenog programa operacije povratnog poziva (engl. *callback*). Operacije povratnog poziva su potrebne kako bi se omogućilo korištenje asinkronih operacija mrežnih usluga. Usluge suradnje i natjecanja, definirane programskim modelom zasnovanim na uslugama, su često korištene usluge s definiranim asinkronim operacijama. Na osnovi informacija prikupljenih analizom izvornog programa moguće je odrediti koje su se usluge suradnje i natjecanja pozivale, te dodati potrebne operacije povratnog poziva u WSDL opis sučelja raspodijeljenog programa. Generirani WSDL opis zapisuje se unutar *<processDefinition>* bloka ciljnog CL programa.

### Objedinjavanje opisa sučelja pozivanih mrežnih usluga

Objedinjavanjem opisa sučelja mrežnih usluga suradnika generira se `<partnerDefinitions>` blok ciljnog CL programa. Prilikom analize izvornog programa dohvaćeni su svi WSDL opisi mrežnih usluga koje su naslovljene naredbom `invoke` i spremišteni u unutarnji međuspremnik WSDL dokumenata. U ovom koraku generiranja ciljnog programa svi se WSDL opisi iz internog spremnika ugrađuju unutar bloka `<partnerDefinitions>`. Osim mrežnih usluga izravno naslovljenih naredbom `invoke`, u blok opisa mrežnih usluga dodaju se i svi opisi mrežnih usluga koje su neizravno naslovljene nekom od ugrađenih naredbi za uporabu usluga suradnje i natjecanja.

## 6.2 Sustav prevođenja i izvođenja raspodijeljenih programa

Sustav prevođenja sastoje se od dvije vrste komponenti: *Rasporedioca programa* i *SSCL prevoditelja*. *Rasporedioci programa* su moduli u koje se privremeno spremaju raspodijeljeni programi napisani SSCL jezikom. *Rasporedioci programa* ostvaruju mehanizme rasporediđivanja procesa prevođenja slanjem raspodijeljenih programa napisanih SSCL jezikom *SSCL Prevoditeljima*. *SSCL Prevoditelji* prevode SSCL programe u CL programe. Prevedene CL programe *Rasporedioci programa* šalje *Rasporedioci poslova* koji je ulazna točka sustava izvođenja raspodijeljenih programa.



Slika 6.9 Okolina prevođenja i okolina izvođenja raspodijeljenih programa

Sustav izvođenja raspodijeljenih programa sastoji se od *Rasporedišala poslova* i *CL Interpretatora*. *CL Interpretatori* povezani s određenim *Rasporedišačem poslova*, dohvaćaju raspodijeljene programe napisane CL jezikom i izvode ih [77].

Primjer prikazan na slici 6.9 prikazuje sustav prevođenja koji se sastoji od dva *Rasporedišača programa* i pet *SSCL Prevoditelja*. *Rasporedišač programa* na računalu 1 povezan je sa *SSCL prevoditeljima* na računalima 3 i 4. *Rasporedišač programa* šalje primljene SSCL programe prvom slobodnom *SSCL prevoditelju*, a kao odgovor prima CL program koji šalje na izvođenje *Rasporedišaču poslova* na računalu 8. *Rasporedišač programa* na računalu 2 šalje SSCL programe *SSCL prevoditeljima* na računalima 5, 6 i 7, a generirane CL programe šalje na izvođenje *Rasporedišaču poslova* na računalu 13.

Pojedini *Rasporedišači programa* povezani s određenim brojem *SSCL prevoditelja* definiraju samostalnu infrastrukturu za prevođenje raspodijeljenih programa. Infrastruktura prevođenja proširena s *Rasporedišačem poslova* definira infrastrukturu za izvođenje raspodijeljenih programa napisanih u jeziku SSCL.

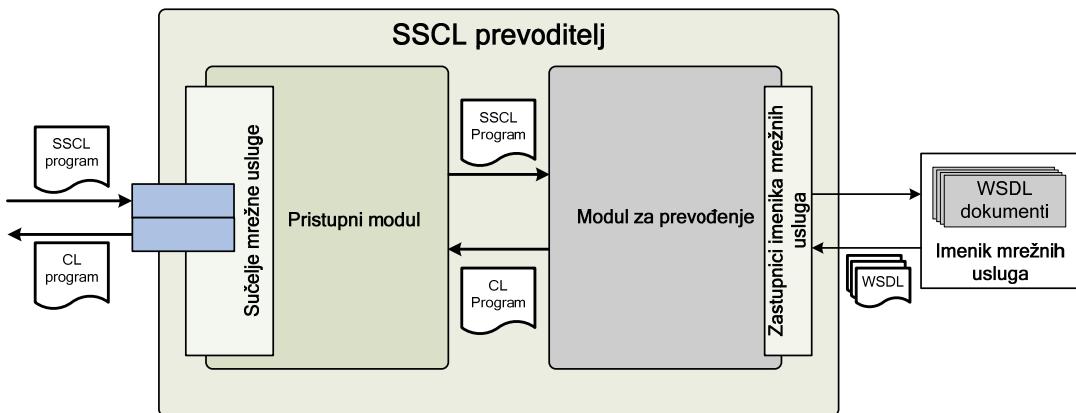
Na primjeru pikazanom na slici 6.9, *SSCL Prevoditelji* na računalima 3 i 4, odnosno na računalima 5, 6 i 7 su samostalne programske cjeline i paralelno prevode raspodijeljene programe. Broj prevoditelja vezanih uz jedan *Rasporedišač poslova* nije ograničen i moguće ga je dinamički prilagođavati potrebama sustava za prevođenje i dostupnim računalnim sredstvima. Povećanjem broja *SSCL Prevoditelja* moguće je oblikovati raspodijeljeni sustav prevođenja koji ima svojstva razmernog rasta s povećanjem broja korisnika, odnosno broja raspodijeljenih programa koji ulaze u sustav.

## 6.3 Arhitektura raspodijeljenog sustava prevođenja

Arhitektura raspodijeljenog sustava prevođenja oblikovana je prema zahtjevima raspodijeljene i otvorene računalne okoline u kojoj je predviđeno izvođenje raspodijeljenog sustava prevođenja. Dvije osnovne jedinice definirane arhitekturom sustava prevođenja su *SSCL Prevoditelji* i *Rasporedišač programa*.

### 6.3.1 Arhitektura SSCL prevoditelja

Arhitektura *SSCL prevoditelja* prikazana je na slici 6.10. Arhitekturu čine *Pristupni modul* i *Modul za prevođenje*. *Pristupni modul* ostvaruje sučelje mrežne usluge *SSCL prevoditelja* kojim se funkcionalnost *Modula za prevođenje* izlaže prema okolini. *Modul za prevođenje* ostvaruje proces prevođenja jezika SSCL u jezik CL. Tijekom procesa prevođenja modul koristi funkcionalnosti vanjskih imenika mrežnih usluga i pribavlja WSDL opise usluga potrebne za prevođenje.

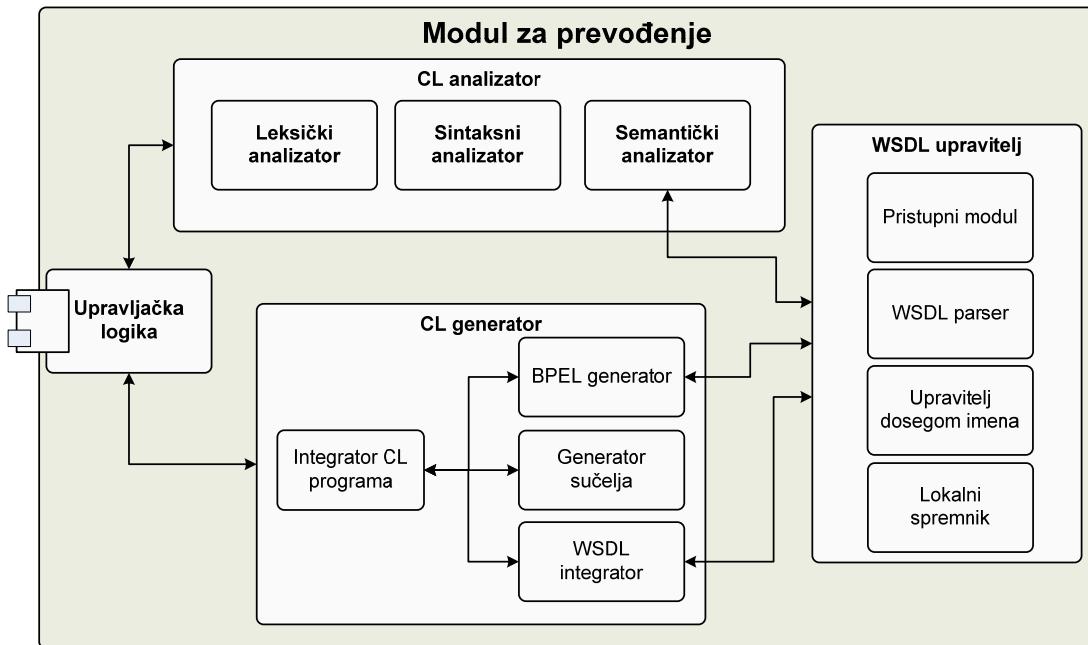


Slika 6.10 Arhitektura SSCL Prevoditelja

### Arhitektura modula za prevođenje

*Modul za prevođenje* ostvaruje proces prevođenja opisan u odjeljku 6.1.2. Arhitektura modula prikazana je na slici 6.11. Logička dekompozicija modula ostvarena je u skladu s funkcionalnim cjelinama procesa prevođenja i sastoји se od *SSCL analizatora* i *CL generatora*. *SSCL analizator* obavlja proces analize izvornog SSCL programa i sastoји se od modula *Leksički*, *Sintakski* i *Semantički analizator*, kojima je ostvarena funkcionalnost pojedine faze analize. Proces generiranja CL programa ostvaren je posebnim modulom nazvanim *CL generator*. *CL generator* je logički podijeljen na module *BPEL generator*, *Generator sučelja*, *WSDL integrator* i *Integrator CL programa*. Prva tri modula ostvaruju tri funkcionalne cjeline procesa generiranja ciljnog programa. Integrator CL programa upravlja radom tri navedena modula i rezultate pojedinih modula povezuje u cjeloviti CL program.

*WSDL upravitelj* je modul koji dohvaća WSDL opise mrežnih usluga iz imenika usluga i sprema dohvaćene opise u lokalne spremnike. Osnovna uloga *WSDL upravitelja* je da osigurava učinkoviti pristup dijelovima WSDL dokumenata potrebnim u pojedinim fazama rada ostalih modula. *WSDL upravitelj* sastoји se od četiri podmodula, *Pristupni modul*, *WSDL parser*, *Upravitelj područjem imena* i *Lokalni spremnik*. *Pristupni modul* ostvaruje funkcionalnosti pristupa različitim imenicima mrežnih usluga iz kojih dohvaća WSDL opise mrežnih usluga. Dohvaćeni WSDL dokumenti spremaju se u *Lokalni spremnik* u kojem se čuvaju tijekom procesa prevođenja SSCL programa. *WSDL parser* omogućuje učinkovito pretraživanje i dohvat pojedinih dijelova WSDL dokumenata. *Upravitelj prostorom imena* zadužen je za preslikavanje prefiksa prostora imena (engl. *namespace prefix*) [17] XML elemenata definiranih u izvornim WSDL dokumentima u prefikse definirane u BPEL opisu procesa. Naime, moguće je da dva različita WSDL dokumenta definiraju isti prefiks za dva različita prostora imena. Prilikom generiranja CL programa tim prostorima imena moraju se dodijeliti različiti prefiksi jer u protivnom nije moguće odrediti koji XML elementi CL programa pripadaju kojem području imena.



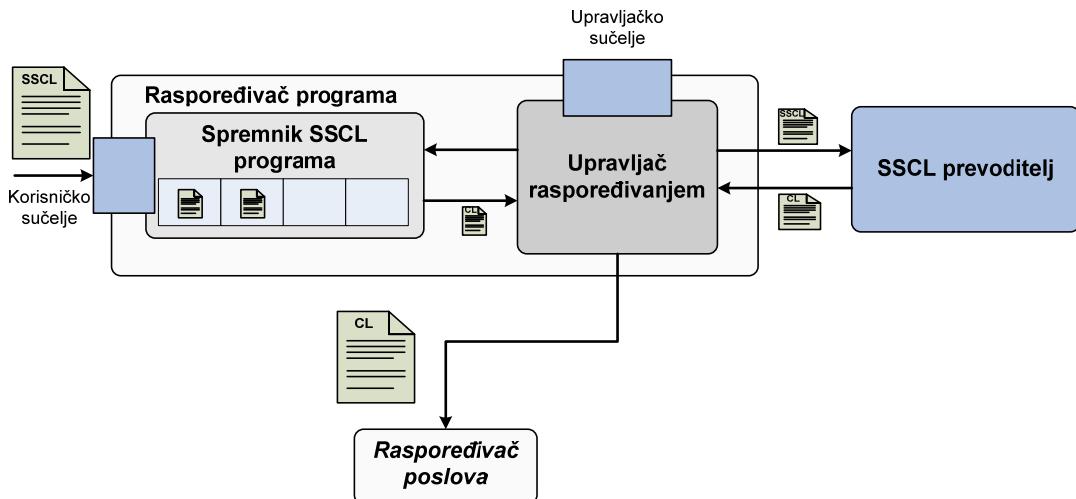
Slika 6.11 Arhitektura modula za prevodenje

Programska logika koja upravlja radom cijelog modula za prevodenje ostvarena je modulom *Upravljačka logika*. Ovim modulom se funkcionalnosti pojedinih faza rada jezičnog procesa povezuju u jedinstvenu funkcionalnost koja izvorni raspodijeljeni program napisan jezikom SSCL prevodi u CL program. Funkcionalnost cijelog modula za prevodenje izložena je prema drugim dijelovima sustava programskim sučeljem koje je ostvareno kao dio modula *Upravljačka logika*.

### 6.3.2 Arhitektura raspoređivača programa

Funkcionalnosti ostvarene modulom *Raspoređivač programa* su prihvatanje raspodijeljenih programa od korisnika, privremeno spremanje raspodijeljenih programa, slanje raspodijeljenih programa na prevodenje i slanje prevedenih raspodijeljenih programa sustavu za izvođenje.

Raspoređivač programa sastoji se od *Spremnika SSCL programa* i *Upravljača raspoređivanjem*. *Spremnik SSCL programa* ostvaren je kao FIFO (*First In First Out*) spremnik programa koji služi za privremeno spremanje programa poslanih na prevodenje. *Upravljač raspoređivanjem* je modul koji definira logiku procesa raspoređivanja. Proces raspoređivanja je iterativni postupak kojim se dohvata program iz *Spremnika SSCL programa*, šalje na prevodenje raspoloživom *SSCL prevoditelju*, te se prevedeni CL program šalje sustavu za izvođenje, odnosno *Raspoređivaču poslova*.

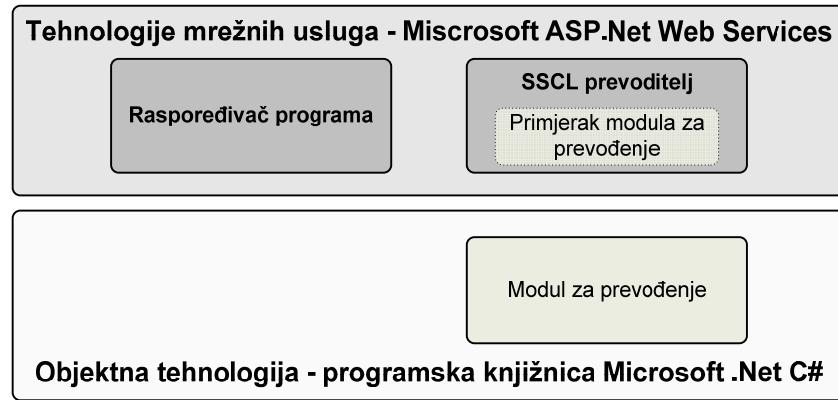
Slika 6.12 Arhitektura *Raspoređivača programa*

Programsko sučelje raspoređivača programa podijeljeno je na korisničko sučelje i upravljačko sučelje. Korisničko sučelje sadrži operacije za pristup *Spremniku SSCL programa* i omogućuje korisniku slanje SSCL programa na prevođenje, odnosno spremanje raspodijeljenih programa u *Spremnik SSCL programa*. Upravljačkim sučeljem definirane su operacije za pristup *Upravljaču raspoređivanja* koje korisniku omogućuju pokretanje i zaustavljanje procesa raspoređivanja programa.

## 6.4 Programsко ostvarenje raspodijeljenog sustava prevođenja

Raspodijeljena okolina prevođenja ostvarena je uporabom objektno-orientirane tehnologije i tehnologije mrežnih usluga. Tehnologije korištene u programskom ostvarenju sustava za prevođenje prikazane su na slici 6.13. Komponente raspodijeljene okoline za prevođenje, *Raspoređivač programa* i *SSCL prevoditelj*, ostvareni su tehnologijama mrežnih usluga. Modul za prevođenje kojim je ostvarena funkcionalnost prevoditelja jezika SSCL u jezik CL ostvaren je objektno-orientiranom tehnologijom.

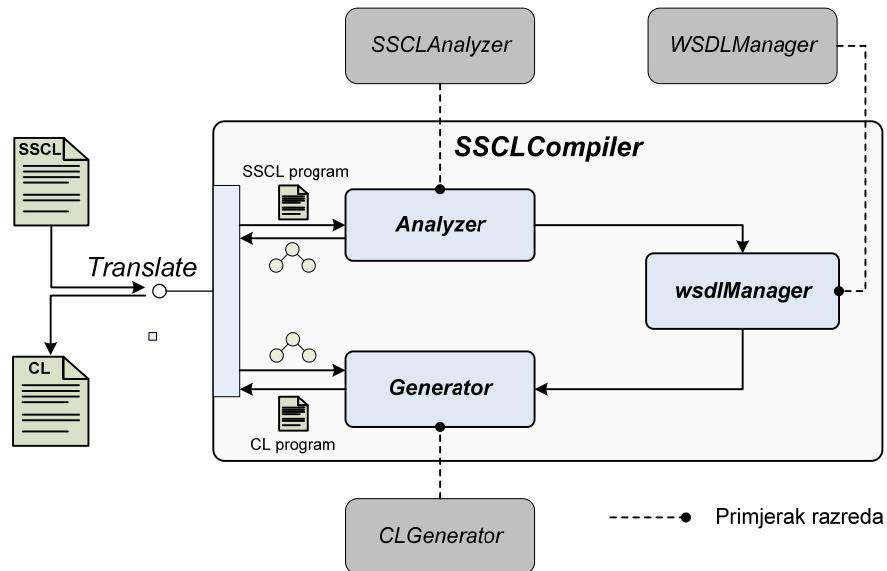
Cjelokupno programsko ostvarenje zasnovano je na programskom okrilju Microsoft .Net. Kao tehnologija ostvarenja mrežnih usluga korištena je *ASP.Net Web Services* tehnologija, a kao objektna tehnologija korišten je programski jezik C#.



Slika 6.13 Tehnologije programskog ostvarenja raspodijeljenog sustava prevodenja

#### 6.4.1 Programsко ostvarenje modula za prevodenje

Modul za prevodenje ostvaren je razredom *SSCLCompiler*. Razred *SSCLCompiler* prikazan je na slici 6.14 i sastoji se od primjerka razreda *SSCLAnalyzer*, primjerka razreda *CLGenerator* i primjerka razreda *WSDLManager*.



Slika 6.14 Programsko ostvarenje modula za prevodenje

Pristupno sučelje razreda *SSCLCompiler* sadrži samo metodu *Translate(string SSCLProgram)* koja kao parametar prima tekstualni zapis SSCL programa, a povratna vrijednost je tekstualni zapis CL programa. Nakon poziva metode *Translate* poziva se metoda *Analyze(string SSCLProgram)* primjerka razreda *SSCLAnalyzer* koja obavlja analizu izvornog SSCL programa. Nakon uspješno izvedene analize izvornog SSCL programa, metoda *Analyze* vraća hijerarhijski zapis izvornog programa. Hijerarhijski zapis je parametar metode *Generate* primjerka razreda *CLGenerator*. Tijekom procesa analize izvornog programa stvara

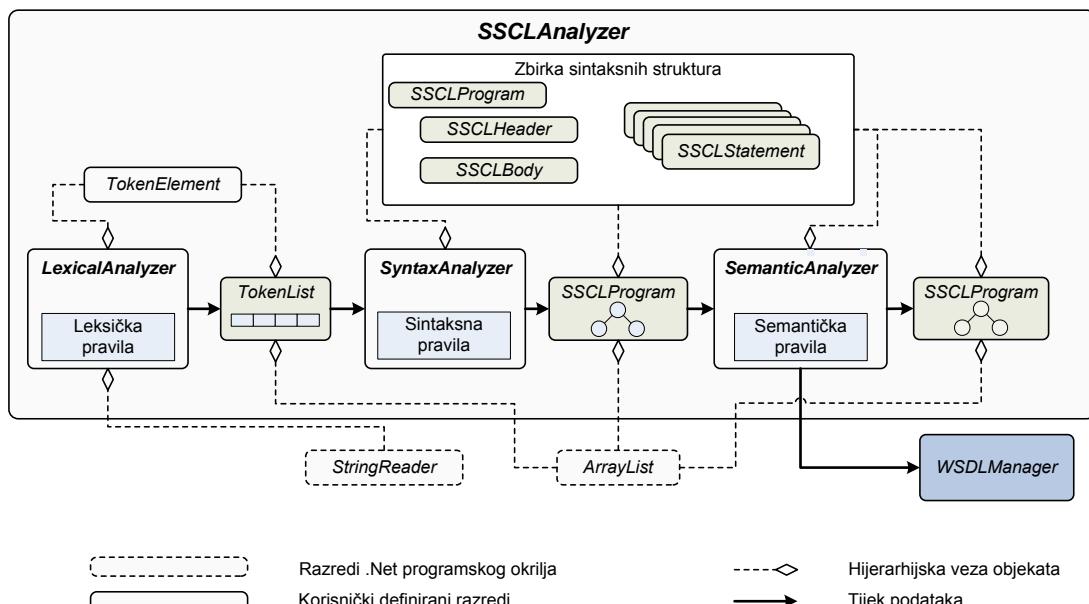
se i primjerak razreda *WSDLManager* koji se koristi za upravljanje WSDL dokumentima s opisima mrežnih usluga suradnika SSCL programa.

Programsko ostvarenje razreda *SSCLCompiler* i svih njegovih podrazreda ostvareno je C# programskim jezikom i prevedeno je kao programska knjižnica koja je spremljena u datoteci *SSCLCompilerLibrary.dll*. Navedena se knjižnica koristi za izgradnju mrežne usluge *SSCLTranslator*.

### *Programsko ostvarenje modula SSCL analizator*

Programsko ostvarenje modula SSCL Analizator prikazano je na slici 6.15. Cjelokupna funkcionalnost modula ostvarena je razredom *SSCLAnalyzer* koji povezuje funkcionalnosti razreda *LexicalAnalyzer*, razreda *SyntaxAnalyzer* i razreda *SemanticAnalyzer* kojima su ostvarene pojedine faze procesa analize izvornog programa.

Leksički analizator ostvaren je kao zaseban razred *LexicalAnalyzer* koji ima zasebno zapisana leksička pravila SSCL jezika i ostvaruje mehanizam koji čita ulazni niz i prema zapisanim pravilima grupira leksičke jedinke. Slijedno čitanje znakova ulaznog programa ostvareno je proširenjem razreda *StringReader* koji je dio .Net radnog okvira (engl. *framework*). Prepoznate leksičke jedinke i pripadni razred leksičke jedinke spremi se u strukturu *TokenElement*. Izlaz cjelokupnog procesa leksičke analize je podatkovna struktura *TokenList* koja sadrži slijed primjeraka razreda *TokenElement* spremljenih u primjerak razreda *ArrayList*.



**Slika 6.15 Programsko ostvarenje modula SSCL analizator**

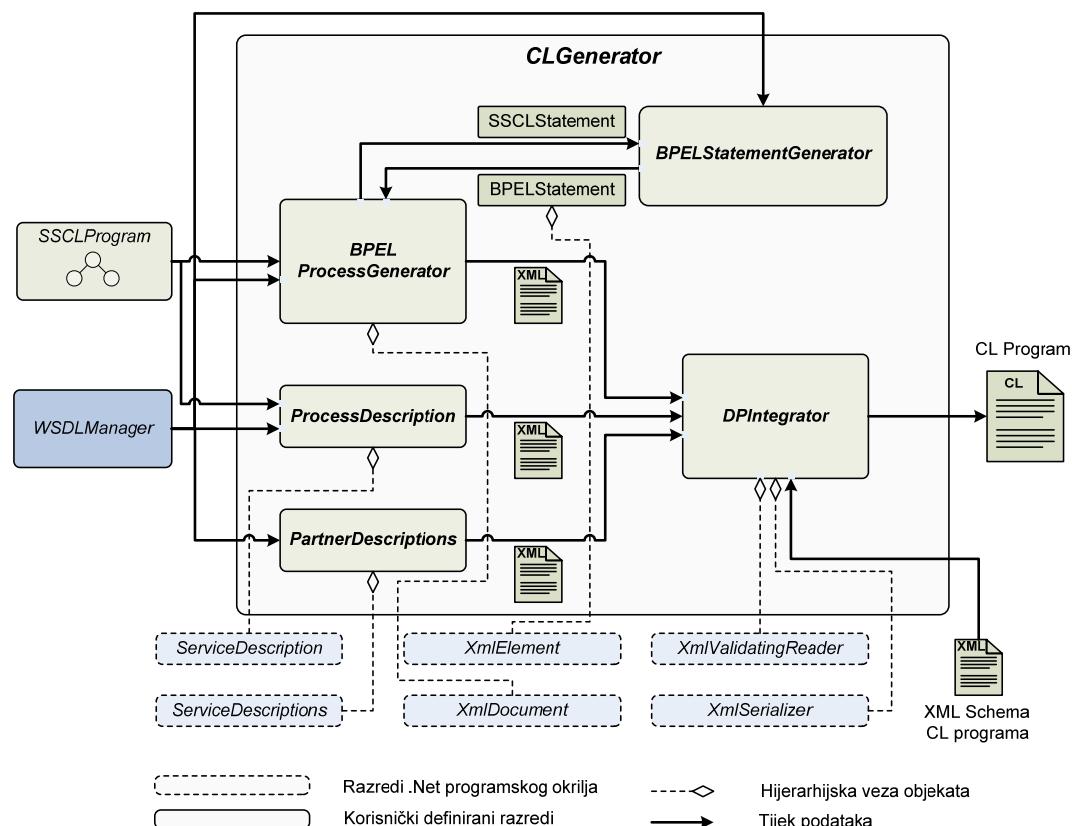
Sintaksni analizator ostvaren je razredom *SyntaxAnalyzer* koji tehnikom rekurzivnog spusta [5] gradi sintaksno stablo ulaznog programa. Nakon prepoznavanja pojedine sintaksne

cjeline SSCL programa, stvara se primjerak razreda koji opisuje prepoznatu sintaksnu cjelinu. Cjelokupno sintaksno stablo SSCL programa spremi se u primjerak razreda *SSCLProgram* koji je izlazna struktura podataka procesa sintaksne analize.

Proces semantičke analize ostvaren je razredom *SemanticAnalyzer* koji obilaskom sintaksnog stabla provjerava semantička pravila i gradi generativno stablo. Generativno stablo je u memoriji spremljeno korištenjem istih podatkovnih struktura kao i sintaksno stablo samo što su podatkovne strukture proširene dodatnim informacijama prikupljenim tijekom semantičke analize. Osim toga, semantički analizator stvara objekt razreda *WSDLManager* i u njega spremi WSDL opise sučelja svih mrežnih usluga suradnika izvornog programa. Izlaz iz cjelokupnog procesa analize je generativno stablo spremljeno u primjerak razreda *SSCLProgram* i stvoreni primjerak razreda *WSDLManager*.

### Programsko ostvarenje modula CL generator

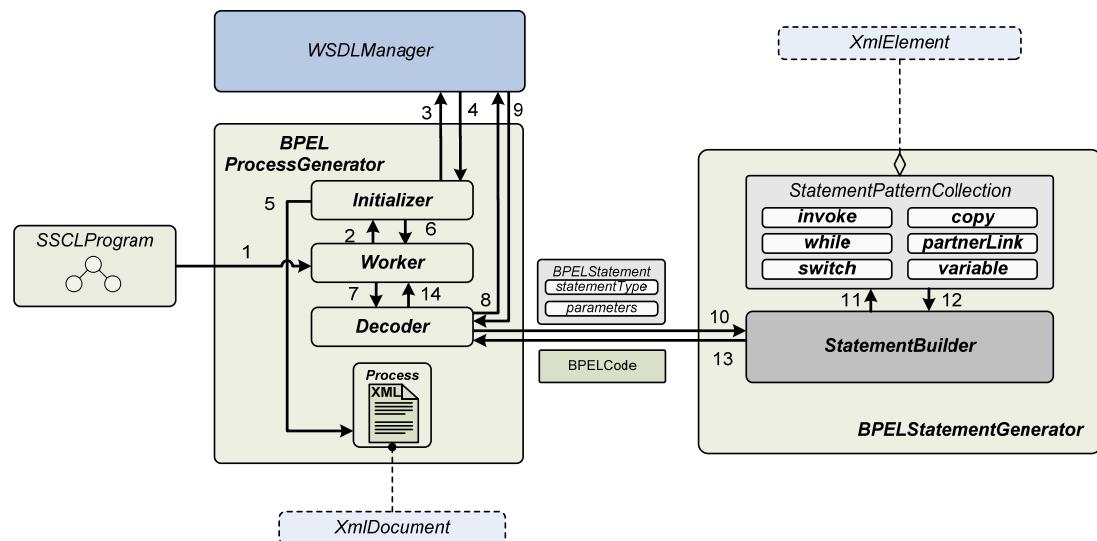
Programsko ostvarenje modula CL generator prikazano je na slici 6.16. Cjelokupna funkcionalnost modula ostvarena je razredom *CLGenerator* koji sadrži primjerke razreda *BPELProcessGenerator*, *BPELStatementGenerator*, *ProcessDescription* i *PartnerDescriptions*.



Slika 6.16 Programsко ostvarenje *CL Generatora*

Razred *BPELProcessGenerator* na osnovi generativnog stabla SSCL programa generira XML dokument s opisom BPEL procesa. Generiranje opisa BPEL procesa odvija se međudjelovanjem primjeraka razreda *BPELProcessGenerator* i *BPELStatementGenerator*. Proces generiranja BPEL procesa i ostvarenje razreda *BPELProcessGenerator* i *BPELStatementGenerator* prikazani su na slici 6.17. Razred *BPELProcessGenerator* ostvaruje funkcionalnost stvaranja XML dokumenta za opis BPEL procesa, obilaska generativnog stabla SSCL programa i prevođenja SSCL naredbi u istovjetni skup naredbi BPEL jezika. Funkcionalnost generiranja pojedinačnih naredbi BPEL jezika ostvarena je razredom *BPELStatementGenerator*.

Razred *BPELProcessGenerator* ostvaren je primjercima razreda *Initializer*, *Worker* i *Decoder*. *Worker* razred ostvaračke logiku procesa generiranja BPEL procesa. Nakon što *BPELProcessGenerator* primi generativno stablo SSCL programa proslijedi ga primjerku razreda *Worker* (1).



Slika 6.17 Ostvarenje generiranja BPEL4WS opisa procesa

Razred *Worker* prvo pokreće inicijalizacijski proces ostvaren razredom *Initializer* (2). Funkcionalnosti razreda *Initializer* su stvaranje objekta za spremanje XML zapisa BPEL4WS procesa i inicijalizacija BPEL4WS procesa. Inicijalizacija BPEL4WS procesa obuhvaća stvaranje strukture XML dokumenta u skladu s pravilima BPEL4WS programskog jezika. Prvo se stvara korijenski XML element `<process>`. Na osnovi podataka dohvaćenih iz primjerka razreda *WSDLManager* (3, 4) definiraju se svi potrebni atributi elementa `<process>`. Tijekom definiranja atributa elementa `<process>` potrebno je definirati i odgovarajuće prefiks područja imena (engl. *namespace prefix*) za cijeli XML dokument. Imena svih potrebnih područja imena i pripadajući prefiksi dohvaćaju se iz primjerka razreda *WSDLManager*. Nakon definiranja korijenskog elementa i njegovih atributa generira se blok deklaracije

varijabli BPEL4WS programa. Generirani blok deklaracije sadržan je unutar elementa `<variables>` koji se dodaje elementu `<process>`. Nakon toga se generira blok deklaracije lokalnih imena pozivanih mrežnih usluga. Lokalna imena spremaju se unutar bloka `<partnerLinks>`. Proces inicijalizacije BPEL4WS programa završava definiranjem elementa `<sequence>` unutar kojeg će se generirati naredbe BPEL4WS programa. Nakon završetka procesa inicijalizacije, kazaljka na stvoreni XML dokument proslijedi se primjerku razreda *Decoder* (5), a primjerku razreda *Worker* dojavljuje se da je proces inicijalizacije završio (6).

Nakon završetka inicijalizacije, primjerak razreda *Worker* počinje proces obilaska generativnog stabla SSCL programa. Tijekom obilaska stabla primjerak razreda *Worker* čita sve naredbe SSCL programa i svaku pročitanu naredbu proslijedi primjerku razreda *Decoder* (7). *Decoder* obavlja pretvorbu naredbe SSCL programa u niz primjeraka razreda *BPELStatement* koji definiraju naredbe BPEL4WS programske jezike. Kako je SSCL jezik više razine, jedna naredba SSCL programa prevodi se u više naredbi jezika BPEL4WS. Primjer prikazan na slici 6.8 prikazuje kako se jedna *invoke* naredba SSCL programske jezike prevodi u četiri bloka naredbi BPEL4WS jezika. Razred *BPELStatement* sadrži pokazatelj BPEL4WS naredbe i niz parametara naredbe. Pokazatelj BPEL4WS naredbe jedinstveno određuje naredbu jezika BPEL4WS, a niz parametara sadrži pokazatelje BPEL4WS varijabli i konstante. Jedine konstante koje se mogu pojaviti kao parametar BPEL4WS naredbe su tekstualne konstante koje sadrže opis dijela sučelja raspodijeljenih programa. Navedene konstante sadrže WSDL opis operacija povratnog poziva koje je potrebno navesti prilikom poziva asinkronih operacija primjenskih mrežnih usluga. U tom slučaju, opis sučelja BPEL procesa je nužan kako bi pozvana mrežna usluga mogla poslati odgovor na odgovarajuće sučelje raspodijeljenog programa. Sučelje raspodijeljenog programa dohvata se iz *WSDLManager* objekta (8, 9).

Podatkovna struktura *BPELStatement* predaje se primjerku razreda *BPELStatementGenerator* (10), koji generira odgovarajuću naredbu jezika BPEL4WS. Razred *BPELStatementGenerator* sastoji se od skupa uzorka naredbi jezika BPEL4WS. Uzorak BPEL4WS naredbe ostvaruje se kao proširenje razreda *XMLElement* i sadrži XML zapis strukture BPEL4WS naredbe. Razred *StatementBuilder* na osnovi pokazatelja naredbe dohvata odgovarajući uzorak naredbe jezika BPEL4WS (11, 12) i popunjava uzorak sa skupom parametara. Generirana naredba se u obliku primjerka razreda *XMLElement* vraća razredu *Decoder* (13), koji je zapisuje na odgovarajuće mjesto unutar BPEL4WS programa. Koraci 8 do 13 ponavljaju se sve dok se ne generira BPEL4WS kôd za sve primjerke razreda *BPELStatement* nastale dekodiranjem jedne naredbe jezika SSCL. Nakon toga, *Decoder* javlja primjerku razreda *Worker* da je završio s prevodenjem naredbe jezika SSCL (14) te primjerak razreda *Worker* nastavlja s obilaskom generativnog stabla. Koraci 7-14 ponavljaju se sve dok nije obiđeno cijelokupno stablo.

Razred *ProcessDescription*, prikazan na slici 6.16, koristi se za generiranje WSDL opisa sučelja raspodijeljenog programa. Sučelje raspodijeljenog programa mora sadržavati operacije povratnog poziva (engl. *callback*) nužne za asinkronu komunikaciju s primjenskim mrežnim uslugama. Razred *ProcessDescription* generira opise navedenih operacija na osnovi informacija o primjenskim uslugama navedenim unutar primjerka razreda *WSDLManager*. Osnovni razred .Net radnog okvira koji se koristi za definiranje opisa sučelja raspodijeljenog programa je razred *ServiceDescription*.

Razred *PartnerDescriptions* koristi se za generiranje dijela raspodijeljenog programa koji sadrži WSDL opise svih mrežnih usluga suradnika raspodijeljenog programa. Svi potrebni WSDL dokumenti dohvaćaju se iz primjerka razreda *WSDLManager* i spremaju se unutar zbirke *ServiceDescriptions*.

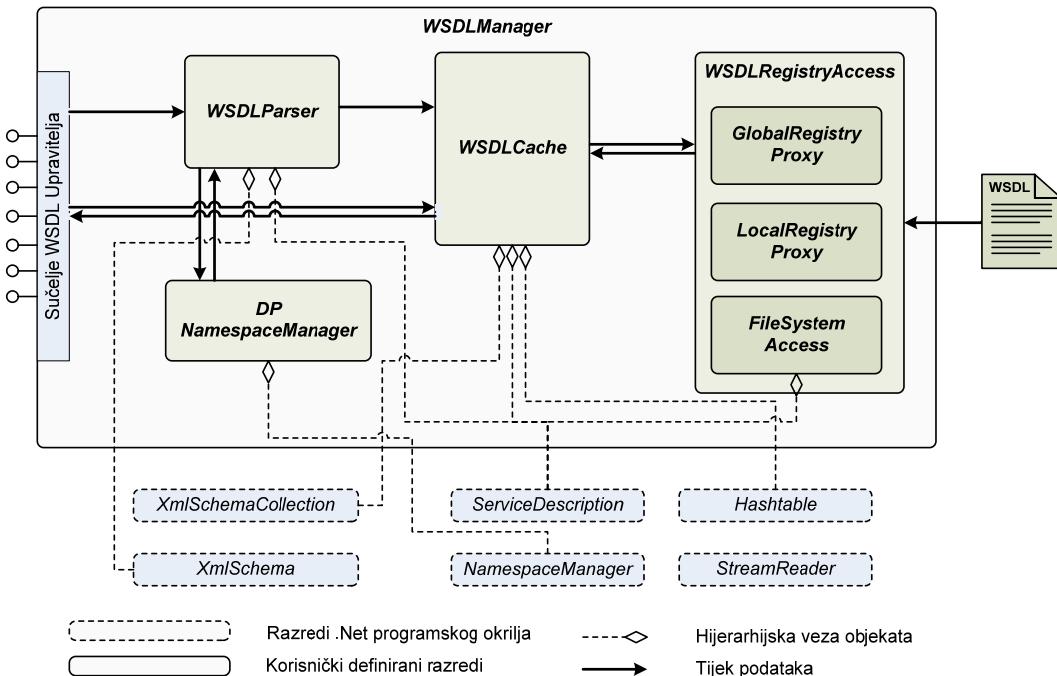
Razred *DPIIntegrator* koristi izlaze primjeraka razreda *BPELProcessGenerator*, *ProcessDescription* i *PartnerDescriptpitons* te ih povezuje prema pravilima CL jezika. *DPIIntegrator* stvara primjerak razreda *XMLDocument* te definira korijenski element *<dpProgram>*. Unutar korijenskog elementa redom se dodaju element *<process>* koji sadrži opis BPEL procesa, element *<processDescription>* koji sadrži opis sučelja raspodijeljenog programa i element *<partnerDescriptions>* unutar kojeg je sadržan niz WSDL opisa mrežnih usluga koje je generirao primjerak razreda *partnerDescriptions*. Ispravnost generiranog dokumenta provjerava se primjenom zadane XML Scheme raspodijeljenog programa uporabom *XMLValidatingReader* objekta. Ako je generirani raspodijeljeni program ispravan, on se serijalizira u tekstualni dokument koji sadrži XML zapis raspodijeljenog programa.

### *Programsko ostvarenje modula WSDL upravitelja*

Modul WSDL upravitelj programski je ostvaren razredom *WSDLManager* čija je struktura prikazana na slici 6.18. Razred *WSDLManager* sadrži po jedan primjerak razreda *WSDLParse*, razreda *WSDLCache*, razreda *WSDLRegistryAccess* i razreda *DPNamespaceManager*.

Programsko sučelje razreda *WSDLManager* definira metode kojima ostali razredi modula za prevodenje pristupaju potrebnim funkcionalnostima za upravljanje WSDL dokumentima. Definirane metode dijele se na metode za upravljanje WSDL dokumentima i metode za pristup dijelovima WSDL dokumenta. Za upravljanje WSDL dokumentima definirane su metode *AddWSDL* i *GetWSDL*. Pozivi ovih metoda izravno se sa sučelja *WSDLManager* objekta preusmjeravaju na istoimene metode sučelja *WSDLCache* objekta kojim je ostvaren lokalni spremnik WSDL dokumenata. *AddWSDL* metoda koristi se za dohvat WSDL dokumenta iz imenika WSDL usluga i povezivanje dohvaćenog WSDL dokumenta s lokalnim imenom usluge definiranim u SSCL programu. WSDL dokument se dohvaca iz imenika na

osnovi globalnog pokazatelja mrežne usluge. Globalni pokazatelj je najčešće URL usluge, ali programsko ostvarenje dopušta uporabu i korisnički definiranih pokazatelja. WSDL opis mrežne usluge spremaju se u primjerak razreda *ServiceDescription*. *ServiceDescription* objekt omogućuje spremanje deserijaliziranog WSDL dokumenta u memoriju računala. Primjerak razreda *ServiceDescription* spremaju se u lokalni spremnik. Lokalni spremnik je ostvaren primjenom primjerka razreda *Hashtable*. Razred *Hashtable* je dio .Net programskog okrilja kojim je ostvareno raspršeno adresiranje. Kao ključ raspršenog adresiranja koristi se lokalno ime mrežne usluge definirano u SSCL programu.

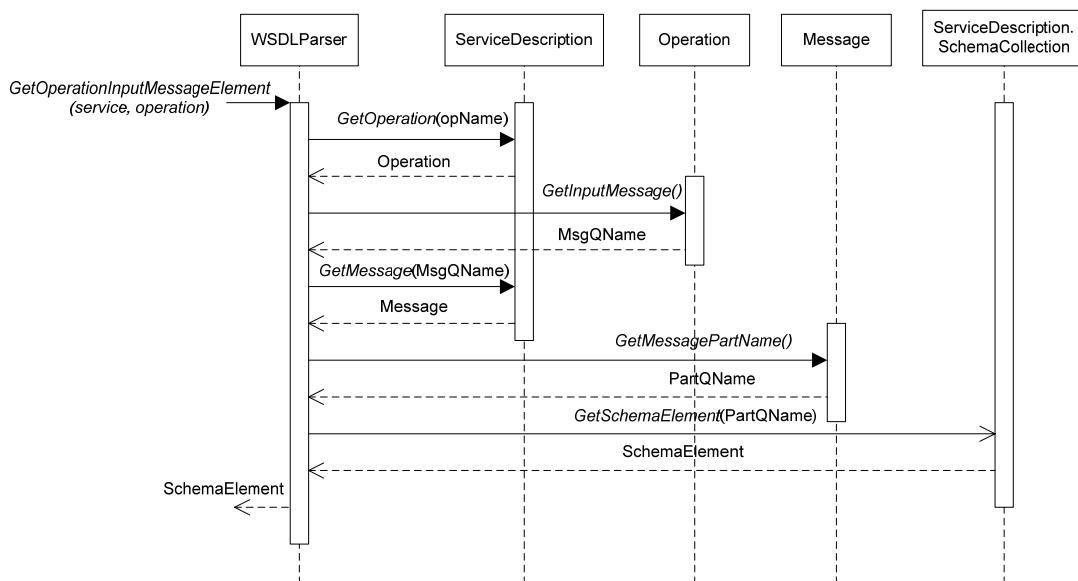


Slika 6.18 Programsko ostvarenje modula WSDL upravitelj

Metoda *GetWSDL* koristi se za dohvaćanje WSDL dokumenta koji je spremljen u lokalnom spremniku WSDL dokumenata. *GetWSDL* metoda dohvaća WSDL dokument mrežne usluge na osnovi lokalnog imena mrežne usluge definiranog u SSCL programu.

Razred *WSDLCache* pristupa imeniku mrežnih usluga koristeći modul za pristup imenicima. Modul za pristup imenicima ostvaren je razredom *WSDLRegistryAccess*. Razred *WSDLRegistryAccess* omogućuje pristup različitim ostvarenjima imenika mrežnih usluga. Trenutno ostvarenje omogućuje pristup WSDL dokumentima spremljenim u imenicima mrežnih usluga *GlobalRegistry* i *LocalRegistry*. *LocalRegistry* i *GlobalRegistry* imenici ostvareni su kao mrežne usluge, a opis ostvarenja nalazi se u sklpu rada [75]. Osim sučelja prema navedenim mrežnim uslugama imenika, ostvaren je i modul za pristup WSDL dokumentima spremljenim unutar lokalnog datotečnog sustava računala na kojem se izvodi *SSCL Prevoditelj*.

Sve metode razreda *WSDLManager* kojima se pristupa specifičnim informacijama u WSDL dokumentima izravno se proslijedu na sučelje primjerka razreda *WSDLPParser*. Razred *WSDLPParser* pristupa svim informacijama unutar WSDL dokumenta potrebnim za analizu izvornog SSCL programa i generiranje ciljnog CL programa. Osnovni objekt koji se koristi za ostvarenje razreda *WSDLPParser* je primjerak razreda *ServiceDescription* kojim se tekstualni WSDL dokument spremu u memorijsku strukturu koja omogućuje izravan pristup dijelovima WSDL dokumenta. Razred *WSDLPParser* nadograđuje razred *ServiceDescription* metodama prilagođenim za potrebe procesa prevođenja.



**Slika 6.19 Primjer izvodenja metode *GetOperationInputMessageElement* razreda *WSDLPParser***

Primjer specifične metode ostvarene za proces prevođenja razreda *WSDLPParser* je metoda *GetOperationInputMessageElement*. Metoda *GetOperationInputMessageElement* dohvaća XMLSchema opis ulazne SOAP poruke operacije mrežne usluge. Ulazni parametri metode su lokalno ime mrežne usluge definirano u SSCL programu i naziv operacije. Nakon poziva metode, *WSDLPParser* šalje zahtjev lokalnom spremniku WSDL dokumenata za opisom naslovljene mrežne usluge. Lokalni spremnik kao odgovor šalje primjerak razreda *ServiceDescription* koji sadrži WSDL opis tražene usluge. Nakon toga, izvodi se niz akcija kojima se u dohvaćenom WSDL dokumentu traži ulazna SOAP poruka operacije. Slijed metoda koje je potrebno izvesti prikazan je na slici 6.19. Prvo se dohvaća primjerak razreda *Operation* kojim je opisana operacija mrežne usluge. Nakon toga se iz primjerka razreda *Operation* dohvaća ime ulazne poruke operacije. Na osnovi imena ulazne poruke dohvaća se primjerak razreda *Message* koji sadrži pokazatelje XML Schema opisa struktura podataka poruke. Iz primjerka razreda *Message* dohvaća se pokazatelj XMLSchema opisa poruke. Na

osnovi dohvaćenog imena, dohvaća se XMLSchema element iz primjerka razreda *XMLSchemaCollection* pridruženog WSDL opisu.

WSDL parser koristi funkcionalnosti objekta *DPNamespaceManager* kojim je ostvareno preslikavanje prefiksa prostora imena (engl. *namespace prefix*) definiranih u izvornim WSDL dokumentima u prefikse lokalno definirane u CL programu. Ostale metode sučelja razreda *WSDLManager* prikazane su u tablici 6.2.

**Tablica 6-2 Operacije definirane sučeljem *WSDLManager* objekta**

Naziv metode	Opis metode
<i>AddWSDL</i>	Dodavanje WSDL opisa u lokalni spremnik.
<i>GetWSDL</i>	Dohvat WSDL opisa iz lokalnog spremnika.
<i>GetOperationRequestElement</i>	Vraća XMLSchema element ulazne SOAP poruke operacije mrežne usluge.
<i>GetOperationResponseElement</i>	Vraća XMLSchema element izlazne SOAP poruke operacije mrežne usluge.
<i>GetOperationRequestMessageElement</i>	Vraća XML element definiran prvom razinom XML Schema dokumenta koji opisuje ulaznu SOAP poruku.
<i>GetOperationResponseMessageElement</i>	Vraća XML element definiran prvom razinom XML Schema dokumenta koji opisuje izlaznu SOAP poruku.
<i>GetOperationParameterCount</i>	Vraća broj parametara operacije.
<i>DoesOperationReturnValue</i>	Provjerava da li operacija ima izlazni parametar.
<i>DoesOperationReturnBoolean</i>	Provjerava da li je izlazni parametar operacije tipa <i>Boolean</i> .
<i>DoesServiceContainsOperation</i>	Provjerava da li mrežna usluga sadrži operaciju sa zadanim imenom.
<i>GetServiceBindingAddress</i>	Vraća adresu ostvarenja mrežne usluge.

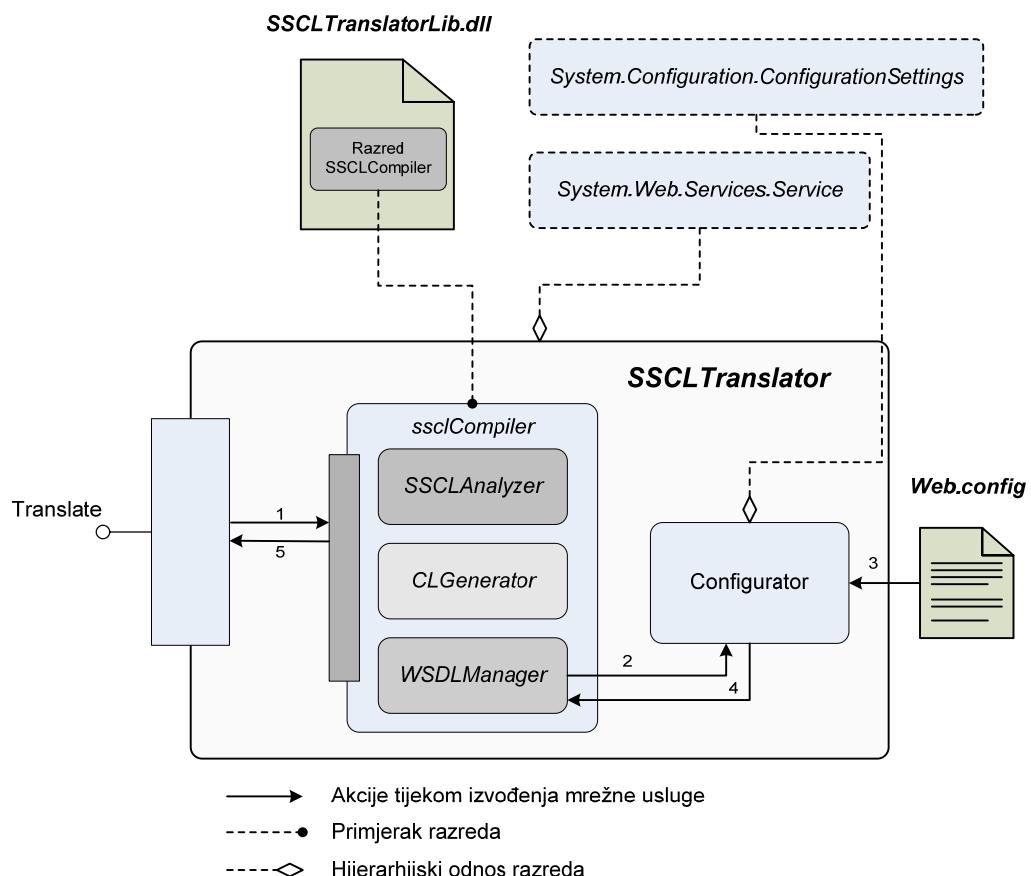
#### 6.4.2 Programsko ostvarenje modula SSCL prevoditelj i modula Raspoređivač programa

Moduli *SSCL prevoditelj* i *Raspoređivač programa* ostvareni su ASP.Net tehnologijom mrežnih usluga. *SSCL prevoditelj* ostvaren je mrežnom uslugom *SSCLTranslator*, a ostvarenje modula *Raspoređivač programa* podijeljeno je na mrežnu uslugu kojom je ostvaren spremnik raspodijeljenih programa poslanih na prevođenje i mrežnu uslugu koja ostvaruje proces raspoređivanja.

##### *Programsko ostvarenje modula SSCL prevoditelj*

Modul *SSCL prevoditelj* ostvaren je mrežnom uslugom *SSCLTranslator*. Ostvarenje mrežne usluge prikazano je na slici 6.20. Središnji dio programskega ostvarenja je primjerak razreda *SSCLCompiler* kojim je ostvaren proces prevođenja SSCL programa u CL program. Osim

modula za prevođenje, mrežna usluga sadrži i konfiguracijski modul kojim se postavljaju komunikacijski parametri potrebni za pristup imenicima mrežnih usluga. Proces konfiguracije ostvaren je razredom *Configurator* koji koristi funkcionalnosti razreda *ConfigurationSetting*. Razred *ConfigurationSetting* dio je .Net radnog okvira i omogućuje definiranje parametara primjenskih programa tijekom izvođenja programa. Komunikacijski parametri potrebni za pristup imenicima usluga zapisani su u datoteci *Web.config*.



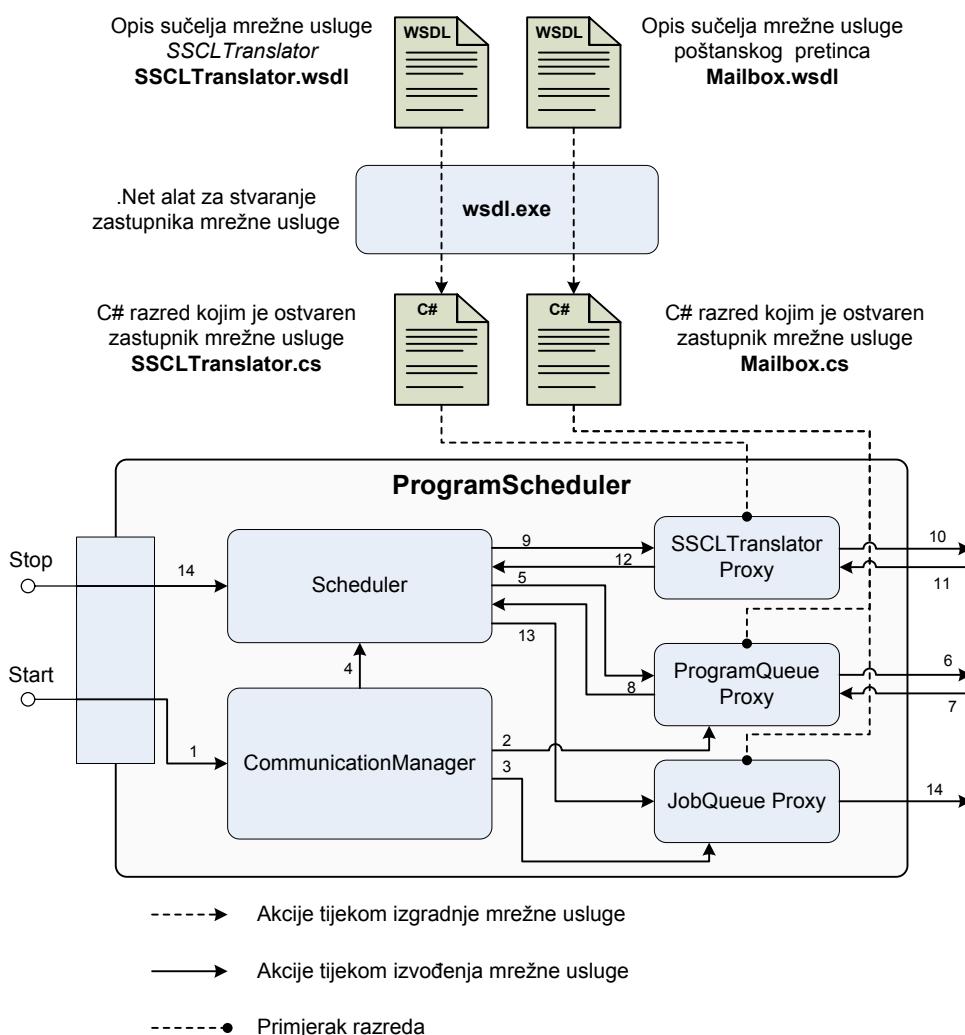
Slika 6.20 Programsko ostvarenje modula SSCL Prevoditelj

Programsko sučelje mrežne usluge *SSCLTranslator* definira samo metodu *Translate(string SSCLCode)*. Slijed operacija koje se izvode nakon poziva operacije *Translate* prikazan je na slici 6.20. SSCL program koji je naveden kao parametar operacije *Translate* izravno se proslijeđuje istoimenoj operaciji primjerka razreda *SSCLCompiler*(1). Tijekom prevođenja programa primjerak razreda *WSDLManager* pristupa imenicima mrežnih usluga. Prije pristupa imenicima mrežnih usluga, *WSDLManager* poziva metodu razreda *Configurator* za dohvatac adrese imenika mrežne usluge (2). *Configurator* dohvatac parametre zapisane u datoteci *Web.config* (3) i proslijeđuje ih primjerku razreda *WSDLManager* (4). Nakon dohvatac adrese imenika mrežnih usluga, razred *WSDLManager* dohvatac potrebne WSDL opise mrežnih usluga i proces prevođenja se nastavlja. Nakon završetka procesa prevođenja,

generirani CL program se putem sučelja mrežne usluge vraća korisniku koji je pozvao metodu mrežne usluge (5).

### *Programsko ostvarenje modula Raspoređivač programa*

Arhitektura modula *Raspoređivač programa* opisana je u poglavljiju 6.3.2. Modul *Raspoređivač programa* sastoji se od modula *Spremnik SSCL programa* i modula *Upravljač raspoređivanjem*. Modul *Spremnik SSCL programa* ostvaren je primjerkom mrežne usluge poštanskog pretinca. Mrežna usluga poštanskog pretinca je usluga s očuvanjem stanja te ostvaruje FIFO spremnik raspodijeljenih programa.



**Slika 6.21 Programsко ostvarenje modula Raspoređivač Programa**

*Upravljač raspoređivanjem* ostvaren je mrežnom uslugom *ProgramScheduler* koja je prikazana na slici 6.21. *ProgramScheduler* povezuje spremnik SSCL programa, mrežnu uslugu za prevođenje *SSCLTranslator* i spremnik CL programa sustava za izvođenje raspodijeljenih programa. *ProgramScheduler* sadrži primjerak razreda *Scheduler* koji

ostvaruje programsku logiku procesa raspoređivanja, zastupnike mrežnih usluga s kojima usluga *ProgramScheduler* komunicira te primjerak razreda *CommunicationManager* koji je zadužen za postavljanje komunikacijskih parametara. Usluga *ProgramScheduler* pristupa mrežnoj usluzi za prevođenje, usluzi spremnika SSCL programa i usluzi spremnika CL programa za koje koristi zastupnike *SSCLTranslatorProxy*, *ProgramQueueProxy* i *JobQueueProxy*.

Zastupnici mrežnih usluga generirani su na osnovi WSDL opisa mrežnih usluga uporabom alata *wsdl.exe*. Alat *wsdl.exe* dio je .Net razvojne potpore koji omogućuje automatsko generiranje C# programskog kôda potrebnog za programski pristup mrežnoj usluzi. Zastupnik *SSCLTranslatorProxy* generiran je iz WSDL opisa mrežne usluge *SSCLTranslator*. Zastupnici *ProgramQueueProxy* i *JobQueueProxy* generirani su iz WSDL opisa mrežne usluge poštanskog pretinca.

Programsko sučelje mrežne usluge sadrži metode za pokretanje i za zaustavljanje procesa raspoređivanja. Tijekom pokretanja procesa raspoređivanja pozivom operacije *Start*, potrebno je navesti pokazatelje primjeraka mrežne usluge poštanskog pretinca koji se koriste kao spremnik SSCL programa i spremnik CL programa. Navedeni pokazatelji proslijeduju se primjerku razreda *CommunicationManager* (1) koji popunjava komunikacijske parametre zastupnika mrežnih usluga *ProgramQueueProxy* i *JobQueueProxy* (2, 3). Nakon toga, pokreće se proces raspoređivanja (4). Proces raspoređivanja sastoji se od koraka 5-13. Razred *Scheduler* upućuje zastupniku spremnika programa zahtjev za dohvatom SSCL programa (5). Zastupnik dohvaća SSCL program iz spremnika programa (6) i primljeni SSCL program (7) proslijedi razredu *Scheduler* (8). Dohvaćeni SSCL program šalje se zastupniku *SSCLTranslatorProxy* (9) koji proslijedi program na prevođenje (10) usluzi *SSCLTranslator*, prima prevedeni CL program (11) i proslijedi ga modulu *Scheduler* (12). U posljednjem koraku, prevedeni CL program šalje se raspodijeljenom sustavu za izvođenje CL programa posredstvom zastupnika *JobQueueProxy* (13). Proces raspoređivanja ponavlja se od koraka 5 sve dok korisnik ne pozove metodu *Stop*. Pozivom naredbe *Stop* zaustavlja se iterativno izvođenje koraka 5-13 kojima upravlja objekt *Scheduler*, odnosno zaustavlja se proces prevođenja.

## 7 Zaključak

Većina korisnika računalnih sustava nisu stručnjaci iz područja računarstva i nemaju znanja potrebna za oblikovanje i izgradnju programskih sustava koji bi im pomogli u obavljanju svakodnevnih poslovnih i privatnih zadaća. Programiranje prilagođeno krajnjem korisniku je grana računarstva kojoj je cilj oblikovanje programskih modela, alata i tehnologija koje će krajnjim korisnicima bez prethodnog znanja o programiranju omogućiti razvoj programskih sustava.

Računarstvo zasnovano na uslugama prikladno je za oblikovanje programskih sustava od strane krajnjih korisnika. Načela računarstva zasnovanog na uslugama omogućuju brzu i jednostavnu izgradnju raspodijeljenih programskih sustava. Osnovni koncept računarstva zasnovanog na uslugama su mrežne usluge, a osnovna metoda izgradnje primjenskih sustava zasnovanih na uslugama je kompozicija mrežnih usluga. Iako je kompozicija mrežnih usluga metoda izgradnje programskih sustava prilagođena krajnjim korisnicima, njena primjena od strane krajnjih korisnika nije raširena. Ograničena primjena kompozicije usluga od strane krajnjih korisnika posljedica je nedostatka prikladnog programskog modela i programskih jezika prilagođenih krajnjim korisnicima.

Programski model zasnovan na uslugama je razvojni model prilagođen krajnjim korisnicima. Prema načelima programskog modela zasnovanog na uslugama, programski sustavi izgrađuju se korištenjem funkcionalnosti primjenskih mrežnih usluga, a koordinacijska logika programskog sustava definira se raspodijeljenim programima i uslugama suradnje i natjecanja.

U magistarskom radu opisan je jezik SSCL (*Simple Service Composition Language*) za definiciju kompozicije mrežnih usluga ostvarene prema načelima programskog modela zasnovanog na uslugama. Jezik SSCL namijenjen je izgradnji raspodijeljenih programa. Izgradnja raspodijeljenih programa u jeziku SSCL prilagođena je krajnjim korisnicima računalnih sustava koji imaju ograničena znanja o tehnikama programiranja i programskim jezicima. Jezik je oblikovan prema načelima skriptnih programskih jezika i isključivo je namijenjen povezivanju funkcionalnosti mrežnih usluga. Definirani skup naredbi sastoji se od naredbi za poziv operacija primjenskih mrežnih usluga te naredbi upravljanja uslugama za suradnju i natjecanje. Zbog ograničene primjene i malog skupa naredbi, sintaksna i semantička pravila jezika SSCL su vrlo jednostavna čime je proces učenja jezika skraćen i prilagođen krajnjim korisnicima. Unatoč malom skupu definiranih naredbi, izražajnost jezika SSCL jednaka je izražajnosti normiranih jezika za opis kompozicije mrežnih usluga. U radu je provedena analiza izražajnosti jezika SSCL prikazom programskog ostvarenja uzoraka tijeka

izvođenja koji su zajednički ostalim jezicima za opis raspodijeljenih primjenskih sustava zasnovanih na uslugama.

Razvojem primjenskih sustava uporabom jezika SSCL pokazano je da su programi za dva reda veličine manji od istovjetnih programa napisanih u drugim programskim jezicima za opis kompozicije usluga, poput jezika BPEL4WS. Nadalje, smanjenje broja naredbi programa napisanih u jeziku SSCL ubrzava izgradnju raspodijeljenih primjenskih sustava. Vrijeme razvoja raspodijeljenih primjenskih sustava uporabom jezika SSCL smanjeno je za red veličine u odnosu na razvoj istovjetnih primjenskih sustava uporabom programskega jezika CL.

U okviru magistarskog rada programski je ostvaren raspodijeljeni sustav prevođenja raspodijeljenih programa napisanih u jeziku SSCL u raspodijeljene programa napisane u CL programskom jeziku. Raspodijeljeni sustav prevođenja ostvaren je prema načelima arhitektura zasnovanih na uslugama i sastoji se od slabo povezanih prevoditelja raspodijeljenih na radna računala povezana globalnom mrežom Internet. Ostvareni sustav prevođenja ima svojstva razmjernog rasta, otpornosti na pogreške i mogućnosti izvođenja u raznorodnim raspodijeljenim okolinama. Svojstvo razmjernog rasta osigurano je mogućnošću dinamičke promjene broja prevoditelja za vrijeme izvođenja sustava. Slaba povezanost prevoditelja omogućuje nastavak izvođenja procesa prevođenja u slučaju prestanka rada pojedinih prevoditelja.

Stečena iskustva u oblikovanju korisničkog jezika SSCL bit će osnova daljnog istraživanja i prilagodbe programskega modela zasnovanog na uslugama krajnjim korisnicima. Cilj daljnjih istraživanja je oblikovanje, izgradnja i ispitivanje korisničkog sučelja za oblikovanje kompozicije mrežnih usluga. Poseban značaj u razvoju korisničkog sučelja posvetit će se približavanju procesa izgradnje raspodijeljenih programskih sustava postupcima poznatim krajnjem korisniku.

## 8 Literatura

- [1] T. Erl: "**Service Oriented Architecture: A Field Guide to Integrating XML and Web Services**", *Prentice Hall*, 2004.
- [2] T. Erl: "**Service Oriented Architecture: Concepts, Techology, and Design**", *Prentice Hall*, 2005.
- [3] D. Kaye: "**Loosely Coupled: The Missing Pieces of Web Services**", *RDS Press*, 2003.
- [4] E. Newcomer, G. Lomow: "**Understanding SOA with Web Services**", *Addison Wesley Professional*, 2004.
- [5] S. Srbljić: "**Jezični procesori 2: Analiza i sinteza ciljnog programa**", *Element*, 1. izdanje, siječanj 2002.
- [6] M.P. Papazoglou, D. Georgakopoulos: "**Service Oriented Computing**", *Communications of the ACM*, Volume 46, Number 10, October 2003, pp. 25-28.
- [7] M. N. Huns, M. P. Singh: "**Service-Oriented Computing: Key Concepts and Principles**", *IEEE Internet Computing*, Volume 9, Number 1, January/February 2005.
- [8] R. Sessions: "**Fuzzy Boundaries: Objects, Componentes, and Web Services**", *ACM Queue*, Volume 9, Number 9, December/January 2004.-2005., pp. 40-47.
- [9] 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.
- [10] M.P. Papazoglou: "**Extending the Service-Oriented Architecture**", *Business Integration Journal*, February 2005, pp. 18-21.
- [11] M.P. Papazoglou, D. Georgakopoulos: "**Service Oriented Computing**", *Communications of the ACM*, Volume 46, Number 10, October 2003, pp. 25-28.
- [12] M.P. Papazoglou, J. Dubray: "**A Survey of Web Service Technologies**", *University of Tereno*, Technical report #DIT-04-058, June 2004.
- [13] 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, October 2004, pp. 73-83.

- [14] S. Vinoski: "**WS-Nonexistent Standards**", *IEEE Internet Computing*, Volume 8, Number 6, November/December 2004, pp. 94-96.
- [15] 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>).
- [16] D.C. Fallside, et al.: "**XML Schema Part 0: Premier Second Edition**", *W3C*, October 2004., (<http://www.w3.org/TR/xmlschema-0>).
- [17] T. Bray, D. Hollander, and A. Layman: "**Namespaces in XML**", *World Wide Web Consortium (W3C)*, 1999., <http://www.w3.org/TR/REC-xml-names/>
- [18] D. Box, et al.: "**SOAP 1.1**", *World Wide Web Consortium (W3C) Note*, February 2000., (<http://www.w3.org/TR/SOAP>).
- [19] 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>).
- [20] T. Bellwood, et al.: "**UDDI Version 3.0.2 Published specification**", October 2004., ([http://uddi.org/pubs/uddi\\_v3.htm](http://uddi.org/pubs/uddi_v3.htm)).
- [21] D. Box, E. Christensen, F. Curbera, D. Ferguson, J. Frey, M. Hadley, C. Kaler, D. Langworthy, F. Leymann, B. Lovering, S. Lucco, S. Millet, N. Mukhi, M. Nottingham, D. Orchard, J. Shewchuk, E. Sindambiwe, T. Storey, S. Weerawarana, S. Winkler: "**Web Services Addressing (WS-Addressing)**" , August 2004, <http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/>
- [22] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, A. Nadalin, N. Nagaratnam, M. Nottingham, H. Prafullchandra, C. von Riegen, J. Schlimmer, C. Sharp, and J. Shewchuk: "**Web Services Policy Framework (WS-Policy)**", September 2004, <http://www-128.ibm.com/developerworks/library/specification/ws-polfram/>
- [23] K. Ballinger, D. Box, F. Curbera, S. Davanum, D. Ferguson, S. Graham, C. K. Liu, F. Leymann, B. Lovering, A. Nadalin, M. Nottingham, D. Orchard, C. von Riegen, J. Schlimmer, I. Sedukhin, J. Shewchuk, B. Smith, G. Truty, S. Weerawarana, and P. Yendluri: "**Web Services Metadata Exchange (WS-MetadataExchange)**", September 2004, <http://www-128.ibm.com/developerworks/library/specification/ws-mex/>
- [24] R. Bilorusets, D. Box, L. F. Cabrera, D. Davis, D. Ferguson, C. Ferris, T. Freund, M. A. Hondo, J. Ibbotson, L. Jin, C. Kaler, D. Langworthy, A. Lewis, R. Limprecht, S. Lucco,

- D. Mullen, A. Nadalin, M. Nottingham, D. Orchard, J. Roots, S. Samdarshi, J. Shewchuk and T. Storey: "**Web Services Reliable Messaging Protocol (WS-ReliableMessaging)**", February 2005., <http://www-128.ibm.com/developerworks/library/specification/ws-rm/>
- [25] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, and D. Simon: "**Web Services Security (WS-Security)**", Version 1.0, April 2002, <http://www-128.ibm.com/developerworks/library/ws-secure/>
- [26] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della-Libera, B. Dixon, P. Garg, M. Gudgin, P. Hallam-Baker, M. Hondo, C. Kaler, H. Lockhart, R. Martherus, O. H. Maruyama, A. Nadalin, N. Nagaratnam, A. Nash, R. Philpott, D. Platt, H. Prafullchandra, M. Sahu, J. Shewchuk, D. Simon, D. Srinivas, E. Waingold, D. Waite, D. Walter, and R. Zolfonoon: "**Web Services Trust Language (WS-Trust)**", February 2005., <http://www-128.ibm.com/developerworks/webservices/library/specification/ws-trust/>
- [27] S. Anderson, J. Bohren, T. Boubez, M. Chanliau, G. Della-Libera, B. Dixon, P. Garg, M. Gudgin, S. Hada, P. Hallam-Baker, M. Hondo, C. Kaler, H. Lockhart, R. Martherus, H. Maruyama, A. Nadalin, N. Nagaratnam, A. Nash, R. Philpott, D. Platt, H. Prafullchandra, M. Sahu, J. Shewchuk, D. Simon, D. Srinivas, E. Waingold, D. Waite, D. Walter, and R. Zolfonoon: "**Web Services Secure Conversation Language (WS-SecureConversation)**", February 2005., <http://www-128.ibm.com/developerworks/library/specification/ws-secon/>
- [28] S. Bajaj, G. Della-Libera, B. Dixon, M. Dusche, M. Hondo, M. Hur, C. Kaler, H. Lockhart, H. Maruyama, A. Nadalin, N. Nagaratnam, A. Nash, H. Prafullchandra, and J. Shewchuk: "**Web Services Federation Language (WS-Federation)**", Version 1.0, July 2003., <http://www-128.ibm.com/developerworks/library/specification/ws-fed/>
- [29] F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, D. Orchard, J. Shewchuk, and T. Storey: "**Web Services Coordination (WS-Coordination)**", 2002., <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-coordination.asp>.
- [30] W. Cox, F. Cabrera, G. Copeland, T. Freund, J. Klein, T. Storey, S. Thatte: "**Web Services Transaction (WS-Transaction)**", January 2004., <http://dev2dev.bea.com/pub/a/2004/01/ws-transaction.html>

- [31] **"Workflow Process Definition Interface - XML Process Definition Language"**, Workflow Management Coalition, Workflow Standard, Document Number WFMC-TC-1025, October 2002., <http://www.wfmc.org/standards/XPDL.htm>
- [32] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, S. Weerawarana: **"The Next Step in Web Services"**, *Communications of the ACM*, Volume 46, Number 10, October 2003.
- [33] M. Turner, D. Budgen, P. Brereton: **"Turning Software into a Service"**, *Computer*, Volume 36, Number 10, October 2003., pp. 38-44.
- [34] N.Milanovic, M.Malek: **"Current Solutions for Web Service Composition"**, *IEEE Internet Computing*, Volume 8, Number 6, November/December 2004, pp.51-59.
- [35] B. Srivastava, J. Koehler: **"Web Service Composition - Current Solutions and Open Problems"**, *International Conference on Automated Planning and Scheduling (ICAPS), Workshop on Planning for Web Services*, Trento, Italy, June 2003.
- [36] 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.
- [37] C. Peltz: **"Web Services Orchestration and Choreography"**, *Computer*, Volume 36, Number 10, October 2003, pp. 46-52.
- [38] 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.
- [39] J. Pasley: **"How BPEL and SOA Are Changing Web Services Development"**, *IEEE Internet Computing*, Volume 9, Number 3, May/June 2005. pp. 60-67.
- [40] 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>).
- [41] S. Thatte: **"XLANG: Web Services for Business Process Design"**, Microsoft, 2001.
- [42] F. Leymann: **"Web Services Flow Langauges (WSFL)"**, IBM Software Group, Version 1.0, 2001.

- [43] 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>).
- [44] A. Barros, M. Dumas, and P. Oaks: "**A Critical Overview of the Web Services Choreography Description Language (WS-CDL)**", March, 2005. (<http://www.bptrends.com/publicationfiles/03%2D05%20WP%20WS%2DCDL%20Barros%20et%20al%20pdf>)
- [45] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takaci-Nagy, I. Trickovic and S. Zimek: "**Web Service Choreography Interface (WSCI) 1.0**", August 2003., (<http://www.w3.org/TR/wsci/>).
- [46] S. A. White: "**Business Process Modelling Notation (BPMN) 1.0**", May 2004. (<http://www.bpmn.org>)
- [47] A. Arkin: "**Business Process Modelling Language (BPML)**", November 2002. (<http://xml.coverpages.org/ni2002-11-13-a.html>)
- [48] M. Havey: "**BPM Theory for Laymen**", *Web Services Journal*, Volume 5, Number 5, May 2005. pp. 10-18.
- [49] C.A. Ellis: "**Information Control Nets: A Mathematical Model of Office Information Flow**" *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems, ACM Press*, Boulder, Colorado, 1979., pp. 225–240
- [50] M.D. Zisman: "**Representation, Specification and Automation of Office Procedures**", *PhD thesis*, University of Pennsylvania, Warton School of Business, 1977.
- [51] 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.
- [52] R. Milner: "**The Polyadic n-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>).
- [53] T. Berners-Lee, J. Hendler, O. Lassila: "**The Semantic Web**", *Scientific American*, May 2001., pp. 34-43.

- [54] G. Denker, J. R. Hobbs, D. Martin, S. Narayanan, and R. Waldinger: "**Accessing Information and Services on the DAML-Enabled Web**", Semantic Web Working Symposium, Stanford CA, May 2001.
- [55] G. Klyne, J. Carroll: "**Resource Description Framework (RDF): Concepts and Abstract Syntax**", January 2003., (<http://www.w3.org/TR/rdf-concepts>).
- [56] S. A. McIlraith, T. C. Son, H. Zeng: "**Semantic Web Services**", *IEEE Intelligent Systems*, Volume 16, Number 2, March/April 2001. pp. 46-53.
- [57] T. Berners-Lee, J. Hendler, and O. Lassila: "**The Semantic Web**", *Scientific American*, 284(5), , May 2001, pp. 34-43.
- [58] OWL-S Coalition: "**OWL-S 1.0 Release**", (<http://www.daml.org/services/owl-s/1.0/>).
- [59] 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.
- [60] Deborah L. McGuinness, Frank van Harmelen. "**OWL Web Ontology Language Overview**", *World Wide Web Consortium (W3C) Candidate Recommendation*. August 2003., (<http://www.w3.org/TR/owl-features/>).
- [61] "**The WS-Resource Framework (WSRF)**", March 2004, (<http://www.globus.org/wsrf/>)
- [62] 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 1, July 2003., pp. 5-51.
- [63] A. Tanenbaum, M. van Steen: "**Distributed Systems: Principles and Paradigms**", Prentice-Hall, January 2002.
- [64] I. Morsch, G. Stevens, M. Won, M. Klann, Y. Dittrich, and V. Wulf: "**Component-based Technologies for End-User Development**", *Communications of ACM*, Volume 47, Number 9, September 2004., pp. 59-62.
- [65] M. Burnett, C. Cook, and G. Rothermel: "**End-User Software Engineering**", *Communications of ACM*, Volume 47, Number 9, September 2004., pp. 53-58.

- [66] A. Repenning and T. Sumner: "**Agentsheets: A medium for creating domain-oriented visual languages**" *IEEE Computer*, Volume 28, Number 3, March 2005., pp.17–25
- [67] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandijev: "**Meta-design: A Manifesto for End-User Development**", *Communications of ACM*, Volume 47 Number 9, September 2004., pp. 33-37.
- [68] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece: "**Software Cost Estimation with COCOMO II**", Prentice Hall PTR, Upper Saddle River, NJ, 2000.
- [69] M.F. Costabile, D. Fogli, C. Letondal, P. Mussio, A. Piccinno: "**Domain-Expert Users and their Needs of Software Development**", Special Session on EUD, UAHCII Conference, Kreta, June 2003.
- [70] A. Sutcliffe, D. Lee and N. Mehandijev: "**Contributions, Costs and Prospects for End-User Development**", Proceedings HCI International 2003.
- [71] F. Paterno, M. Klann, and V. Wulf: "**End-User Development: Empowering people to flexibly employ advanced information and communication technology**", *Research Agenda nad Roadmap for EUD*, December 2003.
- [72] J. Ousterhout, "Tcl and the Tk Toolkit", *Addison-Wesley*, ISBN 0-201-63337-X, 1994.
- [73] A. G. Yoder and D. L. Cohn: "**Real Spreadsheets for Real Programmers**", *Proceedings of the 1994 IEEE Conference on Computer Languages (ICCL'94)*, pp. 20-30
- [74] D. Čapalija: "**Objava-preplata mehanizmi za ostvarivanje mreža zasnovanih na sadržaju**", *Diplomski rad*, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, Zagreb, 2005.
- [75] M. Podravec: "**Otkrivanje i postavljanje usluga u sustavima zasnovanim na uslugama**", *Magistarski rad*, Fakultet elektrotehnike i računarstva, Zagreb, 2005.
- [76] D. Škvorc: "**Prividna mreža računalnih sustava zasnovanih na uslugama**", *Magistarski rad*, Fakultet elektrotehnike i računarstva, Zagreb, 2005.
- [77] D. Skrobo: "**Raspodijeljeno usporedno interpretiranje programa u arhitekturama zasnovanim na uslugama**", *Magistarski rad*, Fakultet elektrotehnike i računarstva, Zagreb, 2005.

- [78] M. Popović: "**Nadziranje pristupa računalnim sustavima zasnovanim na uslugama**", *Magistarski rad*, Fakultet elektrotehnike i računarstva, Zagreb, 2005.
- [79] A. Milanović: "**Programski model zasnovan na uslugama**", *Doktorska disertacija*, Fakultet elektrotehnike i računarstva, Zagreb, 2005.
- [80] D. Skrobo, A. Milanović, S. Srbljić: "**Distributed Program Interpretation in Service-Oriented Architectures**", *Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI'05)*, Orlando, Florida, U.S.A., 2005, pp. 193-197.
- [81] M. Podravec, I. Skuliber, S. Srbljić: "**Service Discovery and Deployment in Service-Oriented Computing Environment**", *Proceedings of the 9th World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI'05)*, Orlando, Florida, U.S.A., 2005, pp. 5-10.
- [82] A. Milanović, S. Srbljić, D. Skrobo, D. Čapalija, and S. Rešković: "**Coopetition Mechanisms for Service-Oriented Distributed Systems**", *Proceedings of the CCCT 2005 (The 3<sup>d</sup> International Conference on Computing, Communications and Control Technologies)*, Austin, Texas, USA, July 2005, Vol. 1, pp. 118-123.

## 9 Životopis

Rođen sam 30. travnja 1980. godine u Zagrebu. Osnovnu i srednju školu pohađao sam u Bjelovaru. Nakon završetka Prirodoslovno-matematičke gimnazije u Bjelovaru, godine 1998. upisujem dodiplomski studij na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Za vrijeme studija radim na nekoliko studentskih projekata na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave. Tijekom posljednje dvije godine studija radim kao demonstrator na Zavodu za elektroniku, mikroelektroniku, računalne i intelligentne 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 završetka studija s naglaskom na znanstveno-istraživačkom radu. Diplomirao sam u lipnju 2003. na studiju računarstva s diplomskim radom pod naslovom "Raspodijeljena priručna memorija posredničkog sustava". Rezultate diplomskog rada ostvarujem u sklopu projekta u suradnji sa tvrtkom Ericsson Nikola Tesla d.d., Zagreb, a pod vodstvom Prof. dr. sc. Siniše Srbljića.

Akademске godine 2003./2004. upisujem poslijediplomski studij za stjecanje akademskog stupnja magistra znanosti. Od siječnja 2004. godine do rujna 2004. godine bio sam zaposlen u Erste&Steiermärkische Bank d.d., na radnom mjestu projektanta sistemske podrške. Od rujna 2004. godine zaposlen sam na Zavodu za elektroniku, mikroelektroniku, računalne i intelligentne 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 projekta "Ericsson Summer Camp 2005" kao voditelj studenata dodiplomske nastave. U okviru nastavnih aktivnosti na Zavodu, aktivno sudjelujem u provedbi laboratorijskih vježbi iz kolegija "Automati, formalni jezici i jezični procesori I", "Automati, formalni jezici i jezični procesori II", "Mreže računala", "Operacijski sustavi I" i "Operacijski sustavi II".

U svojem dosadašnjem znanstveno-istraživačkom sudjelovanju sam sa dva članka na međunarodnim konferencijama.

## 10 Sažetak

Programska arhitektura zasnovana na uslugama omogućuje jednoobrazno povezivanje raznorodnih programskih sustava te brzu i jednostavnu izgradnju raspodijeljenih programskih sustava. Programski model zasnovan na uslugama je krajnjim korisnicima prilagođen model izgradnje raspodijeljenih programskih sustava zasnovanih na uslugama. Sustavi oblikovani prema načelima programskog modela zasnovanog na uslugama izgrađuju se od primjenskih mrežnih usluga, usluga suradnje i natjecanja te raspodijeljenih programa. Raspodijeljeni programi koriste funkcionalnosti primjenskih mrežnih usluga te uporabom usluga suradnje i natjecanja definiraju koordinacijsku logiku cjelokupnog raspodijeljenog programskog sustava.

U magistarskom radu opisan je korisnički jezik SSCL (*Simple Service Composition Language*) za izgradnju raspodijeljenih programa. Izgradnja raspodijeljenih programa u jeziku SSCL prilagođena je krajnjim korisnicima računalnih sustava koji imaju ograničena znanja i iskustva o tehnikama programiranja. Jezik SSCL oblikovan je prema načelima skriptnih programskih jezika i isključivo je namijenjen povezivanju funkcionalnosti mrežnih usluga. Zbog ograničene primjene i malog skupa naredbi, sintaksna i semantička pravila jezika SSCL su vrlo jednostavna čime je proces učenja jezika skraćen i prilagođen korisnicima bez prethodnog iskustva u programiranju. U radu je na nizu primjera uzorka tijeka izvođenja provedena analiza izražajnosti jezika SSCL, a dobiveni rezultati pokazuju dobru izražajnost unatoč malom skupu naredbi jezika. Prevođenje raspodijeljenih programa napisanih u jeziku SSCL ostvareno je raspodijeljenim sustavom prevođenja. Raspodijeljeni sustav prevođenja ostvaren je u skladu s načelima arhitektura zasnovanih na uslugama čime je omogućena njegova uporaba u raznorodnoj okolini mreže Internet.

# 11 Summary

## **"End-User Language for Service-Oriented Programming Model"**

Service Oriented Architecture (SOA) promotes services as basic building elements for rapid development of distributed applications and integration of heterogeneous systems. Service-oriented programming model (SOPM) is well-suited for end-user development of service-oriented applications. Service-oriented applications developed using SOPM consist of application services, distributed programs, and coopetition mechanisms. Distributed programs utilize application services and define overall application's coordination logic by using functionalities of coopetition services.

The master thesis describes an end-user service composition language named *Simple Service Composition Language* (SSCL) that enables development of distributed programs. SSCL language is well-suited for end-users that are not familiar with programming languages and have little experience with programming techniques. SSCL language is based on principles of script languages and its only purpose is description of service composition logic. As SSCL language is a domain specific language with small statement set, its syntax and semantic rules are simple. Simplicity of the language improves the language learning process and adapts it to end-users. As a part of the master thesis, the expressiveness of the SSCL language was analyzed by implementing a set of workflow patterns. The results of the analysis show that the SSCL language is highly expressive regardless of its simplicity. Distributed programs written in SSCL language are translated to interpretable language by distributed translation system. Distributed translation system is designed according to SOA principles and is implemented using standard Web Services technologies. Service oriented implementation enables execution of translation system in heterogeneous environments such as Internet.

## 12 Ključne riječi

Korisnički programske jezici, prevodenje programa, programiranje prilagođeno krajnjem korisniku, raspodijeljeni sustavi zasnovani na uslugama, arhitektura zasnovana na uslugama, kompozicija usluga zasnovana na opisima procesa.

End-user programming languages, program translation, end-user programming, service-oriented distributed systems, service-oriented architecture, process-oriented service composition.

# Dodatak A

## Leksička pravila jezika SSCL

Leksička pravila jezika SSCL opisana su regularnim izrazima navedenim u tablici A-1. Regularni izraz  $r_1$  opisuje leksičke jedinke razreda *ključna riječ*, regularni izraz  $r_2$  opisuje leksičke jedinke razreda *pokazatelj*, a regularni izraz  $r_3$  opisuje leksičke jedinke razreda *konstanta*. Jezik SSCL definira samo operator pridruživanja koji je definiran regularnim izrazom  $r_4$ . Specijalni znakovi definirani jezikom SSCL su znakovi navoda ' " ' i znak nabrajanja ', '.

**Tablica A-1** Leksičke jedinke jezika SSCL opisane regularnim izrazima

$r_1 =$	program + endprogram + service + invoke + if + elseif + else + endif + while + endwhile + createbinarysemaphore + destroybinarysemaphore + obtainbinarysemaphore + releasebinarysemaphore + createcountingsemaphore + destroycountingsemaphore + releasecountingsemaphore + createmailbox + destroymailbox + putmessage + getmessage + createeventchannel + destroyeventchannel + subscribe + unsubscribe + publish + republish+ unpublish + variable
$r_2 =$	<b>(a+b+c+...+z+A+B+C+...+Z) (a+b+c+...+z+A+B+C+...+Z+0+1+2+...+9+_)*</b>
$r_3 =$	" <b>(a+b+c+...+z+A+B+C+...+Z+0+1+2+...+9+_)*</b> "
$r_4 =$	=
$r_5 =$	" + ,

## Dodatak B

### Sintaksna pravila jezika SSCL

Sintaksna pravila jezika SSCL formalno su opisana gramatikom čije produkcije su prikazane u tablici B-1. Nezavršni znakovi gramatike su označeni zagrada  $<>$ , a početni nezavršni znak je  $<\text{Program}>$ . Svi ostali znakovi gramatike su završni znakovi. Masno otisnuti završni znakovi označavaju ključne riječi jezika SSCL, dok završni znakovi *KONSTANTA* i *POKAZATELJ* predstavljaju istoimene leksičke jedinke definirane u dodatu A.

**Tablica B-1** Producije gramatike jezika SSCL

Producija	Opis
$\langle\text{Program}\rangle \rightarrow \textbf{program } \langle\text{Naziv}\rangle \langle\text{Tijelo programa}\rangle \textbf{endprogram}$	Osnovna struktura SSCL programa
$\langle\text{Naziv}\rangle \rightarrow \textbf{KONSTANTA}$	
$\langle\text{Tijelo programa}\rangle \rightarrow \langle\text{Blok deklaracije}\rangle \langle\text{Blok naredbi}\rangle$	
$\langle\text{Blok deklaracije}\rangle \rightarrow \langle\text{Def imena usluga}\rangle \langle\text{Deklaracije var}\rangle$	
$\langle\text{Blok naredbi}\rangle \rightarrow \langle\text{Naredba}\rangle \langle\text{Blok naredbi}\rangle$	
$\langle\text{Blok naredbi}\rangle \rightarrow \varepsilon$	
$\langle\text{Def imena usluga}\rangle \rightarrow \langle\text{Definicija}\rangle \langle\text{Definicija imena usluga}\rangle$	Blok definicije imena usluga
$\langle\text{Def imena usluga}\rangle \rightarrow \varepsilon$	
$\langle\text{Definicija}\rangle \rightarrow \textbf{service } \textit{POKAZATELJ}, \langle\text{URL}\rangle$	
$\langle\text{URL}\rangle \rightarrow \textbf{KONSTANTA}$	
$\langle\text{Deklaracije var}\rangle \rightarrow \langle\text{Deklaracija}\rangle \langle\text{Deklaracije varijabli}\rangle$	Blok deklaracije varijabli
$\langle\text{Deklaracije var}\rangle \rightarrow \varepsilon$	
$\langle\text{Deklaracija}\rangle \rightarrow \textbf{variable } \langle\text{Lista identif}\rangle$	
$\langle\text{Lista identif}\rangle \rightarrow \textit{POKAZATELJ} \langle\text{Nastavak liste}\rangle$	
$\langle\text{Nastavak liste}\rangle \rightarrow , \textit{POKAZATELJ} \langle\text{Nastavak liste}\rangle$	
$\langle\text{Nastavak liste}\rangle \rightarrow \varepsilon$	

<p>&lt;Naredba&gt; → &lt;Invoke&gt;   &lt;Assign&gt;   &lt;While&gt;   &lt;If&gt;            &lt;CreateMB&gt;   &lt;DestroyMB&gt;   &lt;PutMessage&gt;            &lt;GetMessage&gt;   &lt;CreateBinS&gt;   &lt;DestroyBinS&gt;            &lt;ObtainBinS&gt;   &lt;ReleaseBinS&gt;   &lt;CreateCntS&gt;            &lt;DestroyCntS&gt;   &lt;ObtainCntS&gt;   &lt;ReleaseCntS&gt;            &lt;CreateEch&gt;   &lt;DestroyEch&gt;   &lt;Subscribe&gt;            &lt;UnSubscribe&gt;   &lt;Publish&gt;   &lt;RePublish&gt;            &lt;UnPublish&gt;</p>	<p>Naredbe SSCL jezika</p>
<p>&lt;Invoke&gt; → <b>invoke</b> &lt;invoke Tail&gt;</p>	<p>Naredba poziva primjenskih mrežnih usluga</p>
<p>&lt;Invoke Tail&gt; → &lt;lokalno ime&gt;, &lt;Operacija&gt;, &lt;Lista param&gt;</p>	
<p>&lt;Invoke Tail&gt; → &lt;URL&gt; &lt;Operacija&gt; &lt;Lista param&gt;</p>	
<p>&lt;Invoke Tail&gt; → &lt;URL&gt;, &lt;ime primjerka&gt;, &lt;Operacija&gt;, &lt;Lista param&gt;</p>	
<p>&lt;lokalno ime&gt; → <i>POKAZATELJ</i></p>	
<p>&lt;Operacija&gt; → <i>KONSTANTA</i></p>	
<p>&lt;Lista param&gt; → &lt;Lista identif&gt;</p>	
<p>&lt;Assign&gt; → &lt;Odredište&gt; = &lt;Izvorište&gt;</p>	<p>Naredba pridruživanja</p>
<p>&lt;Odredište&gt; → <i>POKAZATELJ</i></p>	
<p>&lt;Izvorište&gt; → <i>POKAZATELJ</i>   <i>KONSTANTA</i></p>	
<p>&lt;While&gt; → <b>while</b> &lt;Uvjet&gt; &lt;Blok naredbi&gt; <b>endwhile</b></p>	<p>Naredbe upravljanja tijekom izvođenja programa</p>
<p>&lt;Uvjet&gt; → <i>POKAZATELJ</i>   &lt;invoke Tail&gt;</p>	
<p>&lt;If&gt; → <b>if</b> &lt;Uvjet&gt; &lt;Blok naredbi&gt; &lt;If tail&gt; <b>endif</b></p>	
<p>&lt;If Tail&gt; → &lt;Else If&gt; &lt;If tail&gt;</p>	
<p>&lt;If Tail&gt; → &lt;Else&gt;</p>	
<p>&lt;If Tail&gt; → <math>\epsilon</math></p>	
<p>&lt;Else If&gt; → <b>elseif</b> &lt;Uvjet&gt; &lt;Blok naredbi&gt;</p>	
<p>&lt;Else&gt; → <b>else</b> &lt;Blok naredbi&gt;</p>	
<p>&lt;CreateMB&gt; → <b>createmailbox</b> &lt;URL&gt;, &lt;ime primjerka&gt;</p>	<p>Naredbe za korištenje usluge poštanskog pretinca</p>
<p>&lt;DestroyMB&gt; → <b>destroymailbox</b> &lt;URL&gt;, &lt;ime primjerka&gt;</p>	
<p>&lt;PutMessage&gt; → <b>putmessage</b> &lt;URL&gt;, &lt;ime primjerka&gt;, &lt;poruka&gt;</p>	
<p>&lt;GetMessage&gt; → <b>getmessage</b> &lt;URL&gt;, &lt;ime primjerka&gt;, &lt;rezultat&gt;</p>	
<p>&lt;ime primjerka&gt; → <i>KONSTANTA</i></p>	

<code>&lt;poruka&gt;</code>	$\rightarrow POKAZATELJ \mid KONSTANTA$	
<code>&lt;rezultat&gt;</code>	$\rightarrow POKAZATELJ$	
<code>&lt;CreateBinS&gt;</code>	$\rightarrow \text{createbinarysemaphore } <\text{URL}>, <\text{ime primjerka}>$	Naredbe za korištenje usluge binarnog semafora
<code>&lt;DestroyBinS&gt;</code>	$\rightarrow \text{destroybinarysemaphore } <\text{URL}>, <\text{ime primjerka}>$	
<code>&lt;ObtainBinS&gt;</code>	$\rightarrow \text{obtainbinarysemaphore } <\text{URL}>, <\text{ime primjerka}>$	
<code>&lt;ReleaseBinS&gt;</code>	$\rightarrow \text{releasebinarysemaphore } <\text{URL}>, <\text{ime primjerka}>$	
<code>&lt;CreateCntS&gt;</code>	$\rightarrow \text{createcountingsemaphore } <\text{URL}>, <\text{ime primjerka}>, <\text{kapacitet}>$	Naredbe za korištenje usluge općeg semafora
<code>&lt;DestroyCntS&gt;</code>	$\rightarrow \text{destroycountingsemaphore } <\text{URL}>, <\text{ime primjerka}>$	
<code>&lt;ObtainCntS&gt;</code>	$\rightarrow \text{obtaincountingsemaphore } <\text{URL}>, <\text{ime primjerka}>$	
<code>&lt;ReleaseCntS&gt;</code>	$\rightarrow \text{releascountingsemaphore } <\text{URL}>, <\text{ime primjerka}>$	
<code>&lt;kapacitet&gt;</code>	$\rightarrow POKAZATELJ$	
<code>&lt;CreateEch&gt;</code>	$\rightarrow \text{createeventchannel } <\text{URL}>, <\text{ime primjerka}>$	Naredbe za korištenje usluge usmjernika događaja
<code>&lt;DestroyEch&gt;</code>	$\rightarrow \text{destroyeventchannel } <\text{URL}>, <\text{ime primjerka}>$	
<code>&lt;Publish&gt;</code>	$\rightarrow \text{publish } <\text{URL}>, <\text{ime primjerka}>, <\text{tip događaja}>, <\text{dok događaja}>, <\text{id događaja}>$	
<code>&lt;Unpublish&gt;</code>	$\rightarrow \text{unpublish } <\text{URL}>, <\text{ime primjerka}>, <\text{id događaja}>, <\text{rezultat}>$	
<code>&lt;RePublish&gt;</code>	$\rightarrow \text{republish } <\text{URL}>, <\text{ime primjerka}>, <\text{tip događaja}>, <\text{id događaja}>, <\text{dok događaja}>, <\text{id događaja}>$	
<code>&lt;Subscribe&gt;</code>	$\rightarrow \text{subscribe } <\text{URL}>, <\text{ime primjerka}>, <\text{epr}>, <\text{callback}>, <\text{dok pretplate}>, <\text{id pretplate}>$	
<code>&lt;Unsubscribe&gt;</code>	$\rightarrow \text{unsubscribe } <\text{URL}>, <\text{ime primjerka}>, <\text{id pretplate}>, <\text{rezultat}>$	
<code>&lt;tip događaja&gt;</code>	$\rightarrow \text{"exhaustive" } \mid \text{"permanent" } \mid \text{"notification" }$	
<code>&lt;dok događaja&gt;</code>	$\rightarrow POKAZATELJ$	
<code>&lt;id događaja&gt;</code>	$\rightarrow POKAZATELJ$	
<code>&lt;epr&gt;</code>	$\rightarrow POKAZATELJ$	
<code>&lt;callback&gt;</code>	$\rightarrow POKAZATELJ$	
<code>&lt;dok pretplate&gt;</code>	$\rightarrow POKAZATELJ$	
<code>&lt;id pretplate&gt;</code>	$\rightarrow POKAZATELJ$	