

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1535

**OBJAVA/PRETPLATA MEHANIZMI ZA
OSTVARIVANJE MREŽA ZASNOVANIH NA
SADRŽAJU**

Davor Čapalija

Zagreb, lipanj 2005.

Zahvaljujem mentoru prof.dr.sc. Siniši Srbljiću na pruženoj potpori u stručnom i osobnom razvoju, te na pomoći u omogućavanju daljnog školovanja.

Također, zahvaljujem mr.sc. Andri Milanoviću na brojnim korisnim savjetima, primjedbama i pomoći pri pisanju diplomskog rada.

Dipl.ing. Danielu Skrobi i kolegi Saši Reškoviću zahvaljujem na pomoći pri ostvarivanju diplomskog rada.

Sadržaj

1.	UVOD	5
2.	RAČUNARSTVO ZASNOVANO NA USLUGAMA.....	7
2.1.	MREŽNE USLUGE	7
2.2.	RAČUNALNA ARHITEKTURA ZASNOVANA NA USLUGAMA.....	9
2.2.1.	<i>Standard Web services</i>	11
2.2.2.	<i>Prošireni skup standarda WS-*.....</i>	16
2.3.	KOMPOZICIJA USLUGA	19
2.3.1.	<i>Kompozicija usluga zasnovana na procesima.....</i>	19
3.	KOMUNIKACIJA ZASNOVANA NA MODELU OBJAVA/PRETPLATA	21
3.1.	RAZREDBA MODELA MEĐUDJELOVANJA PROCESA U RASPODIJELJENIM SUSTAVIMA.....	22
3.1.1.	<i>Razredba prema modelu koordinacije</i>	23
3.1.2.	<i>Razredba prema obrascima komunikacije i suradnje raspodijeljenih procesa.....</i>	25
3.2.	MODELI KOMUNIKACIJSKIH POSREDNIČKIH SUSTAVA ZA IZGRADNJU RASPODIJELJENIH PROGRAMSKIH SUSTAVA.....	29
3.2.1.	<i>Pozivi udaljenih procedura i pozivi nad udaljenim objektima</i>	30
3.2.2.	<i>Posrednički sustavi zasnovani na razmjeni poruka.....</i>	31
3.2.3.	<i>Dijeljeni podatkovni prostori</i>	32
3.2.4.	<i>Objava/pretplata</i>	33
3.3.	KOMUNIKACIJSKI SUSTAV ZASNOVAN NA MODELU OBJAVA/PRETPLATA	34
3.3.1.	<i>Posrednik u sustavu objava/prelata</i>	34
3.3.2.	<i>Pretplaćivanje i objavljivanje</i>	36
3.3.3.	<i>Mehanizmi pretplaćivanja i filtriranja</i>	38
3.4.	RASPODIJELJENA ARHITEKTURA KOMUNIKACIJSKOG SUSTAVA ZASNOVANOG NA MODELU OBJAVA/PRETPLATA	46
3.4.1.	<i>Prekrivna mreža za usmjeravanje</i>	47
3.5.	POSTOJEĆI SUSTAVI ZASNOVANI NA MODELU OBJAVA/PRETPLATA	48
3.5.1.	<i>TIBCO TIB/Rendezvous Bus</i>	49
4.	MREŽE ZASNOVANE NA SADRŽAJU	51
4.1.	OBLIKOVANJE MREŽE POSREDNIKA	52
4.2.	ARHITEKTURA SUSTAVA	54
4.2.1.	<i>Hijerarhijska korisnik/posrednik arhitektura</i>	55
4.2.2.	<i>Ačiklična arhitektura ravnopravnih sudionika</i>	56
4.2.3.	<i>Općenita arhitektura ravnopravnih sudionika</i>	56
4.3.	ALGORITMI USMJERAVANJA I STRATEGIJE OBRADE DOGAĐAJA	57
4.3.1.	<i>Načela usmjeravanja</i>	57
4.3.2.	<i>Razredi algoritama usmjeravanja</i>	59

4.3.3. <i>Algoritmi i podatkovne strukture</i>	60
5. OKOLINA RASPODIJELJENOG SUSTAVA ZASNOVANOG NA USLUGAMA	66
5.1. MEHANIZMI SURADNJE I NATJECANJA.....	67
5.1.1. <i>Binarni i opći semafor</i>	68
5.1.2. <i>Spremnik poruka</i>	69
6. USMJERNIK DOGAĐAJA ZASNOVAN NA USLUGAMA	70
6.1. ARHITEKTURA USMJERNIKA DOGAĐAJA	71
6.1.1. <i>Okolina usmjernika događaja</i>	73
6.1.2. <i>Vrste postojanosti događaja</i>	77
6.1.3. <i>Održavanje stanja usmjernika događaja</i>	77
6.1.4. <i>Arhitektura instance usmjernika događaja</i>	79
6.1.5. <i>Arhitektura interpretatora</i>	83
6.1.6. <i>Arhitektura korisnika objavitelja i korisnika preplatnika</i>	84
6.2. PROGRAMSKO OSTVARENJE USMJERNIKA DOGAĐAJA	86
6.2.1. <i>Programska arhitektura i osnovni razredi</i>	87
6.2.2. <i>Sučelje mrežne usluge</i>	88
6.2.3. <i>Strukture podataka zasnovane na XML-u</i>	89
6.2.4. <i>Spremnik usmjernika događaja</i>	90
6.2.5. <i>Instanca usmjernika događaja</i>	90
6.2.6. <i>Mehanizmi generiranja lokalnih zastupnika</i>	92
6.2.7. <i>Asinkrono pozivanje interpretatora i dojavljivanje preplatnicima</i>	93
7. ZAKLJUČAK.....	94
LITERATURA	95

1. Uvod

Stalni rast globalne mreže Internet i raspodijeljenih tehnologija potaknuo je pojavljivanje raznorodnih sustava velikih razmjera. Osim toga, prisutan je i trend prelaska s čvrsto povezanih sustava na slabo povezane sustave, koji su izgrađeni prema paradigmi računarstva zasnovanog na uslugama (engl. *Service-oriented computing*, SOC). Računarstvo zasnovano na uslugama postaje novi osnovni koncept za razvoj raznorodnih raspodijeljenih programskih rješenja. SOC se zasniva na uslugama, cjelovitim i autonomnim jedinicama programske potpore, koje se mogu opisati, objaviti, pronaći, orkestrirati i programirati koristeći standardan skup protokola. Osnovna prednost SOC-a je neovisnost o određenom računalnom sklopoljju, programskoj potpori te razvojnoj platformi. [7]

Da bi se omogućio razvoj sustava velikih razmjera koji su slabo povezani, raspodijeljeni sustavi ostvaruju se primjenom koncepta *programiranja na veliko* (engl. *programming in the large*). Suprotno konceptu programiranja na malo (engl. *programming in the small*), programiranje na veliko zasniva se na povezivanju visoko apstraktnih i složenih funkcionalnosti ostvarenih kao usluge. Međutim, koncepti programiranja na malo, kao što su objektno orijentiran razvoj programske potpore i razvoj zasnovan na komponentama, i dalje se koriste za razvoj individualnih usluga te za razvoj izoliranih sustava zasnovanih na jedinstvenoj platformi kao što je CORBA.

Rastom veličine raspodijeljenog sustava povećava se i složenost sustava kojom je potrebno ovladati. Složenost sustava posljedica je velikog broja autonomnih i raznorodnih komponenti od kojih je sačinjen raspodijeljeni sustav. Da bi se riješio problem složenosti i raznorodnosti, razvijeni su posrednički sustavi koji prikrivaju raznorodnost računalnih platformi i pružaju apstraktan i homogen izgled sustava. Najčešće korišteni posrednički sustavi zasnovani na pozivu udaljenih metoda kao što su CORBA i Java RMI namijenjeni su za izgradnju raspodijeljenih programskih sustava srednjih razmjera. Također i posrednički sustavi zasnovani na razmjeni poruka putem repova kao što je *Java Message Service* nisu primjenjivi za razvoj raspodijeljenih sustava velikih razmjera. Osnovni problem navedenih sustava je što pružaju čvrsto povezani *jedan-na-jedan* model komunikacije u kojem određeni sudionik mora poznavati velik broj ostalih sudionika. Stoga rast broja sudionika u sustavima velikih razmjera otežava rukovanje zavisnostima među sudionicima.

Zbog nadilaženja nedostataka koje posjeduju navedeni sustavi, u novije vrijeme razvijaju se sustavi zasnovani na objava/preplata modelu komunikacije koji omogućava slabo povezanu i *više-na-više* komunikaciju. U objava/preplata sustavima jedna skupina korisnika objavljuje sadržaje, dok druga skupina korisnika izražava interes za određene sadržaje pretplaćivanjem. Objava/preplata sustav ispituje sadržaje i dostavlja ih zainteresiranim korisnicima. Kako bi se omogućila primjena u

raspodijeljenim sustavima velikih razmjera razvijeni su objava/preplata sustavi raspodijeljene arhitekture. Sudionici raspodijeljenog objava/preplata sustava čine mrežu zasnovanu na sadržaju. Za razliku od tradicionalnih mreža u kojima se usmjeravanje provodi na osnovi adresa odredišta, u mrežama zasnovanim na sadržaju usmjeravanje je zasnovano na interesima odredišta za određenim sadržajem.

U ovom radu detaljno se razmatra objava/preplata komunikacijski model te svojstva postojećih komunikacijskih sustava zasnovanih na objava/preplata modelu. Također, proučavaju se načela izgradnje mreža zasnovanih na sadržaju te mehanizmi usmjeravanja sadržaja. Nadalje, programski se ostvaruje objava/preplata mehanizam s naprednim mogućnostima tumačenja sadržaja. Arhitektura programskog ostvarenja zasniva se na uslugama i tekstualnoj komunikaciji, a u razvoju se koristi skup standarda WS-*. Tumačenje sadržaja provodi se pomoću interpretatora koji se ostvaraju kao zasebne usluge.

Na početku rada opisana je paradigma računarstva zasnovanog na uslugama te arhitekture sustava zasnovanih na uslugama i standardi prema kojima se oni ostvaraju. U trećem poglavlju uspoređuju se komunikacijski modeli za ostvarivanje komunikacijskih posredničkih sustava. Posebice se razmatraju posrednički sustavi zasnovani na objava/preplata modelu te postupci njihovog oblikovanja i ostvarenja. Četvrto poglavlje sadrži pregled arhitektura i postupaka oblikovanja mreža zasnovanih na sadržaju. Objasnjena su načela i algoritmi usmjeravanja sadržaja u mrežama zasnovanim na sadržaju. U petom poglavlju opisan je raspodijeljeni sustav zasnovan na suradnji i natjecanju (*Coopetition-based distributed system*, CBDS) koji je ostvaren u okviru projekta *CRO-GRID Posrednički sustavi*. Prikazani su mehanizmi suradnje i natjecanja pomoću kojih se izgrađuju raspodijeljeni programski sustavi u CBDS-u. Arhitektura i programsko ostvarenje usmjernika događaja (eng. *Event-channel*) kao naprednog objava/preplata mehanizma zasnovanog na uslugama prikazano je u šestom poglavlju. Na kraju rada izneseno je vrednovanje ostvarenog objava/preplata mehanizma i zaključak.

2. Računarstvo zasnovano na uslugama

U suvremenom računarstvu odvija se pomak u paradigmi oblikovanja, arhitekturi, načinu isporuke i načinu korištenja složenih programskih sustava. Računarstvo zasnovano na uslugama (*engl. service oriented computing, SOC*) nova je paradigma razvoja raspodijeljenih sustava i sustava za e-poslovanje (elektroničko poslovanje, engl. *e-business*). Nova paradigma nadograđuje i proširuje postojeće paradigme objektno orijentiranog računarstva te računarstva zasnovanog na komponentama. Nadalje, računarstvo zasnovano na uslugama omogućava izgradnju agilnih mreža poslovnih programskih sustava koji surađuju unutar i izvan granica organizacijskog okruženja. Izgradnja programskih sustava zasniva se na uslugama koje su autonomni programski konstrukti neovisni o platformi. Usluge su izgrađene tako da ih je moguće opisati, objaviti, pronaći, orkestrirati i konfigurirati na standardiziran način. Navedene ključne značajke usluga ostvaraju se uz pomoć jezika XML koji je postao osnovni standard za povezivanje i zajednički rad složenih programskih sustava velikih razmjera [4].

2.1. Mrežne usluge

U posljednja četiri desetljeća metodologija oblikovanja programskih sustava prošla je kroz mnoge razvojne faze. Glavni uzrok promjene metodologije oblikovanja je brzi rast složenosti programskih sustava. Način na koji se rast složenosti nadilazi je kontinuirano uvođenje programskih konstrukta krupnije zrnatosti. Programski konstrukti razvijali su se postupno, najprije je osnovni konstrukt bila procedura, uslijedio je objekt, pa komponenta, te konačno usluga. Pojam programskog konstrukta uspoređuje se s pojmom *crne kutije* jer poput nje skriva unutarnje detalje programskog ostvarenja te upravlja pristupom ponašanju i podacima putem sučelja. Za oblikovanje programskih sustava na finijoj razini zrnatosti danas se koriste objekti. Pojedini objekti oblikuju se na način da apstrahiraju određene objekte iz stvarnog svijeta. Međutim, teškoće nastaju pri pokušaju grupiranja velikog broja objekata u vrlo složeni sustav. Naime, sitna zrnatost objekata otežava upravljanje zavisnostima među velikim brojem objekata. Kako bi se lakše upravljalo zavisnostima u velikom sustavu koristi se komponenta koja je sljedeći krupniji programski konstrukt. Komponenta je skupina objekata koji surađuju kako bi pružili neku funkcionalnost sustava. Također i komponenta predstavlja *crnu kutiju* jer skriva objekte od kojih se sastoji pomoću sučelja. Ostale komponente ne ovise o unutarnjim objektima pripadne komponente pa se time olakšava razvoj složenog sustava. Danas se koriste tehnologije EJB (*Enterprise Java Beans*), .NET i CORBA koje pružaju učinkovit razvoj programskih komponenti i programskih sustava zasnovanih na komponentama. *Razvoj zasnovan na komponentama* (*engl. component-based development*) olakšava i ubrzava oblikovanje složenih sustava visoke kvalitete zbog

jednostavnog načina upravljanja zavisnostima među komponentama unutar programskog sustava. Nadalje, pri razvoju programskog sustava zasnovanog na komponentama funkcionalnost sustava ostvaruje se povezivanjem komponenata koje su fizički dostupne. Odnosno, funkcionalnosti potrebne za izgradnju sustava nabavljaju se od raznih isporučitelja u obliku komponenti. Međutim, razvojem mreže Internet javljaju se mnoge funkcionalnosti koje su nužno fizički udaljene. Primjerice, funkcionalnosti poput rezervacije letova i pretraživanja poslovnih imenika nije moguće kupiti u obliku fizičkih komponenti. Kao posljedica, u posljednje vrijeme sve prisutniji su raspodijeljeni sustavi koji koriste funkcionalnosti razmještene na više računala koja komuniciraju mrežom Internet. Međutim, za razvoj raspodijeljenih sustava širokih razmjera koji komuniciraju mrežom Internet potrebno je povezati raznorodne računalne platforme ostvarene pomoću .NET, EJB, CORBA tehnologije ili neke starije kao što je COBOL. Prvi problem pri povezivanju je razlika u specifikaciji pozivanja udaljenih metoda. Primjerice, sustav ostvaren pomoću EJB tehnologije zahtjeva pozivanje udaljenih metoda pomoću RMI specifikacije, dok sustav ostvaren CORBA tehnologijom zahtjeva pozivanje putem IIOP specifikacije. Nadalje, komponente pojedinog sustava smještene su unutar lokalnih mreža koje su od ostatka mreže odijeljene vatrozidom koji upravlja prometom poruka. Također, svaka od platformi koristi poseban protokol prijenosa poruka te način zapisivanja poruka, a potrebno je i poznavanje točne adrese komponente. Mada navedene platforme nastoje omogućiti zajednički rad s ostalim platformama, navedene poteškoće to dosta otežavaju. Osim toga, za pozivanje druge komponente potrebno je poznavati javno sučelje te komponente, a promjenom sučelja potrebno je promijeniti i način pozivanja. Također broj udaljenih sučelja na mreži koje komponenta mora poznavati može biti znatan te zbog toga mogu nastati zavisnosti velikih razmjera među komponentama.

Usluga je skup ponašanja koji pruža neka komponenta na korištenje drugoj komponenti na osnovu ugovora o sučelju. Nadalje, usluge pružaju mogućnost jednostavnog povezivanja i zajedničkog rada. Zbog toga je također moguće njihovo dinamičko pronalaženje i korištenje.

Mrežna usluga ima sučelje kojem se pristupa putem mrežne adrese. *Mrežne usluge* (engl. *web services*) su se tek nedavno pojavile kao standard, no ideja pružanja usluga putem WWW-a postoji već dulje vrijeme. Većina korisnika mreže Internet je upotrebljavala ili pružala neku uslugu putem WWW-a. Primjerice, jedna od često korištenih usluga je prosljeđivanje elektronske pošte korisnicima preplaćenim na liste elektronske pošte. [3]

Osnovna razlika između starih i suvremenih mrežnih usluga je način njihovog korištenja. Stare usluge zahtijevaju određen zahvat čovjeka korisnika putem preglednika Web stranica (engl. *web browser*). Suvremene mrežne usluge zasnivaju se na otvorenosti mrežnih sustava. Naime, suvremene mrežne usluge izlažu svoje funkcionalnosti putem programskog sučelja koje se javno objavljuje. Na taj način omogućeno je automatizirano korištenje mrežnih usluga.

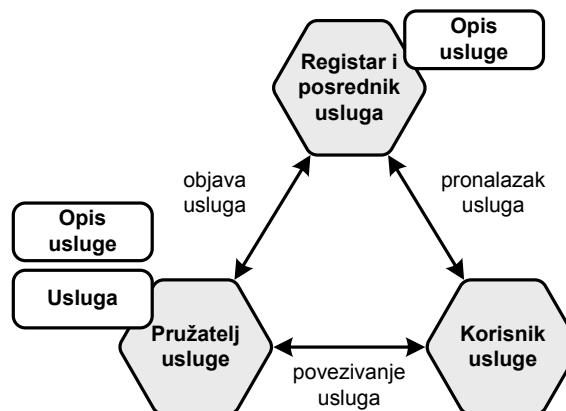
2.2. Računalna arhitektura zasnovana na uslugama

U računarstvu zasnovanom na uslugama naglasak je na cijelokupnoj arhitekturi sustava. Naime, ključne tehnologije za razvoj zasebnih komponenti poput baza podataka i raznih drugih programskih komponenti razvijene su i vrlo dobro su poznate. Stoga stvarna uspješnost raspodijeljenih raznorodnih sustava ovisi o mogućnostima povezivanja postojećih tehnologija u zajednički radni okvir, odnosno cijelokupnu arhitekturu sustava. Zbog toga se nastoji postojeće tehnologije formulirati u obliku metodologija oblikovanja složenih programskih sustava. Formulirane metodologije moguće je objediniti u općenitu infrastrukturu koja olakšava i ubrzava razvoj raspodijeljenih raznorodnih sustava.

U posljednje vrijeme postignut je zamjetan napredak u razvoju standarda i alata vezanih uz mrežne usluge. Suvremene mrežne usluge opisuju se i pozivaju na standardan način. Zbog toga je također omogućeno njihovo objavljivanje i pronalazak na standardan način.

Osnovni model arhitekture zasnovane na uslugama osim samih usluga definira tri vrste sudionika i njihove međuodnose. Model je prikazan na slici 1. Vrste sudionika su sljedeće [33]:

- **Pružatelji mrežnih usluga:** ostvaruju mrežne usluge i objavljaju ih prijavom u registar usluga
- **Registri usluga:** održavaju popis objavljenih usluga i omogućavaju korisnicima usluga pronalazak pružatelja usluga
- **Korisnici usluga:** pretražuju registar usluga za odgovarajućim pružateljima usluga, odabiru pružatelje i koriste pronađene usluge



Slika 1: Osnovni model arhitekture zasnovane na uslugama

Tehnologija *Web services* ostvarenje je modela arhitekture zasnovane na uslugama i suvremeniji je standard. Međutim, *Web services* tehnologija nije nužna i jedina tehnologija za ostvarivanje arhitekture zasnovane na uslugama. Naime, moguće je istovremeno korištenje više arhitektura zasnovanih na uslugama ostvarenih pomoću različitih tehnologija. Uvjet je da sva ostvarenja

zadovoljavaju ključne zahtjeve potrebne da bi neka arhitektura bila zasnovana na uslugama. Stoga se pod pojmom arhitekture zasnovane na uslugama prvenstveno misli na određena svojstva, organizaciju, metode oblikovanja i razvoj složenih računalnih sustava [36].

Ostvarenje mrežnih usluga *Web services* tehnologijom koristi model prema kojem u određenom međudjelovanju korisnik upotrebljava jednu uslugu od jednog pružatelja. Razlog tome je da je većina programskih sustava oblikovana prema korisnik/poslužitelj arhitekturi u kojoj korisnik šalje zahtjev jednom poslužitelju i od istog dobiva odgovor.

Ključni zahtjevi nad arhitekturom zasnovanom na uslugama [3] su:

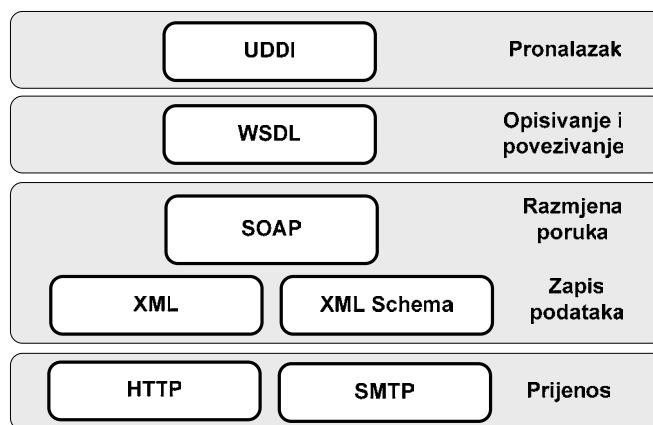
- **Slaba povezanost:** Usluge trebaju biti cjelovite (engl. *self-contained*) i autonomne te oblikovane tako da ne zahtijevaju informacije o internoj strukturi, mehanizmima, pravilima i podatkovnim tipovima drugih usluga. Također, ne zahtijeva se jamčenje konzistentnosti određenog podataka ili stanja tijekom međudjelovanja više usluga. Međutim, moguće je uspostavljanje ugovornih odnosa visoke razine među uslugama za postizanje konzistentnosti na razini sustava.
- **Neutralnost platforme:** Zahtijeva se mogućnost zajedničkog rada i povezivanja usluga samo na osnovi njihovih sučelja. Odnosno, usluge koje međudjeluju s nekom uslugom ne smiju ovisiti o detaljima njezinog programskog ostvarenja. Nadalje, razvoj sustava zasnovan na uslugama ne smije se oslanjati isključivo na neki poseban programski jezik ili platformu, jer bi to onemogućilo slobodu izbora pri razvijanju programskog sustava. Također, oslanjanjem na neku posebnu platformu isključila bi se mogućnost povezivanja većine naslijedenih (engl. *legacy*) programske sustava s novim sustavima zasnovanim na uslugama.
- **Prilagodljiva konfigurabilnost:** Zahtijeva se mogućnost konfiguriranja programskog sustava na jednostavan i prilagodljiv način. Nadalje, zahtijeva se mogućnost povezivanja usluga za vrijeme rada programskega sustava, odnosno zahtijeva se mogućnost mijenjanja konfiguracije sustava dinamički i po potrebi, bez gubitka ispravnosti.
- **Postojanost:** Usluge nisu nužno dugog vijeka, no s obzirom da se međudjelovanje odvija među autonomnim raznorodnim uslugama u dinamičnoj okolini, nužna je mogućnost upravljanja iznimkama. Usluge bi trebale postojati dovoljno dugo da otkriju važne iznimke, poduzmu mjere za ispravljanje tih iznimki, te odgovore na mjere ispravljanja iznimki koje su poduzele druge usluge. Usluge bi također trebale postojati dovoljno dugo da ih je moguće pronaći te da je moguće pouzdati se u njih i njihovo ponašanje.
- **Krupna zrnatost:** Usluge se oblikuju na razini krupne zrnatosti, a također je dovoljno poznavanje ostalih usluga na razini krupne zrnatosti. Umjesto oblikovanja akcija i međudjelovanja na detaljnoj razini, oblikuju se bitna svojstva usluga na visokoj razini koja se

koriste u svrhe definiranja poslovnih ugovora među sudionicima. Također, krupna zrnatost smanjuje ovisnost među sudionicima i potiče komunikaciju putem manjeg broja poruka većeg značaja.

- **Timovi usluga:** Umjesto oblikovanja izračunavanja na centraliziran način, izračunavanje se oblikuje tako da ga provodi više autonomnih ravnopravnih sudionika. Za razliku od centraliziranog izračunavanja gdje jedna komponenta upravlja drugim komponentama, izračunavanje u otvorenim sustavima provodi više partnerskih usluga koje rade kao jedan tim. Stoga je umjesto individualne komponente kao u centraliziranom načinu osnovna jedinka za oblikovanje sustava *tim usluga*. Oblikovanje zasnovano na timu usluga posljedica je razvoja arhitekture ravnopravnih sudionika (engl. *peer to peer architecture*).

2.2.1. Standard Web services

Arhitektura zasnovana na uslugama koristi načela prema kojima se spajanje, komunikacija, opis i pronalazak usluga odvijaju na standardan način. Suvremeni standard *Web services* ostvaruje navedena načela pomoću stoga mehanizama i protokola prikazanih na slici 2 [35, 36]. Na dnu stoga nalazi se prijenosni protokol koji služi za prijenos podataka među sudionicima. Sljedeća razina stoga pruža zajednički način zapisivanja i oblikovanja podataka kako bi se omogućila obrada podataka razmijenjenih između korisnika i pružatelja. Zadaću općenitog načina zapisivanja podataka ispunjava jezik XML [37] koji je danas općeprihvaćen standard. Nadalje, potreban je zajednički protokol za razmjenu poruka među uslugama i pozivanje metoda udaljenih usluga. *Protokol za jednostavni pristup objektima* [38] (engl. *Simple Object Access Protocol*, SOAP) zasnovan na jeziku XML pruža navedene mogućnosti i koristi se u standardu *Web services*. Osim toga, sučelje usluge potrebno je opisati u obliku čitljivom ostalim uslugama. Opis sučelja sastoji se od naziva metoda, njihovih parametra i rezultata izvođenja metoda. Za tu svrhu razvijen je *jezik za opisivanje mrežnih usluga* [39] (engl. *Web service description language*, WSDL).



Slika 2: Osnovni stog protokola i mehanizama *Web services* standarda

Konačno, potreban je mehanizam za pronalaženje usluga. UDDI [40] (*Universal Description, Discovery, and Integration*) je standard za izgradnju registra koji pohranjuje opise usluga i omogućuje njihovo pretraživanje. Osim standarda kao što su XML, SOAP, WSDL i UDDI razvijene su specifikacije semantike za razna područja primjene.

XML

XML (engl. *eXtensible Mark-up Language*) je *proširivi jezik zasnovan na oznakama* koji pruža tekstualni format zapisivanja podataka neovisan o računalnoj platformi. XML je osnovni jezik pomoću kojeg su oblikovani ostali standardni jezici stoga mrežnih usluga. Za razliku od HTML-a gdje oznake određuju način prikazivanja podataka, XML oznake nose informaciju o značenju podataka. Također, XML uvodi dodatna pravila u odnosu na sintaksu HTML-a zbog lakšeg parsiranja i obrade. Nadalje, XML može sadržavati nestrukturirane podatke poput tekstualnih dokumenta i strukturirane podatke kao u bazama podataka. Moguće je i postavljanje upita na sadržaj dokumenta, odnosno moguće je postavljati upite prema strukturi i vrijednostima podataka u XML dokumentu. Suprotno od skupa oznaka jezika HTML koji je unaprijed definiran, skup oznaka XML-a ima svojstvo proširivosti, odnosno skup oznaka moguće je definirati i proširiti ovisno o potrebi. Stoga se XML koristi za razne primjene jer pruža mogućnost definiranja struktura i sintakse ovisno o potrebi. Primjerice, XML se koristi u užim industrijskim primjenama, no također i za izgradnju standardnih jezika.

Skup oznaka koji koristi određeni XML dokument naziva se rječnik (engl. *vocabulary*) XML dokumenta. S obzirom da je razvijeno mnoštvo nezavisnih rječnika XML dokumenata, moguće je da oznake i atributi istih imena posjeduju različito značenje u različitim rječnicima. Da bi se izbjeglo miješanje i zabuna uvodi se pojam prostora imena (engl. *namespace*) koji se zadaje pomoću URI-a (engl. *uniform resource identifier*). Smještanjem u zadani prostor imena svaki element i svaki atribut jednoznačno su određeni.

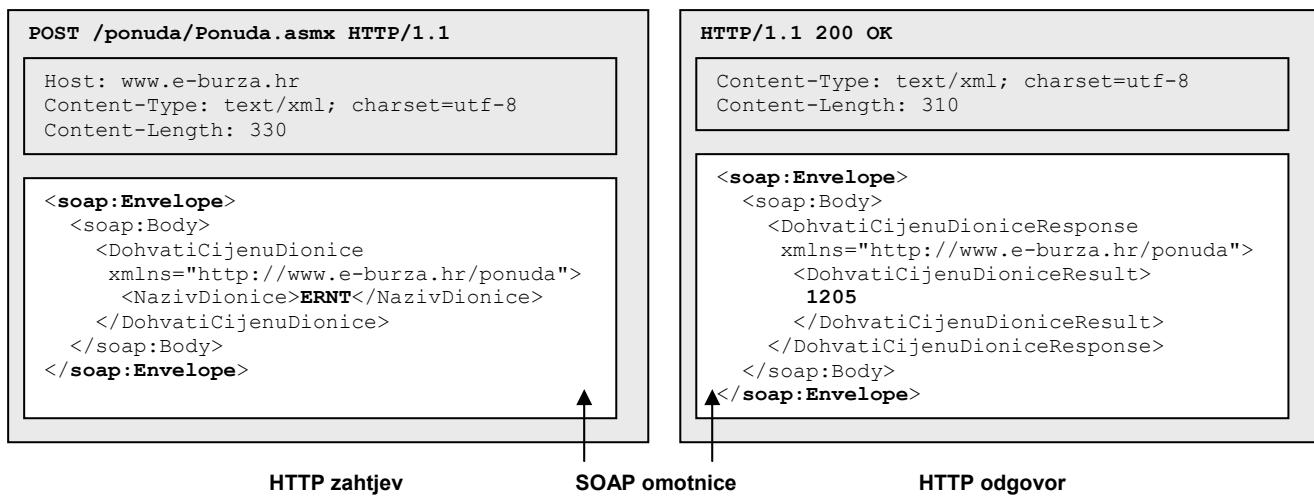
Svaki XML dokument sadrži dva svojstva valjanosti: svojstvo dobre oblikovanosti te svojstvo ispravnosti. XML dokument je *dobro oblikovan* (engl. *well-formed*) ako slijedi određena pravila koja definira XML jezik. Primjerice, XML dokument smije imati samo jedan korijenski element, potrebno je da svaka početna oznaka ima pripadnu završnu oznaku te se zahtijeva da element dijete bude pravilno ugniježđen.

XML dokument sadrži određene podatkovne tipove koji su definirani hijerarhijskom strukturom XML elemenata ovisno o primjeni. *XML Schema* [41] jezik zasnovan na XML-u pruža standardan način definiranja strukture XML dokumenta i podatkovnih tipova. Primjerice, moguće je definirati jednostavni cjelobrojni tip podataka te složeni tip kao što je zapis sastavljen od cjelobrojnog podatka i znakovnog niza. Osim toga, pomoću *XML Schemae* definiraju se ograničenja nad vrijednostima podatkovnih tipova. Stoga je svojstvo ispravnosti (engl. *validity*) XML dokumenta zadovoljeno ako podatkovni tipovi odgovaraju definiranim tipovima u pridruženom *XML Schema* dokumentu te ako

vrijednosti tipova zadovoljavaju postavljena ograničenja. Programska komponenta koja provjerava ispravnost XML dokumenata obično se naziva XML *validator*.

SOAP

SOAP je jednostavan protokol zasnovan na jeziku XML namijenjen za razmjenu strukturiranih podataka definiranih tipova među ravnopravnim sudionicima u raspodijeljenoj okolini. U početku SOAP je prvenstveno korišten kao mehanizam udaljenog poziva procedura (RPC) putem XML formata među umreženim računalima. Međutim, u posljednje vrijeme razvio se kao standard za razmjenu XML poruka u mreži Internet koristeći HTTP, SMTP i SIP kao prijenosne protokole. U praksi se danas najčešće koristi HTTP kao prijenosni protokol.



Slika 3: Primjer SOAP zahtjeva i odgovora preko HTTP protokola

SOAP protokol definira omotnicu u koju se umeće poruka te definira pravila za zapisivanje poziva udaljenih procedura. Primjer na slici 3 prikazuje poziv mrežne usluge pomoću SOAP protokola. SOAP zahtjev prenosi se kao sadržaj HTTP POST zahtjeva. Zahtjev je poslan poslužitelju *www.e-burza.hr* koji pokreće metodu *DohvatiCijenuDionice* koristeći znakovni niz „ERNT“ kao vrijednost parametra *NazivDionice*. SOAP odgovor nosi rezultat koji je realni broj vrijednosti *1205*. Slično kao i zahtjev, SOAP odgovor umetnut je u HTTP odgovor. Tijelo SOAP zahtjeva sadrži opis poziva metode. Opis poziva metode sastoji se od naziva metode i vrijednosti parametara. U tijelu SOAP odgovora sadržan je rezultat izvođenja metode. Podatkovni tipovi parametara ne definiraju se u SOAP zahtjevu i odgovoru, nego u WSDL dokumentu pridruženom pripadnoj usluzi. Svaka SOAP omotnica nužno sadrži tijelo, no može sadržavati i zaglavje, koje je vrsta upravljačkog kanala. Zaglavljem se prenose dodatne upute i informacije koje primatelj koristi za tumačenje tijela.

WSDL

Pri oblikovanju mrežnih usluga SOAP protokol koristi se za zapisivanje poruka, no SOAP ne definira koje poruke je moguće razmjenjivati s određenom uslugom. Za definiranje poruka, odnosno opisivanje programskog sučelja mrežne usluge koristi se jezik WSDL. Kao i SAOP, WSDL je zasnovan na XML jeziku.

Opis sučelja uključuje definiciju podatkovnih tipova, oblik ulaznih i izlaznih poruka, metode koje pruža usluga, mrežne adrese te protokole kojima je ostvarena usluga. Struktura WSDL dokumenta dijeli se na dva dijela: sučelje usluge i ostvarenje usluge [36]. Na taj način omogućeno je definiranje dijelova odvojeno i neovisno jedan o drugome. Osim toga, sučelje usluge moguće je ostvariti na više načina a dijelove WSDL dokumenta moguće je ponovo iskoristiti u drugim WSDL dokumentima. Na slici 4 dana je struktura WSDL dokumenta koji opisuje sučelje mrežne usluge na adresi <http://www.e-buzra.hr/ponuda/Ponuda.asmx>. Za istu mrežnu uslugu na prethodnoj slici 3 prikazane su SOAP poruke.

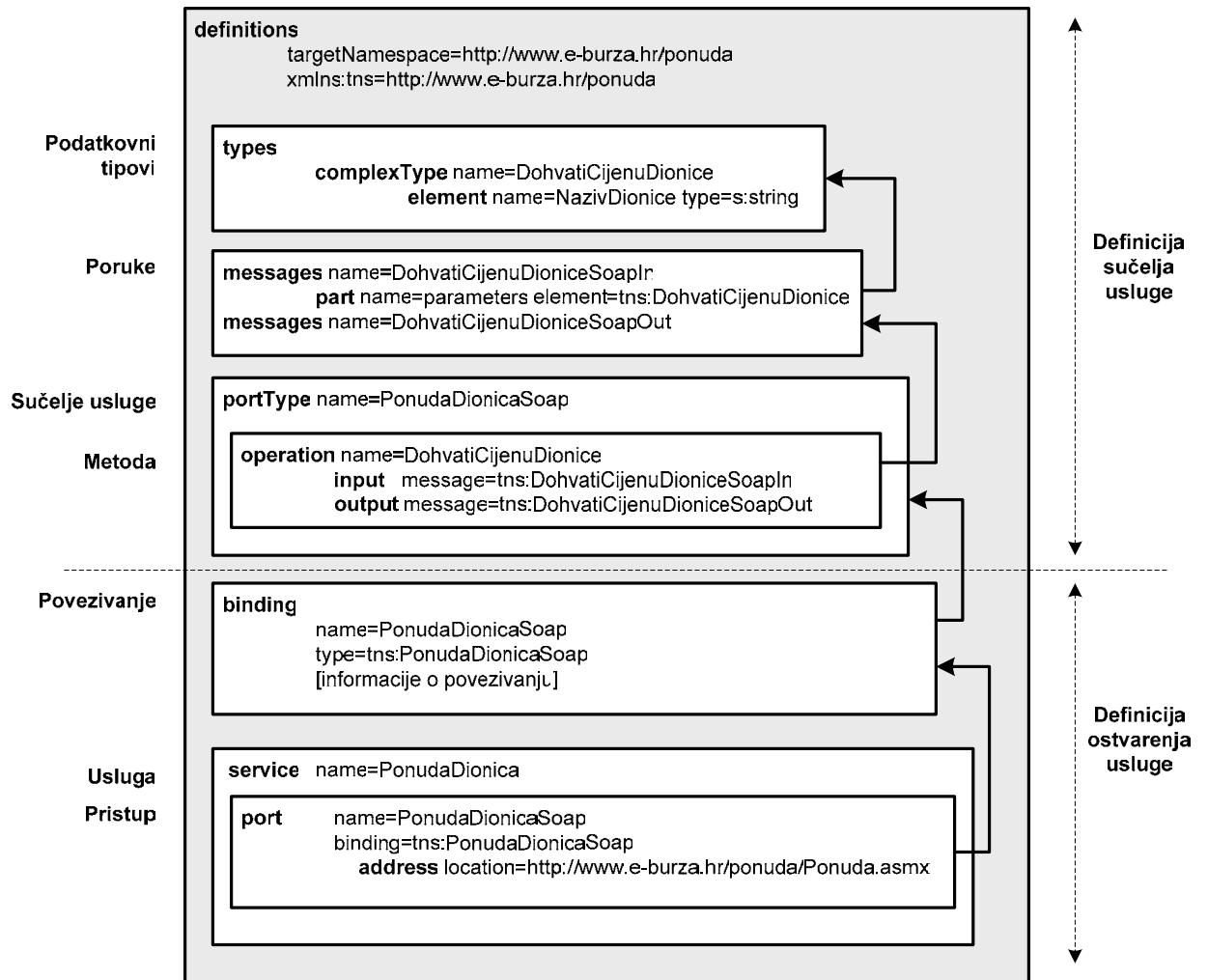
Definicija sučelja usluge sastoji se od sljedećih elemenata:

- **Operacije** (element *Operation*): odgovaraju metodama u proceduralnom programskom jeziku, odnosno predstavljaju atome funkcionalnosti. U danom primjeru naziv jedine operacije je *DohvatiCijenuDionice*.
- **Poruka** (element *Message*): skup podataka definiranih tipova koji se razmjenjuje prilikom poziva operacija usluge. WSDL omogućava tri vrste poruka: ulazne, izlazne i ulazno/izlazne. *DohvatiCijenuDioniceSoapIn* je ulazna poruka operacije *DohvatiCijenuDionice* u navedenom primjeru.
- **Podatkovni tip** (element *Type*): podatkovni tipovi definirani su pomoću *XML Schema* dokumenta koji je umetnut u WSDL dokument. Definirani podatkovni tipovi čine rječnik kojim se oblikuju poruke. Podatkovni tip *NazivDionice* u primjeru je znakovni niz koji koristi ulazna poruka *DohvatiCijenuDioniceSoapIn* operacije *DohvatiCijenuDionice*.
- **Sučelje** (element *Port Type*): definira sučelje usluge, odnosno skup operacija, slično sučeljima u jezicima C# i Java koja definiraju skup metoda koje objekt javno izlaže. U primjeru sučelje *PonudaDionicaSoap* sadrži jedinu operaciju *DohvatiCijenuDionice*.

Definicija ostvarenja usluge sastoji se od sljedećih elemenata:

- **Povezivanje** (element *Binding*): određuje prijenosni protokol pomoću kojeg je ostvarena usluga, primjerice HTTP ili SMTP. Također, određuje oblik zapisivanja podataka koje koristi određeno sučelje.

- **Pristup usluzi** (element *Port*): krajnja točka na kojoj je smješteno određeno ostvarenje sučelja usluge, a definirana je mrežnom adresom i povezivanjem. Pristup usluzi *PonudaDionicaSoap* ostvaren je pomoću povezivanja *PonudaDionicaSoap* na adresi <http://www.e-burza.hr/ponuda/Ponuda.asmx>.
- **Usluga** (element *Service*): sastoji se od skupa pristupa usluzi



Slika 4: Struktura WSDL dokumenta

UDDI

UDDI specifikacija omogućava korisnicima objedinjen i sustavan način pronalaska pružatelja usluga putem centraliziranog registra usluga. UDDI registar pruža funkcionalnost automatiziranog *on-line imenika* mrežnih usluga. UDDI specifikacija sastoji se dvije osnovne specifikacije koje definiraju:

- Informacije o pojedinoj usluzi u registru te način zapisivanja tih informacija
- Programsko sučelje za postavljanje upita i ažuriranje informacija u registru

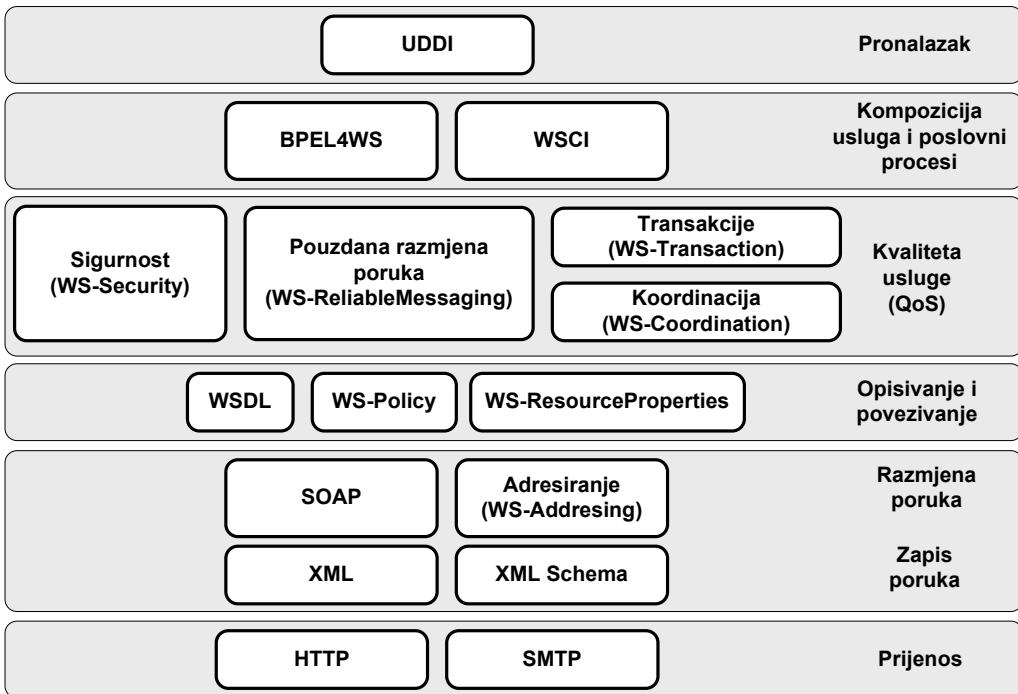
Pristup registru ostvaren je putem posebne mrežne usluge koja je zasnovana na standardnom SOAP protokolu.

UDDI registar zapisuje tri vrste informacija o mrežnim uslugama:

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

2.2.2. Prošireni skup standarda WS-*

Osnovna arhitektura mrežnih usluga definira raspodijeljeni model izračunavanja u kojem usluge međudjeluju razmjenom XML dokumenata te definira sintaksu za razmjenu informacija. Na taj način omogućuje se opisivanje, objava i međudjelovanje usluga na standardan način. Međutim, mnogi raspodijeljeni programski sustavi zahtijevaju razna složenija ponašanja usluga. Primjerice, zahtijevaju se sljedeći mehanizmi: provjera vjerodostojnosti (engl. *authentication*), osiguranje kvalitete usluge pomoću pouzdane i sigurne razmjene poruka te putem mogućnosti provođenja transakcija. Također, potrebni su mehanizmi za koordinaciju, kompoziciju i orkestraciju usluga. Stoga se u posljednje vrijeme ulažu napor u razvoj specifikacija više razine kojima je cilj omogućiti ostvarivanje složenog ponašanja usluga na definiran i standardan način. Pri tome, specifikacije nisu usko vezane uz neko posebno područje primjene kako bi ih bilo moguće koristiti za široki skup primjena. Razvijene su brojne specifikacije koje pokrivaju navedena područja, a najznačajnije su [42]: specifikacija *jezika za opisivanje izvođenja poslovnih procesa* (engl. *Business process execution Language for Web services, BPEL4WS*) koji se koristi za kompoziciju usluga. Nadalje, razvijena je specifikacija za koordinaciju usluga (*WS-Coordination*) te specifikacija koja definira transakcije, odnosno čvrsto međudjelovanje više usluga (*WS-Transaction*). Razvijene su i specifikacije za sigurni zajednički rad mrežnih usluga (*WS-Security*) te za pouzdanu razmjenu poruka (*WS-ReliableMessaging*).



Slika 5: Prošireni stog WS-* standarda

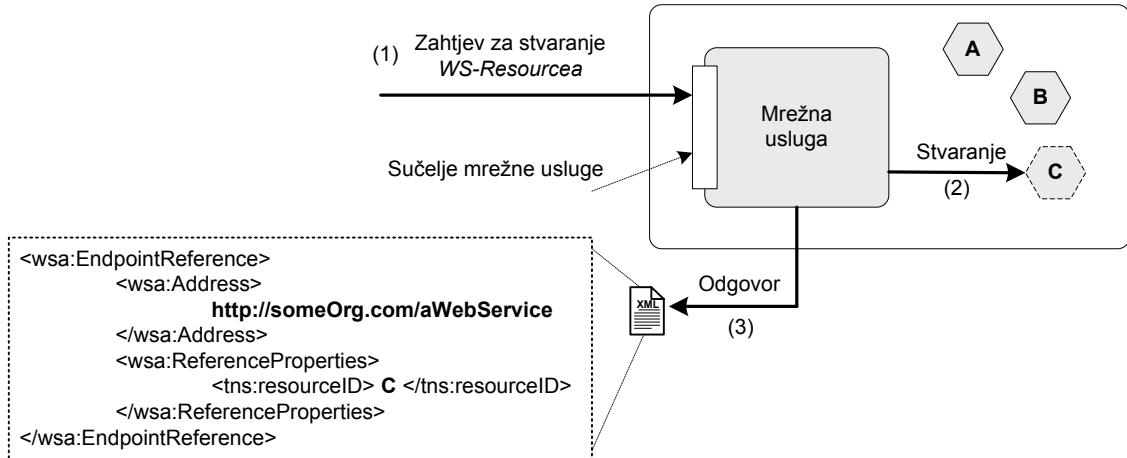
Svi navedeni aspekti ključni su elementi za značajnije poslovno međudjelovanje. Nadalje, razvijen je radni okvir *WS-Policy* koji definira skup pravila za opisivanje informacija o kvaliteti, mogućnostima, zahtjevima i svojstvima mrežne usluge. Dodatne informacije zapisuju se u WSDL opis usluge, a uvedenim pravilima proširuju se opisne mogućnosti WSDL jezika.

Stog mehanizama i protokola proširene arhitekture mrežnih usluga oblikovan je modularno, odnosno moguće je koristi zasebne dijelove stoga ovisno o potrebi i primjeni. Prošireni stog WS-* standarda prikazan je na slici 5.

WS-Resource Framework

Mrežne usluge često pružaju korisniku mogućnost pristupa i upravljanja stanjem neke komponente ili podatkovnih vrijednosti koje postoje i mijenjaju se za vrijeme međudjelovanja. Dakle, poruke koje se razmjenjuju putem mrežne usluge namijenjene su pristupu nekom sredstvu sa stanjem. Međutim, sučelja mrežnih usluga ne pružaju eksplicitan i standardan pristup sredstvima sa stanjem. Korisnik je primoran samostalno zaključiti o postojanju sredstva sa stanjem u pozadini mrežne usluge. Iz tog razloga razvijen je radni okvir *WS-Resource Framework* [43] koji sadrži skup specifikacija koje definiraju standardne obrasce za pronalazak, ispitivanje i međudjelovanje sa sredstvom sa stanjem. Radni okvir *WS-Resource Framework* definira pojam *WS-Resourcea* te osnovne postupke za stvaranje, pristup i uništavanje *WS-Resourcea*. Pojedine specifikacije definiraju dodatne funkcionalnosti vezane uz *WS-Resource*: adresiranje (*WS-Addressing*), upravljanje vremenom života sredstva (*WS-*

ResourceLifetime), upravljanje svojstvima sredstva (*WS-ResourceProperties*), grupiranje više *WS-Resourcea* (*WS-ServiceGroup*) i mehanizam prijavljivanja pogrešaka (*WS-BaseFaults*).



Slika 6: Primjer stvaranja *WS-Resourcea*

WS-Resource je komponenta koja je kombinacija mrežne usluge i sredstva sa stanjem. Za upravljanje stanjem sredstva brine se neka druga komponenta računalnog sustava poput datotečnog sustava, baze podataka ili procesa operacijskog sustava. Mrežna usluga pruža pristup sredstvu sa stanjem pa je uvedena formalna specifikacija odnosa između mrežne usluge i sredstva sa stanjem koja se naziva *obrazac impliciranog sredstva* (engl. *implied resource pattern*) [44]. Obrazac impliciranog sredstva zahtijeva način adresiranja *WS-Resourcea* prema specifikaciji *WS-Addressing*. Sukladno tome, svaki *WS-Resource* jednoznačno je određen referencom krajne točke (engl. *endpoint reference*). Referenca krajne točke sastoji se od fizičke adrese mrežne usluge ovisne o protokolu te skupa svojstava reference (engl. *reference properties*) koji se naziva kontekst. Kontekst jednoznačno definira primjerak *WS-Resourcea* u skupu primjeraka vezanih uz određenu mrežnu uslugu. Referenca krajne točke *WS-Resourcea* slična je pojmu memorijske kazaljke objekta u objektno orijentiranom jeziku.

Definicija vrste *WS-Resourcea* zadaje se putem opisa njegovog sučelja koji se umeće u WSDL dokument mrežne usluge. Također, definira se XML dokument sa skupom svojstava sredstva koji predstavlja ukupno stanje sredstva. Dokument sa svojstvima moguće je čitati i mijenjati koristeći operacije koje su definirane vrstom *WS-Resourcea*, odnosno sučeljem usluge.

U primjeru na slici 6 korisnik šalje zahtjev za stvaranje novog *WS-Resourcea*. Zahtjev se obrađuje i stvara se sredstvo sa stanjem kojem je identifikator „C“. Odgovor mrežne usluge sadrži referencu krajne točke stvorenog *WS-Resourcea*. U ovom primjeru jedino svojstvo reference krajne točke je identifikator sredstva označen s *resourceID*.

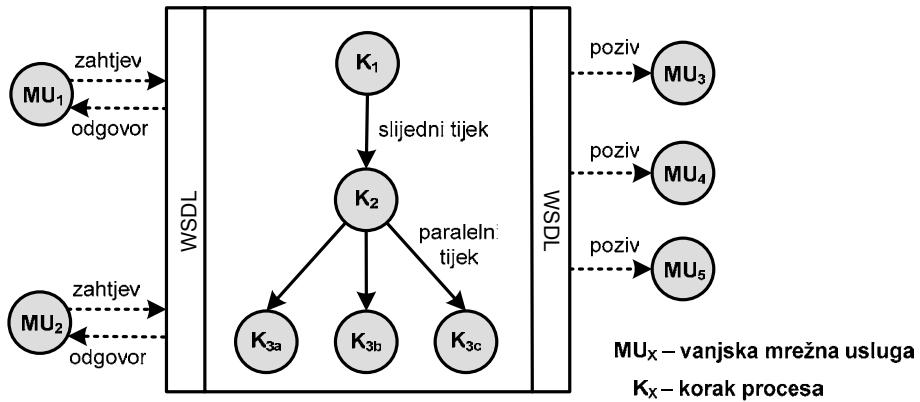
2.3. Kompozicija usluga

S obzirom da su usluge u računarstvu zasnovanom na uslugama osnovni blokovi za izgradnju programskih sustava, osnovni mehanizam pri razvoju je kompozicija usluga [42]. Mehanizmom kompozicije, više usluga se grupira prema određenom obrascu u svrhe postizanja poslovnih ciljeva, rješavanja znanstvenih problema ili da bi se izgradila nova složenija usluga. Kako su kompozicije usluga također usluge, mehanizam kompozicije usluga moguće je iterativno ponavljati po potrebi. Nadalje, mehanizam kompozicije usluga pruža način integracije programskih sustava. Moguće je bez teškoća integrirati programske sustave unutar jednog poslovnog okruženja, no također je moguća integracija programskih sustava više poslovnih subjekata.

2.3.1. Kompozicija usluga zasnovana na procesima

Specifikacija jezika BPEL [46] služi za modeliranje ponašanja mrežnih usluga koje međudjeluju unutar poslovnog procesa. BPEL jezik pruža gramatiku zasnovanu na XML-u za opisivanje upravljačke logike za koordiniranje mrežnih usluga koje međudjeluju u tijeku procesa. Programska komponenta za orkestraciju izvodi BPEL proces koordiniranjem aktivnosti koje su opisane BPEL jezikom. BPEL proces također čini određenu mrežnu uslugu čije je sučelje opisano WSDL dokumentom. WSDL sučelje definira poduprte operacije dok je BPEL jezikom definiran način na koji se one izvode. Nadalje, WSDL opisuje ulazne i izlazne točke BPEL procesa, te podatkovne tipove koji se razmjenjuju pri komunikaciji sa BPEL procesom. Dodatno, WSDL referencira vanjske usluge koje BPEL proces koristi.

BPEL omogućava oblikovanje *izvodljivih procesa i apstraktnih poslovnih procesa*. Izvodljivi proces modelira ponašanje sudionika u određenom poslovnom međudjelovanju, odnosno modelira se izvođenje radnog tijeka procesa (engl. *workflow execution*). Apstraktni proces ili poslovni protokol određuje javnu razmjenu poruka među sudionicima. Poslovni protokoli nisu izvodljivi i ne sadrže interne informacije o tijeku pojedinog procesa. Pojam izvođenja poslovnog procesa također se naziva i *orkestracija* usluga jer se opisuje ponašanje usluga unutar jednog procesa. Rad usluga orkestira se s jednog mesta i usluge zajedno čine izvodljivi proces. Razmjena javnih poruka među procesima još se naziva *koreografija*. Definira se udio svakog sudionika u međudjelovanju iz njegova perspektive, odnosno svi sudionici su promatrani ravnopravno. S druge strane, pri orkestraciji jedan centralni proces definira upravljanje ponašanjem ostalih usluga [45]. BPEL jezikom definiraju se dvije vrste aktivnosti: *osnovne aktivnosti* i *strukturirane aktivnosti*. Osnovna aktivnost je naredba koja definira međudjelovanje s vanjskom mrežnom uslugom, odnosno upravlja primanjem zahtjeva od drugih mrežnih usluga, odgovaranjem mrežnim uslugama i asinkronim pozivanje drugih mrežnih usluga.



Slika 7: Primjer tijeka procesa u BPEL-u

U tipičnom scenariju ponašanja, BPEL izvodljivi proces prima zahtjev od neke mrežne usluge, poziva nekoliko vanjskih mrežnih usluga kako bi dohvatio određene podatke, te odgovara pozivajućoj mrežnoj usluzi. Na slici 7 poruke *poziv*, *zahtjev* i *odgovor* predstavljaju osnovne aktivnosti procesa. *Strukturirane aktivnosti* predstavljaju programsku logiku BPEL procesa, odnosno upravljaju cjelokupnim tijekom procesa, određujući slijed osnovnih aktivnosti koje međudjeluju s vanjskim mrežnim uslugama. Strukturirane aktivnosti omogućuju petlje s uvjetima te dinamičko grananje tijeka procesa. U primjeru na slici 7 prikazane su dvije strukturirane aktivnosti: programski konstrukt za definiranje slijednog tijeka izvođenja te programski konstrukt za definiranje paralelnog tijeka izvođenja.

3. Komunikacija zasnovana na modelu objava/preplata

Raspodijeljeni sustavi posjeduju brojne prednosti nad centraliziranim sustavima. Raspodijeljeni sustavi podržavaju veći broj korisnika uz manji trošak a ukupna raspoloživost sustava je veća nego kod centraliziranih sustava. Naime, povećanje mogućnosti sustava sukladno rastu veličine sustava ostvaruje se dodavanjem malih, jeftinih komponenti. Međutim, osim brojnih prednosti postoje nedostaci. Značajan nedostatak je što se rastom veličine sustava povećava i složenost sustava kojom je potrebno ovladati. Složenost sustava proizlazi iz velikog skupa autonomnih i raznorodnih komponenti koje sačinjavaju raspodijeljeni sustav. Da bi se riješio problem složenosti i raznorodnosti raspodijeljenog sustava, razvijeni su posrednički sustavi koji prikrivaju različitosti raznorodnih računalnih platformi i pružaju homogen i apstraktan izgled cjelokupnog raspodijeljenog sustava. Većina današnjih komunikacijskih posredničkih sustava, poput CORBA i Java RMI, zasnovani su na pozivanju udaljenih metoda prema zahtjev/odgovor komunikacijskom obrascu. Korisnik upotrebljava određenu uslugu na udaljenom poslužitelju slanjem zahtjeva ili pozivanjem udaljene metode. Zahtjev/odgovor obrazac komunikacije zadovoljava kvalitetom rada u okolini lokalnih mreža s umjerenim brojem korisnika i poslužitelja. Međutim, sustavi zasnovani na zahtjev/odgovor obrascu nemaju svojstvo razmernog rasta i nisu primjenjivi na mreže širokih razmjera kao što je Internet. Osnovni problem je da zahtjev/odgovor komunikacijski obrazac pruža komunikaciju *jedan-na-jedan* u kojoj jedan korisnik komunicira s jednim poslužiteljem. No za sustave velikih razmjera pogodniji je način komunikacije *više-na-više* u kojem korisnik ne mora odabrat komunikacijskog partnera. Dodatno, komunikacija prema zahtjev/odgovor obrascu je čvrsto povezana jer je poziv udaljenih metoda sinkron, a sudionici moraju biti aktivni u isto vrijeme. Navedena ponašanja sudionika nisu poželjna u sustavima razmjera mreže Internet zbog velikog broja potencijalnih sudionika komunikacije te svojstva dinamičnosti sustava uzrokovanih jednostavnim spajanjem i odspajanjem sudionika iz sustava.

Zbog navedenih nedostataka posredničkih sustava zasnovanih na obrascu zahtjev/odgovor, u posljednje vrijeme razvijeni su razni posrednički sustavi više razine koji koriste postojeće posredničke sustave. Jedan od najčešćih je posrednički sustav zasnovan na događajima, odnosno na obrascu objava/preplata [16]. U obrascu objava/preplata, događaji su osnovni komunikacijski mehanizam, a postoje dvije vrste sudionika. Prva vrsta sudionika, preplatnici na događaje, izražavaju interes za određene kategorije događaja izdavanjem pretplata. Druga vrsta su objavitelji događaja koji objavljuju događaje a koji se kasnije dostavljaju svim zainteresiranim korisnicima. Prednost posredničkih sustava

zasnovanih na obrascu objava/preplata je slaba povezanost sudionika te komunikacija između preplatnika i objavitelja primjenom modela *više-na-više*. Preplatniku nije bitno koji objavitelj objavljuje događaje za koje je preplatnik zainteresiran. Također, objavitelju nije potrebno poznavanje skupa preplatnika koji dobiva događaje koje objavitelj objavljuje.

3.1. Razredba modela međudjelovanja procesa u raspodijeljenim sustavima

Raspodijeljeni sustavi sastoje se od programskih komponenti koje se izvode na različitim računalima u lokalnoj ili rasprostranenoj mreži. Nadalje, raspodijeljeni programski sustav čine procesi koji se paralelno izvode na virtualnim procesorima različitih platformi te različitim operativnim sustavima u raspodijeljenom sustavu. Kako bi se korisniku raspodijeljeni sustav doimao kao jedinstven sustav te kako bi se ostvarili ciljevi raspodijeljenih programskih sustava potrebno je ostvariti učinkovitu komunikaciju i koordinaciju među procesima. Postoje mnogi modeli programskih infrastruktura i posredničkih sustava korištenih pri ostvarenju raspodijeljenih sustava, a najznačajniji su: priključnice, pozivi udaljenih procedura, pozivi nad udaljenim objektima, razmjena poruka putem repova (engl. *message-queuing*), dijeljeni podatkovni prostori te objava/preplata model. Navedeni modeli omogućuju razmjenu podataka među raspodijeljenim procesima te nude razvijatelju programskog sustava usluge mnogo više razine nego što to nude usluge razmjene poruka prijenosnog sloja mreže. Nadalje, modeli se razlikuju prema raznim svojstvima koja proizlaze iz načina komunikacije i koordinacije procesa u modelima. Pri tome, funkcionalnosti koje pružaju modeli na različitim su razinama apstrakcije.

Primjerice, priključnice su sučelje prema uslugama prijenosa podataka koje pruža prijenosni sloj mreže, a ostvarene su razmjenom poruka prema protokolima TCP i UDP. Posebnost objava/preplata modela međudjelovanja među procesima je njegova viša razina apstrakcije. Zbog toga, objava/preplata model moguće je ostvariti pomoću modela niže razine kao što su priključnice ili pozivi nad udaljenim objektima.

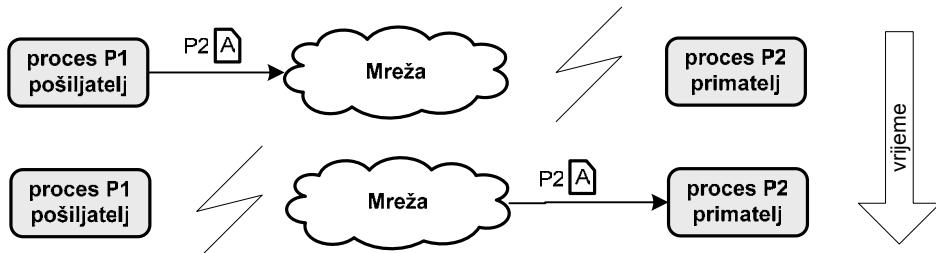
Postoje mnoge razredbe modela međudjelovanja procesa u raspodijeljenim sustavima [13, 18]. Svaka od razredbi provedena je s obzirom na odabrani skup svojstva. Pri razmatranju posredničkih sustava zasnovanih na porukama u radu [5] koriste se dva jednostavna modela međudjelovanja: sinkrona i asinkrona komunikacija. No, postoje i neke složenije razredbe modela međudjelovanja poput razredbe prema mehanizmu dostave podataka s obzirom na tri svojstva [22]. Prvo svojstvo je način pokretanja komunikacije koji može biti gurni ili povuci način (engl. *push – pull*). Drugo svojstvo je periodičnost komunikacije. Ako je unaprijed definiran vremenski raspored onda je komunikacija periodična, a aperiodična ako je potaknuta nastankom slučajnog događaja u sustavu. Treće svojstvo je broj

primatelja pri komunikaciji, odnosno da li je komunikacija *od točke do točke* ili je *višeodredišna* (engl. *point-to-point, multi-point*). Složena razredba posredničkih sustava zasnovanih na razmjeni poruka prema velikom broju svojstava načinjena je u radu [10]. Svojstva su grupirana u tri modela: model dostave poruke (engl. *message delivery model*), model obrade poruke (engl. *message processing model*), te model neuspjeha komunikacije (engl. *message failure model*). Također, razvijen je složeni radni okvir za razredbu posredničkih sustava zasnovanih na objava/preplata modelu međudjelovanja [15]. Razredba se provodi kroz sedam dimenzija: model objekata, model događaja, model imenovanja, model opažanja, model vremena, model dojave te model sredstava (engl. *object, event, naming, observation, time, notification, resource*).

3.1.1. Razredba prema modelu koordinacije

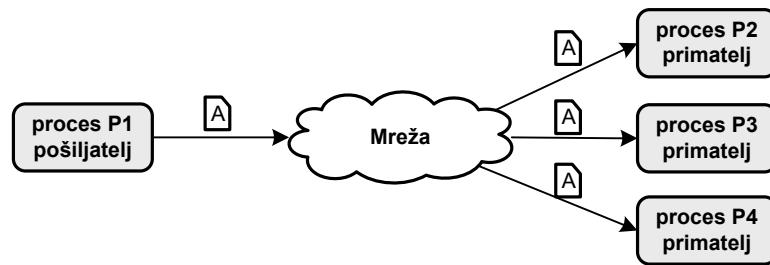
Jedna od najčešćih razredbi modela međudjelovanja među procesima zasniva se na modelima koordinacije među procesima [2]. U raspodijeljenim sustavima zasnovanim na koordinaciji ključno je razdvajanje specifičnih zadataka programskog sustava od same koordinacije među procesima. U takvom raspodijeljenom sustavu, dio zadužen za koordinaciju je onaj koji omogućuje komunikaciju i suradnju među procesima. Modeli koordinacije razmatraju se s obzirom na dva svojstva međudjelovanja procesa: svojstvo vremenske povezanosti te svojstvo referencijske povezanosti.

Raspodijeljeni procesi su **vremenski povezani** ako se moraju izvoditi istovremeno, odnosno ako moraju biti aktivni u isto vrijeme kako bi razmjenjivali podatke. Primjer vremenski nepovezanih procesa prikazan je na slici 8.



Slika 8: Vremenski nepovezani procesi

Raspodijeljeni procesi su **referencijski povezani** ako proces koji pokreće komunikaciju mora poznavati jedinstvenu adresu ili identifikator procesa primatelja. Primjer referencijski nepovezanih procesa prikazan je na slici 9.



Slika 9: Referencijski nepovezani procesi

S obzirom na navedena svojstva postoje četiri modela koordinacije: izravna koordinacija (engl. *direct coordination*), koordinacija putem spremnika poruka (engl. *mailbox coordination*), koordinacija putem sastanaka (engl. *meeting-oriented coordination*) i generativna komunikacija (engl. *generative communication*). Razredba modela koordinacije prikazana je u tablici 1.

Ako su procesi vremenski i referencijski povezani onda je **koordinacija izravna**. Procesi moraju biti aktivni da bi razmjenjivali podatke, a proces pošiljatelj mora poznati adresu procesa primatelja. Jedna od primjena modela izravne koordinacije je u ostvarenju nepostojane komunikacije zasnovane na porukama (engl. *transient message-oriented communication*). Ako su procesi vremenski nepovezani, a referencijski povezani onda se **koordiniraju putem spremnika poruka**. Procesi se ne moraju izvoditi istovremeno jer proces pošiljatelj stavlja poruku u spremnik poruka gdje ona ostaje sve dok ju primatelj ne dohvati. No procesi su referencijski povezani jer pošiljatelj mora poznati adresu spremnika primatelja putem kojeg oni razmjenjuju poruke. Procesi se **koordiniraju putem sastanaka** kada su vremenski povezani, a referencijski nepovezani. Procesi ne poznaju izričito adrese drugih procesa s kojima komuniciraju te da bi koordinirali svoje aktivnosti oni koriste mehanizam koji im omogućuje privremeno grupiranje u vremenu, tj. sastanak. Jedan od mehanizama koji ostvaruje ovu vrstu koordinacije je objava/preplata sustav. Procesi putem sustava objavljuju poruke (događaje) raznih kategorija. Nadalje, procesi se pretplaćuju na određene kategorije poruka. Prilikom objave poruke iz kategorije na koju je proces pretplaćen poruka se dostavlja procesu putem objava/preplata sustava. Procesi su vremenski povezani jer proces koji objavljuje poruku te onaj koji ju prima moraju biti aktivni u isto vrijeme. Dakle, procesi se *sastaju* objavljivanjem događaja, te se stoga koordinacija putem sastanka naziva još i **koordinacija putem događaja** (engl. *event-driven coordination*).

	vremenski povezani	vremenski nepovezani
referencijski povezani	izravna koordinacija	koordinacija putem spremnika poruka
referencijski nepovezani	koordinacija pomoću sastanaka	generativna komunikacija

Tablica 1: Razredba modela koordinacije

Model koordinacije u kojem su procesi vremenski i referencijski nepovezani je **generativna komunikacija** ili **koordinacija zasnovana na podacima**. Prvi sustav kojim je ostvaren takav model koordinacije je Linda. Model koordinacije zasnovan na podacima detaljnije je objašnjen u pododjeljku 3.2.3.

3.1.2. Razredba prema obrascima komunikacije i suradnje raspodijeljenih procesa

Koordinacija procesa u raspodijeljenim sustavima podrazumijeva njihovu komunikaciju i suradnju. Komunikacija omogućava prijenos informacija između procesa koji međudjeluju. Također, pri komunikaciji definira se skup pravila za stvaranje i slijed izmjena poruka. Skup takvih pravila čini *obrazac komuniciranja* raspodijeljenih procesa. Slično, obrazac suradnje sačinjavaju pravila koja definiraju zajedničko djelovanje raspodijeljenih procesa promatranih kao jedna cjelina. Stoga se modeli međudjelovanja mogu podijeliti s obzirom na obrasce komunikacije i suradnje među procesima [13].

3.1.2.1 Obrasci komunikacije

Razredba obrazaca komunikacije može se načiniti prema sljedeća tri obrasca: zahtjev/odgovor (engl. *request/reply*), stavi/dohvati (engl. *put/get*) i preplata/objava/dojava (engl. *Subscribe/publish/notify*) [13]. Navedeni obrasci komunikacije razlikuju se po svojstvima vremenske i referencijske povezanosti, postojanosti komunikacije (engl. *communication persistency*), pokretanju komunikacije i višestrukosti primatelja (engl. *receiver multiplicity*).

- **Postojanost komunikacije:** Postojana komunikacija (engl. *persistent*) osigurava da se predana informacija dostavi primatelju, a prolazna komunikacija (engl. *transient*) ne jamči isporuku informacije, odnosno ne pruža potpunu pouzdanost isporuke.
- **Pokretanje komunikacije:** Moguća su dva načina pokretanja komunikacije: povuci način (engl. *pull*) i gurni način (engl. *push*). Svojstvo pokretanja komunikacije određuje da li isporuku poruke započinje korisnik ili poslužitelj. Dva procesa komuniciraju povuci načinom kada proces korisnik šalje izričit zahtjev za isporuku podatka. Nakon što je zahtjev primljen, slijedi odgovor procesa poslužitelja. S druge strane, ako procesi komuniciraju na gurni način onda je proces korisnik pasivan te čeka na nove podatke od poslužitelja. Kako bi proces poslužitelj mogao dojaviti nastanak novih podataka, proces korisnik mora ostvariti *rukovatelja dogadjaja* (engl. *event handler* – procedura za obradu dogadaja) te sučelje za dojavu pomoću kojeg mu poslužitelj dostavlja novonastale podatke.

- **Višestrukost primatelja:** Ako prilikom slanja podataka postoji samo jedan odredišni proces onda je komunikacija *od točke do točke*, a ako postoji više odredišnih procesa koji primaju poslani podatak onda je komunikacija *višeodredišna* (engl. *multi-point*).

Dodatno, česta svojstva obrazaca komunikacije su svojstvo posrednika u komunikaciji i svojstvo sinkronosti [10].

- **Posrednici u komunikaciji:** Određuje da li postoje posrednici koji sudjeluju u dostavi poruke od izvorišnog do odredišnog procesa. Posrednici mogu biti repovi poruka ili posebni objekti. Posrednik može biti samo jedan ili više njih. Posrednici se koriste kada su poruke oblikovane kao posebni entiteti u sustavu i ključni su za ostvarenje modela razmjene poruka primjenom sinkronog pozivanja objekata.
- **Sinkronost:** Svojstvo koje određuje da li je komunikacija sinkrona ili asinkrona. Pri sinkronoj komunikaciji proces pošiljatelj je nakon slanja poruke ili poziva udaljenog objekta blokiran sve dok ne dobije odgovor od primatelja. Asinkrona komunikacija omogućuje da proces nastavi s radom, a odgovor mu se dojavljuje pomoću posebnih mehanizama.

Obrazac zahtjev/odgovor

Obrazac zahtjev/odgovor jednostavan je i često upotrebljavan komunikacijski obrazac zasnovan na razmjeni zahtjeva koje slijede odgovori. Proces korisnik izdaje zahtjev te ga dostavlja procesu poslužitelju na obradu. Nakon obrade, poslužitelj šalje odgovor korisniku. Korisnik je blokiran dok čeka na odgovor ili potvrdu o primitku zahtjeva od poslužitelja. Proces korisnik i proces poslužitelj moraju se izvoditi istovremeno kako bi se provela komunikacija pa su oni *vremenski povezani*. Također, oni su *referencijski povezani* jer proces korisnik šalje zahtjev poznatom poslužitelju, a zahtjev sadrži identifikator procesa koji ga poslužuje. Obrazac zahtjev/odgovor je tipičan predstavnik *povuci načina* komunikacije koja se odvija *od točke do točke*. Komunikacija se pokreće izričitim zahtjevom za podatcima koji izdaje korisnik. Uporaba obrasca zahtjev/odgovor tradicionalan je pristup pri razvoju komunikacijskog sustava korisnik-poslužitelj [1, 13]. *World Wide Web* (WWW), jedna od najkorištenijih usluga u suvremenim mrežama, upotrebljava zahtjev/odgovor obrazac međudjelovanja kao osnovni komunikacijski model.

Obrazac stavi/dohvati

U obrascu stavi/dohvati, pošiljatelji stavljaju podatke u dobro poznati spremnik podataka (engl. *well-known storage space*). S druge strane, primatelji dohvaćaju podatke iz tog spremnika. Pošiljatelji i primatelji su *vremenski nepovezani* jer je postupak stavljanja poruke neovisan o njezinom dohvaćanju. Pošiljatelji i primatelji ne međudjeluju izravno, nego putem posredničkog spremnika podataka pa je *stavi/dohvati* slabo povezan i posrednički komunikacijski obrazac. S obzirom da su podaci u

spremniku postojani sve dok ih ne dohvati primatelj, obrazac stavi/dohvati omogućava postojanu komunikaciju, što je njegova glavna prednost.

Dva često korištena modela komunikacijskih posredničkih sustava ostvarena su prema stavi/dohvati obrascu: *razmjena poruka putem repova* (engl. *message-queuing*) i *dijeljeni podatkovni prostor* (engl. *shared dataspace*).

Obrazac preplata/objava/dojava

Preplata/objava/dojava je *višeodredišni* komunikacijski obrazac zasnovan na *gurni* načelu. Primatelji podataka *preplaćuju* se na određeni razred podataka time što ostvaruju rukovatelje (engl. *handler*) koji obrađuju podatke objavljene od strane pošiljatelja podataka. U trenutku kada objavitelj *objavi* neki podatak, operacija *dojavi* poziva se na strani preplatnika ako objavljeni podatak spada u razred podataka na koji je preplatnik preplaćen. U ovom slučaju preplatnik i objavitelj su vremenski i referencijski povezani jer međudjeluju izravno. No, većina objava/preplata sustava koji ostvaruju objava/preplata/dojava obrazac pružaju rješenja koja koriste posrednika kojim se postiže referencijski i vremenski nepovezana komunikacija među korisnicima. S druge strane, model koordinacije putem sastanka podrazumijeva vremenski povezane procese. Razlika u svojstvu vremenske povezanosti posljedica je činjenice da postoje ostvarenja objava/preplata/dojava obrasca s vremenski povezanim i vremenski nepovezanim procesima.

3.1.2.2 Obrasci suradnje

Individualni procesi nisu primjenjivi i dostatni za rješavanje složenih problema većih razmjera. Iz tog razloga više raspoloživih procesa se povezuje te se njihovim zajedničkim djelovanjem, odnosno njihovom suradnjom, pruža složenija usluga. Modeli međudjelovanja među raspoloživim procesima dijele se na one koji ostvaruju suradnju *zasnovanu na adresama* te one koji ostvaruju *suradnju zasnovanu na sadržaju*.

Suradnja zasnovana na adresama

Tradicionalni pristup suradnji oslanja se na raspoloživost adrese sudionika s kojim se komunicira, zbog čega su sudionici referencijski povezani, a njihova suradnja je zasnovana na adresama. Sudionici mogu komunicirati i suradivati s ostalim sudionicima jedino ako su im poznate njihove adrese. Suradnja zasnovana na adresama korištena je pri ostvarenju usmjeravanja te jednostrukog razašiljanja (engl. *unicast*) poruka u mrežnom sloju globalne mreže Internet. Poruke sadrže izričite odredišne adrese čime je omogućeno usmjeravanje poruke do odredišta. Međudjelovanje po obrascu zahtjev/odgovor između korisnika koji izdaje zahtjev i dobro poznatog poslužitelja primjer je suradnje koja je izravna, od točke do točke te zasnovana na adresama. Grupna komunikacija je također primjer

suradnje zasnovane na adresama. Grupa se definira kao skup procesa koji imaju zajedničko logičko ime, koje predstavlja adresu cijele grupe. Višestruko razašiljanje na razini mrežnog sloja (engl. *IP multicast*) te neka rješenja višestrukog razašiljanja na razini primjenskog sloja praktični su primjeri grupne komunikacije [21]. Razmjena poruka putem repova je daljnji primjer suradnje zasnovane na adresi gdje svaki primatelj ima pridružen rep poruka kojem pristupa pomoću njegove adrese.

Suradnja zasnovana na sadržaju

Suradnja zasnovana na sadržaju potaknuta je sadržajem poruka koje razmjenjuju sudionici komunikacije. Odredišta poruka definiraju kriterije ili predloške sadržaja koji žele primati, te na taj način primaju samo one poruke čiji sadržaj zadovoljava postavljene kriterije. Suradnja zasnovana na sadržaju često se ostvaruje pomoću posrednika koji provjerava da li poruke zadovoljavaju kriterije. Nadalje, suradnja zasnovana na sadržaju omogućava anonimnu komunikaciju budući da pošiljatelji sadržaja ne moraju poznavati primatelje sadržaja, a anonimnost vrijedi i obrnuto pa pošiljatelji mogu biti anonymni primateljima sadržaja. U usporedbi s vezivanjem pomoću fiksnih adresa i imena, suradnja zasnovana na sadržaju prilagodljivija je zato što je vezanje između sudionika dinamičko i ovisno o podatcima. Dijeljeni podatkovni prostor i objava/preplata primjeri su modela komunikacijskih posredničkih sustava koji koriste obrazac suradnje zasnovan na sadržaju.

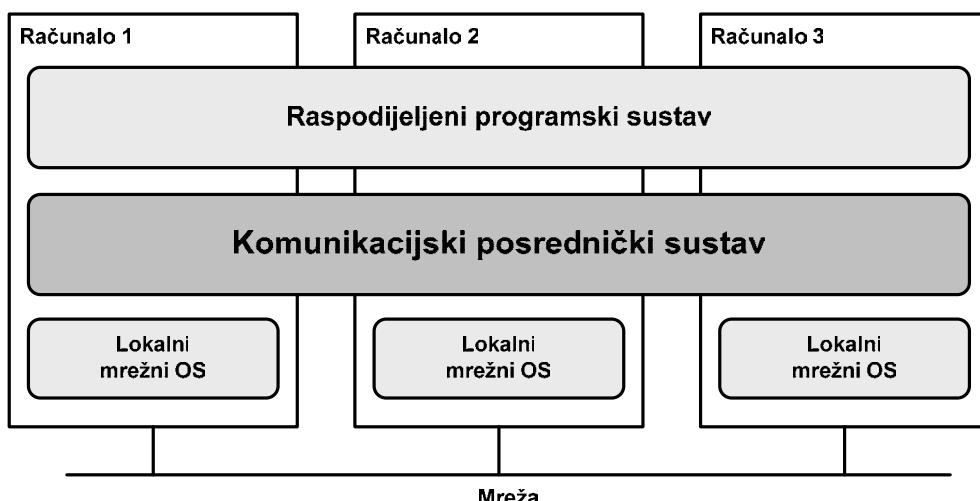
	vremenska povezanost	referencijska povezanost	postojanost	pokretanje komunikacije	višestrukost primatelja
zahtjev/odgovor	povezana	povezana	prolazna	povuci	jedinstveno odredište
stavi/dohvati zasnovan na adresi	nepovezana	povezana	postojana	povuci	jedinstveno odredište
stavi/dohvati zasnovan na sadržaju	nepovezana	nepovezana	postojana	povuci	više odredišta
objava/preplata zasnovana na adresi	povezana	povezana	prolazna	gurni	više odredišta
objava/preplata zasnovana na sadržaju	nepovezana/povezana	nepovezana	postojana	gurni	više odredišta

Tablica 2: Usporedba obrazaca komunikacije i suradnje

Tablica 2 daje usporedbu obrazaca komunikacije i suradnje. Zahtjev/odgovor je čvrsto povezan komunikacijski obrazac sa suradnjom zasnovanom na adresi među procesima koji se izvode istovremeno. Suprotno, stavi/dohvati i objava/preplata/dojava obrasci koriste suradnju zasnovanu na sadržaju te omogućavaju slabo povezano međudjelovanje procesa.

3.2. Modeli komunikacijskih posredničkih sustava za izgradnju raspodijeljenih programskih sustava

Osnovni cilj suvremenih komunikacijskih posredničkih sustava je omogućiti međuplatformsku komunikaciju. Također, komunikacijski posrednički sustavi prikrivaju detalje korištenih komunikacijskih mehanizma niže razine. Iz tog razloga razvijatelji stječu dojam pisanja programskog sustava za jedno jedinstveno računalo. Komunikacijski posrednički sustav dodatni je sloj smješten između raspodijeljenog programskog sustava i mrežnog operativnog sustava [2] kao što je prikazano na slici 10. Za razliku od komunikacijskog posredničkog sustava, posrednici u komunikaciji između sudionika zasebni su entiteti koji omogućuju referencijsku nepovezanost i asinkronu komunikaciju sudionika te postojanost poruka.



Slika 10: Smještaj komunikacijskog posrednika u raspodijeljenom sustavu

Suvremeni modeli komunikacijskih posredničkih sustava su: pozivi udaljenih procedura i pozivi nad udaljenim objektima, posrednički sustavi zasnovani na razmjeni poruka (engl. *message-queuing systems*), dijeljeni podatkovni prostori (engl. *shared dataspaces*) te objava/preplata sustavi (engl. *publish/subscribe*).

Tablica 3 prikazuje razredbu modela komunikacijskih posredničkih sustava prema obrascima komunikacije i suradnje.

	suradnja zasnovana na adresama	suradnja zasnovana na sadržaju
zahtjev/odgovor	pozivi udaljenih procedura	-
stavi/dohvati	sustav za razmjenu poruka	dijeljeni podatkovni prostori
preplata/objava/dojava	grupna komunikacija	objava/preplata

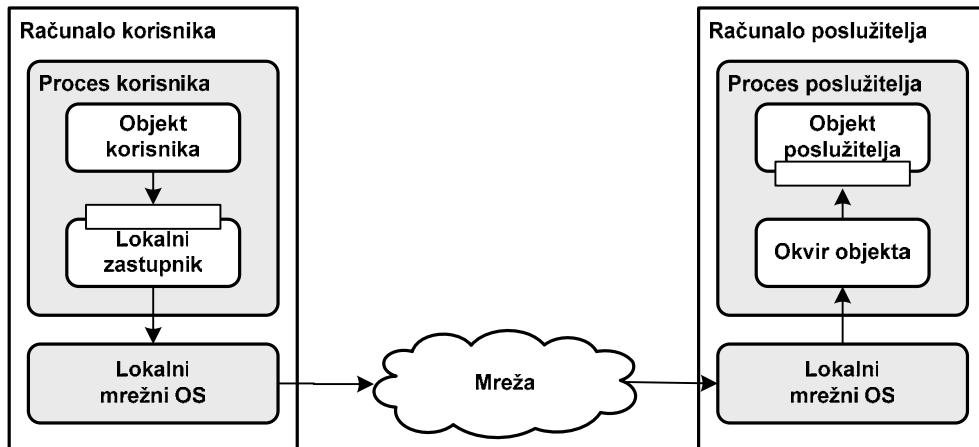
Tablica 3: Razredba modela komunikacijskih posredničkih sustava

3.2.1. Pozivi udaljenih procedura i pozivi nad udaljenim objektima

Pozivi udaljenih procedura (engl. *remote procedure call*, RPC) te pozivi udaljenih objekata (engl. *remote object invocation*, ROI) proširuju načelo pozivanja lokalnih procedura i objekata na okolinu raspodijeljenih sustava. Osnovni cilj uporabe ovog modela je skrivanje fizičkog položaja procesa od razvijatelja programskega sustava. Pozivi udaljenih procedura transparentni su za korisnika čime je olakšano oblikovanje raspodijeljenog programskega sustava. Poziv udaljene procedure sastoji se od mehanizama koji postupkom organizacije na standardan način strukturiraju ime procedure i parametre poziva u korisnički zahtjev na korisničkoj strani, prijenosa korisničkog zahtjeva na poslužiteljsku stranu, te postupka raščlambe parametara i imena procedure iz korisničkog zahtjeva. Rezultat se vraća korisničkom procesu putem istih postupaka, odnosno organizacije, prijenosa i raščlambe.

Model poziva udaljenih objekata zasnovan je na istim načelima, ali je prilagođen objektno-orientiranom programiranju gdje objekti međudjeluju s drugim objektima pozivanjem njihovih metoda putem sučelja koje definira te metode. Slika 11 prikazuje poziv nad udaljenim objektom. Objekt korisnika poziva metodu na udaljenom objektu poslužitelja pomoću lokalnog zastupnika (engl. *client proxy*) koji se nalazi na korisničkoj strani te okvira objekta koji se nalazi na poslužiteljskoj strani (engl. *server skeleton*). Lokalni zastupnik i okvir objekta provode postupak organizacije, prijenosa i raščlambe podataka, prvi s korisničke strane, a drugi s poslužiteljske strane. Lokalni zastupnik udaljenog objekta pruža sučelje jednako pravom sučelju poslužiteljskog objekta. Na taj način ostvareno je skrivanje položaja, jer korisniku poziv udaljene metode izgleda kao lokalni poziv. Pozivanje udaljenih procedura koristi zahtjev/odgovor komunikacijski obrazac te suradnju zasnovanu na adresama. Korisnik mora poznavati ime procedure te adresu poslužitelja pa su korisnik i poslužitelj referencijski povezani. Prilikom poziva procedure oba moraju biti aktivna što ih čini i vremenski

povezanim. Također, komunikacija je sinkrona jer korisnik čeka dok ne dobije rezultat od poslužitelja.

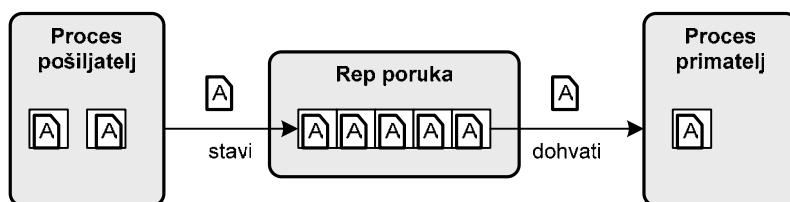


Slika 11: Poziv nad udaljenim objektom

Neka ostvarenja modela poziva nad udaljenim objektima kao što su CORBA, Java RMI, DCOM te .NET Remoting svrstavaju se u komunikacijsku programsку potporu odnosno komunikacijske protokole i nisu potpuni komunikacijski posrednički sustavi. Navedena ostvarenja vezana su uz određenu računalnu platformu te ne omogućuju međuplatformsku komunikaciju. S druge strane, *Web services* (WS) tehnologija ostvarena je pomoću općepoznatih, sveprisutnih i općeprihvaćenih protokola i zbog toga omogućuje komunikaciju različitih računalnih platformi. Dodatna razlika navedenih tehnologija i *Web services* komunikacijskog posredničkog sustava je što potonji omogućuje pozivanje metoda neovisnih udaljenih usluga umjesto objekata [14].

3.2.2. Posrednički sustavi zasnovani na razmjeni poruka

Model posredničkog sustava zasnovanog na razmjeni poruka (engl. *message-oriented middleware*, MOM) koristi repove putem kojih sudionici razmjenjuju poruke. Model koristi stavi/dohvati komunikacijski obrazac, a suradnja među sudionicima je zasnovana na adresama. Repovi poruka ostvaruju spremnik koji omogućuje postojanost poruka prilikom komunikacije, što je glavna prednost stavi/dohvati komunikacijskog obrasca. Osim toga, komunikacija među procesima je asinkrona i od točke do točke. Slika 12 prikazuje međudjelovanje raspodijeljenih procesa stavljanjem i dohvaćanjem poruka iz repa poruka.



Slika 12: Razmjena poruka putem repa

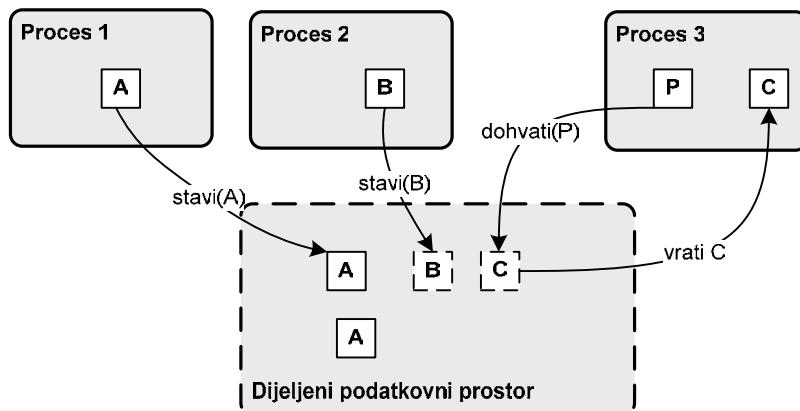
Pošiljatelji i primatelji su referencijski povezani jer pošiljatelj mora poznavati adresu repa poruka primatelja. Nadalje, vremenski su nepovezani jer poruka ostaje u repu dok je ne dohvati primatelj. Posljedica je da se pošiljatelj i primatelj ne moraju izvoditi istovremeno te je stoga komunikacija asinkrona. U jedan rep poruke može stavljati više pošiljatelja, a također više procesa može dohvaćati poruke iz istog repa. No svaka pojedina poruka namijenjena je samo jednom primatelju. Repovi poruka su ključne komponente za izgradnju posredničkih sustava zasnovanih na porukama. Stoga mnoga svojstva posredničkih sustava zasnovanih na porukama proizlaze iz svojstava repova poruka. Primjerice, ako dva podsustava koriste posrednički sustav zasnovan na razmjeni poruka, onda izmjene na jednom sustavu ne uključuju nužno izmjene drugog podsustava. Osim toga, promjene funkcionalnosti primjenskog programskega sustava ne zahtijevaju promjene samog posredničkog sustava zasnovanog na porukama jer je on neovisan o primjeni. Nadalje, posrednički sustav zasnovan na razmjeni poruka podržava veliki broj korisnika, njihovu prilagodljivu integraciju u sustav te anonimnost korisnika u sustavu. Dodatno, kako su razmjenjivane poruke tekstualnog oblika omogućuje se zajednički rad raznorodnih programskih sustava. Također, ujednačavanje opterećenja je prirođeno svojstvo sustava jer je izvođenje procesa vremenski učinkovito. Naime, proces primatelj preuzima poruku tek kada je spreman. S druge strane, proces pošiljatelj ne mora čekati da se poruka preuzeće. Navedena svojstva donose mnoge prednosti posredničkim sustavima zasnovanim na porukama nad sustavima zasnovanim na udaljenim pozivima objekata.

3.2.3. Dijeljeni podatkovni prostori

Dijeljeni podatkovni prostori ostvaruju stavi/dohvati obrazac međudjelovanja zasnovan na sadržaju. Koordinacija, odnosno komunikacija i suradnja među procesima ostvarena je putem operacija nad dijeljenim podacima. Dijeljeni podatkovni prostor ostvaren je kao dijeljena memorija u kojoj se pohranjuju podaci organizirani u *n-torce* [2, 13]. Jedna *n-torka* je nedjeljiv skup imenovanih vrijednosti od kojih svaka ima definiran tip. Proses pohranjuje n-torku u podatkovni prostor pomoću operacije *stavi(n-torka)* koja stvara instancu n-torke u dijeljenom podatkovnom prostoru. Opisani model koordinacije procesa naziva se *generativna komunikacija* zbog načina čitanja n-torki. Prilikom čitanja iz dijeljenog podatkovnog prostora pomoću operacije *dohvati(predložak)* proces definira predložak za uparivanje n-torki. Sam predložak također je n-torka, ali je moguće da neke vrijednosti n-torke ostanu nedefinirane. Uparivanje predloška i pohranjenih n-torki provodi se vrijednost po vrijednost. Ako uparivanje predloška i neke n-torke uspije ona se dohvaća u memorijski prostor procesa. Postoje i dvije dodatne operacije, *ukloni(predložak)*, koja uklanja n-torke koje odgovaraju predlošku, te *ispitaj(predložak)* koja ispituje da li postoje n-torke koje odgovaraju postavljenom predlošku. Operacije dohvati, ukloni i ispitaj mogu biti blokirajuće i neblokirajuće. Pozivom blokirajuće operacije proces pozivatelj je blokiran sve dok se ne pojavi n-torka koja zadovoljava

predložak, dok kod neblokirajućeg poziva proces nastavlja bez obzira da li je n-torka pronađena. Operacija *stavi* uvijek je neblokirajuća. Nadalje, komunikacija može biti višeodredišna jer više procesa može pročitati istu n-torku, sve dok ju neki proces ne ukloni iz dijeljenog podatkovnog prostora [12].

S obzirom da su procesi referencijski nepovezani, model dijeljenog podatkovnog prostora naziva se *asocijativna memorija*, jer n-torke nemaju fizičke adrese nego se podaci adresiraju putem njihovog sadržaja. Također, opisani model naziva se još i koordinacija zasnovana na podacima (engl. *data-oriented coordination*).



Slika 13. Međudjelovanje procesa putem dijeljenog podatkovnog prostora

Posljednjih desetljeća, model dijeljenih podatkovnih prostora uspješno je primijenjen u praktičnim rješenjima u velikom broju sustava. Primjerice koristi se za izgradnju paralelnih sustava, no također i u sustavima za suradnju zasnovanim na Web tehnologiji (engl. *Web-based collaboration system*). Model dijeljenih podatkovnih prostora uporabljen je pri izgradnji platformi poput Sun JavaSpaces te IBM T-Spaces. Svrha tih platformi je omogućiti upravljanje dinamičkim federacijama raznih računalnih uređaja. U federacijama se procesi natječu i surađuju pri uporabi računalnih sredstava. Kako kod dijeljenih podatkovnih prostora nema podjele na korisnike i poslužitelje, svi procesi mogu upotrebljavati sve raspoložive operacije. Stoga se u zadnje vrijeme model koristi za ostvarenje mreža ravnopravnih sudionika (engl. *peer-to-peer network*) u mobilnim okruženjima [11].

3.2.4. Objava/preplata

Objava/preplata model komunikacijskog posredničkog sustava ostvaruje objava/preplata/dojava model međudjelovanja zasnovan na sadržaju. Naziva se još i posrednički sustav zasnovan na događajima (engl. *event-based middleware*) jer je komunikacija među sudionicima poticana događajima, odnosno nastankom ili izmjenom informacija u sustavu [16]. Kako su događaji, preplate i dojave vrste poruka, objava/preplata posrednički sustav svrstava se u širi skup sustava zasnovanih na razmjeni poruka (engl. *message-based middleware*) [5]. Sustav pruža slabo vezanje sudionika komunikacije jer su preplatnici i objavitelji dojava referencijski nepovezani. Postoje ostvarenja u

kojima su sudionici vremenski povezani te u kojima su vremenski nepovezani. U sustavima u kojima su sudionici vremenski povezani preplatniku se dojavljuje događaj u slučaju da je on aktivan prilikom nastanka događaja. Kod sustava s vremenski nepovezanim sudionicima dojava se pohranjuje u sustavu sve dok preplatnik ne postane aktivan. U tom trenutku, preplatniku se uspješno dojavljuje događaj. Jedna od prednosti koje razlikuju objava/preplata model od drugih modela međudjelovanja je mogućnost višeodredišne komunikacije. Objavljeni događaj se uvišestručava te dostavlja na gurni način svim zainteresiranim preplatnicima.

3.3. Komunikacijski sustav zasnovan na modelu objava/preplata

Objava/preplata model međudjelovanja omogućava referencijski i vremenski nepovezanu te asinkronu komunikaciju među objaviteljima informacija i preplatnicima na informacije. Objavitelji i preplatnici komuniciraju izmjenjujući dojave (engl. *notifications*), koje su oblikovane kao događaji koji predstavljaju neku informaciju, odnosno nose objavljeni sadržaj. Objava/preplata model upravljan je događajima (engl. *event-driven*) jer je objavljivanje aperiodično i uzrokovano dostupnošću nove ili izmijenjene informacije, ili je potaknuto promjenom stanja nekog od objavitelja. Nakon što su događaji objavljeni oni se rasprostranjuju (engl. *dissemination*) zainteresiranim preplatnicima. Korisnike često ne zanimaju svi događaji, stoga oni preplaćivanjem izražavaju interes za kategorije događaja koje žele primati. U trenutku objave događaja, on se dostavlja svim korisnicima koji su se preplatili na tu kategoriju događaja. [5, 13, 18]

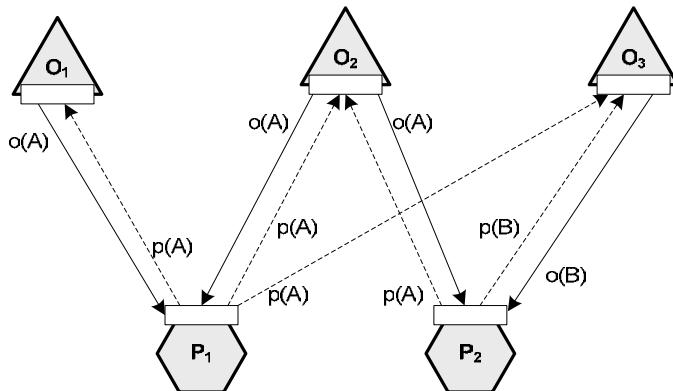
3.3.1. Posrednik u sustavu objava/preplata

U primjeru na slici 14 prikazan je sustav objava/preplata bez posrednika. U takvom ostvarenju zahtijeva se da preplatnici poznaju objavitelje, pa preplatnik P_1 poznaje objavitelje O_1 , O_2 i O_3 . No, preplatnik P_2 poznaje samo objavitelje O_2 i O_3 , te ako objavitelj O_1 objavi događaj kategorije A ili B onda događaj neće biti dojavljen preplatniku P_2 . Nadalje, problem nastaje i ako se novi objavitelj pridruži sustavu. U tom slučaju preplatnici bi trebali biti obaviješteni o spajanju novog objavitelja koji bi im mogao biti od interesa. Kako nije omogućeno učinkovito spajanje i odspajanje iz sustava, on nije proširiv.

Sljedeći nedostatak je da objavitelji moraju poslati isti događaj svim preplatnicima koji su na njega preplaćeni. Npr. O_2 šalje isti događaj A preplatnicima P_1 i P_2 jer su oba preplaćena na tu kategoriju događaja. Na taj način nastaje povećani mrežni promet zbog uvišestručavanja podataka koji se šalju. Kako P_2 ne poznaje O_1 on ne dobiva događaj kategorije A koji je objavio O_1 . Osim toga, objavitelj

sam brine o dostavljanju događaja preplatniku. Ako preplatnik nije aktivan u trenutku objavljivanja događaja, onda objavitelj mora kasnije ponovo pokušati dostaviti isti događaj.

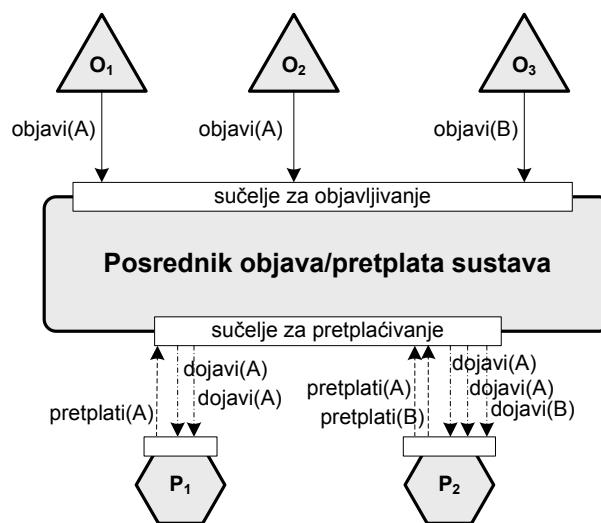
Da bi se korisnici mogli preplaćivati, svaki objavitelj mora ostvariti sučelje za preplaćivanje. Moguće je da objavitelji posjeduju različita sučelja pa preplatnici u tom slučaju moraju imati mogućnost preplaćivanja na različita sučelja.



Slika 14. Objava/preplata sustav bez posrednika

Većina nedostataka objava/preplata sustava bez posrednika proizlaze iz nužne referencijske povezanosti preplatnika i objavitelja.

Postoje ostvarenja objava/preplata sustava bez posrednika za lokalne mreže gdje se referencijska nepovezanost postiže višestrukim razašiljanjem događaja svim korisnicima. Međutim, većina objava/preplata sustava sadrži posrednik poput prikazanog u primjeru na slici 15. Posrednik upravlja preplatama, osigurava pohranu, obradu i učinkovitu dostavu događaja. Obrada događaja zadaća je mehanizma za uparivanje preplata s događajima. Uparivanje se provodi na način da se za svaku preplatu ispituje da li objavljeni događaj odgovara postavljenim kriterijima preplate, te ako odgovara onda se događaj dostavlja pripadnom preplatniku.



Slika 15: Sustav objava/preplata s posrednikom

Sudionici ne moraju poznavati ostale sudionike pa se ne moraju ni unaprijed povezati s određenim sudionicima. Na taj način izbjegava se povezivanje svakog sa svakim putem *jedan na jedan* obrasca komunikacije. Dakle, postojanje posrednika omogućava korisnicima učinkovito iskorištenje sustava s mnogo sudionika jer oni komuniciraju pomoću *više na više* obrasca.

Postojanje posrednika uklanja nedostatke objava/preplata sustava bez posrednika te omogućava referencijski i vremenski nepovezanu komunikaciju. Također, sustav s posrednikom je proširiv jer dopušta dolazak novih sudionika. Dodatno, komunikacija sudionika je asinkrona, a podaci se ne uvišestručavaju prilikom objavljivanja, jer se objavljeni događaj šalje posredniku samo jednom.

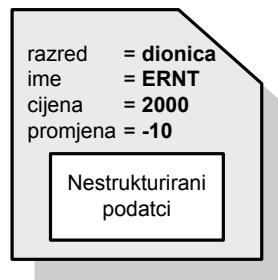
3.3.2. Preplaćivanje i objavljanje

Slika 15 prikazuje osnovni način međudjelovanja preplatnika i objavitelja po obrascu objava/preplata/dojava. Preplatnici se preplaćuju putem sučelja za preplaćivanje koje pruža objava/preplata posrednik. Pomoću metode *preplati* preplatnici zadaju jednu preplatu koja izražava interes za određene vrste događaja. Pojedini preplatnik može zadati više preplata. Npr. preplatnik P_2 preplaćen je na A i B vrste događaja. U trenutku prestanka interesa za neku vrstu događaja, preplatnici pomoću metode *otkaži_preplatu* otkazuju jednu od svojih preplata.

Korisnici objavljaju događaje putem metode *objavi* koja je dio sučelja za objavljanje. Prilikom nastanka događaja vrste A, objavitelj O_2 šalje događaj posredniku samo jednom, bez obzira koliko je preplatnika izrazilo interes za tu vrstu događaja. Uvišestručavanje i dostava događaja preplatnicima uloga je posrednika, pa on dostavlja događaj vrste A preplatnicima P_1 i P_2 . Posrednik preplatnicima šalje još jedan događaj vrste A jer je i O_1 objavio takav. Ovisno o svojstvu postojanosti događaja, posrednik može čuvati događaj sve dok se preplatnik ne spoji na sustav, ako preplatnik nije bio dostupan u trenutku objavljivanja događaja.

Dojava

Informacije koje proizvode objavitelji općenito se nazivaju *dojave*. U mnogim primjenama koristi se naziv *događaji* budući da dojave opisuju važne događaje u sustavu. Objavitelji proizvode dojave aperiodično, odnosno u proizvoljnim vremenskim trenutcima. Dojave su najčešće tekstualne, ali mogu biti zadane u raznim jezicima za opisivanje dojava (engl. *notification language*). Često se upotrebljavaju jednostavne dojave koje sadrže skup parova *atribut-vrijednost* te dodatne nestrukturirane podatke (engl. *payload*). U primjeru na slici 16 atributi su *razred*, *ime*, *promjena* i *cijena*. Vrijednosti tih atributa redom su *dionica*, *ERNT*, *2000* te *-10*.

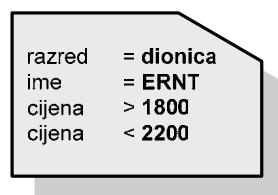


Slika 16: Primjer dojave

Dojava može također sadržavati i neka općenita svojstva: jedinstveni identifikator, vremensku oznaku (engl. *time stamp*) trenutak nastanka, vijek trajanja, prioritet. Objavitelji dojave mogu definirati dodatna svojstva ovisno o primjenskom programu koji koristi objava/preplata sustav. Prilikom prijenosa mrežom dojava je umetnuta u poruku čiji format ovisi o sloju mreže u kojem je objava/preplata sustav ostvaren. U zaglavje poruke zapisuju se adrese izvořišta i odredišta, odnosno adresa objavitelja i adresa objava/preplata posrednika, dok se u tijelo poruke zapisuje sama dojava.

Pretplata

Kao i dojava, pretplata je tekstualni oblik podatka i opisuje se pomoću *jezika za pretplaćivanje* (engl. *subscription language*). Postoje razni mehanizmi za pretplaćivanje te mehanizmi za filtriranje događaja. Ako je dojava zadana kao skup parova atributa i vrijednosti, onda je jedan način zadavanja pretplata u obliku konjunkcije ograničenja nad vrijednostima atributa. Primjerice, ograničenja je moguće izraziti pomoću logičkih i relacijskih operatora: $(razred=dionica) \ \& \ (ime=ERNT) \ \& \ (cijena > 1800) \ \& \ (cijena < 2200)$.



Slika 17: Primjer pretplate

$$P(razred,ime,cijena) \equiv (razred = dionica) \wedge (ime = ERNT) \wedge (cijena > 1800) \wedge (cijena < 2200).$$

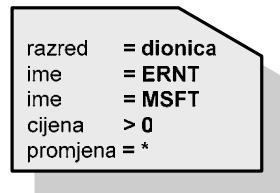
Pretplata zadana u gornjem primjeru je logički predikat nad varijablama *razred, ime, cijena*.

Objava/preplata posrednik pohranjuje pretplate te za svaku pretplatu određuje istinitost predikata s obzirom na trenutno objavljenе događaje.

Oglas

Za razliku od pretplate kojom se definiraju dojave koje su od interesa, oglas definira neki skup dojava koje korisnik može proizvoditi. Razlog za uvođenje oglasa kao dodatnog razreda poruka je obavještavanje objava/preplata sustava o vrsti dojava koje korisnik proizvodi. Oglasi se koriste u

raspodijeljenim arhitekturama objava/preplata sustava za ostvarenje mehanizma rasprostiranja preplata.



Slika 18: Primjer oglasa

Oglas se zadaje skupom ograničenja, odnosno na isti način kao i preplata. U primjeru na slici 18 oglas definira dojave u kojima atribut *razred* poprima isključivo vrijednost „*dionica*“. Atribut *cijena* može poprimiti bilo koju vrijednost veću od 0. Korištenjem zamjenskog znaka „*“ dopušta se da pripadni atribut poprimi bilo koju vrijednost iz skupa. U primjeru, korisnik proizvodi dojave u kojima atribut *promjena* može poprimiti bilo koju vrijednost. Ako je više ograničenja zadano nad istim atributom, onda među njima vrijedi disjunkcija. Prema tome, dojava mora zadovoljiti jedno od ograničenja. U primjeru atribut *ime* poprima vrijednost „*ERNT*“ ili vrijednost „*MSFT*“.

3.3.3. Mehanizmi preplaćivanja i filtriranja

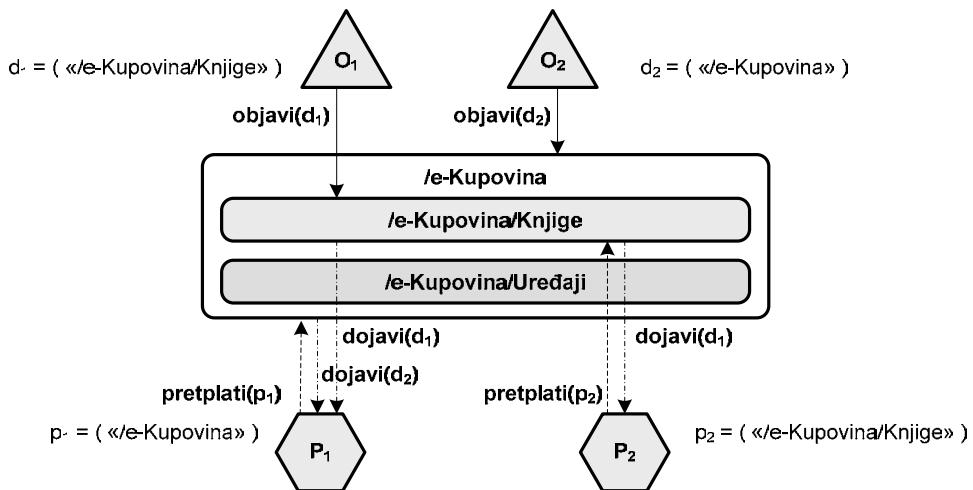
Pretplatnicima su korisni samo neki događaji, odnosno grupe, vrste ili uzorci događaja. Stoga preplata definira interes pretplatnika u obliku skupa pravila po kojima se filtrira skup dojava. Izražajne mogućnosti preplaćivanja i filtriranja koje pruža sustav objava/preplata važna su svojstva koja se od sustava do sustava znatno razlikuju. Razvijeni su mnogi mehanizmi preplaćivanja i filtriranja događaja, te danas postoji filtriranje prema kanalu, temi, tipovima, sadržaju i uzorcima.

3.3.3.1 Filtriranje prema kanalu

Događaji su kategorizirani u unaprijed definirane kanale i svaki se događaj objavljuje na određenom kanalu. Pretplatnici se preplaćuju na kanale te dobivaju sve događaje koji su objavljeni na tom kanalu. Kanali su slični pojmu grupe u kontekstu višestrukog razašiljanja u grupnoj komunikaciji koja je mehanizam za učinkovito dostavljanje istog podatka na više odredišta. Jedan od prvih sustava zasnovanih na objava/preplata modelu je Isis koji pruža mogućnosti grupne komunikacije. Korisnici se učlanjuju u određene komunikacijske grupe, odnosno priključuju na kanale. Primjer uporabe grupne komunikacije je održavanje stroge konzistencije među replikama ključnih komponenti u lokalnoj mreži. Kanali su ostvareni pomoću višestrukog razašiljanja u mrežnom sloju (engl. *IP multicast*), a svaki kanal odgovara određenoj grupi za razašiljanje.

3.3.3.2 Filtriranje prema temi

Filtriranje prema temi je vrlo slično filtriranju prema kanalu. Razlika je da je tema predstavljena s tekstuallnom ključnom riječi, npr. „Knjige“. Kako korisnici rukuju tekstuallnim temama, filtriranje prema temi nudi višu razinu apstrakcije, a svaka tema se preslikava u posebnu komunikacijsku grupu. Primjena je također drugačija, jer nasuprot filtriranju prema kanalu koje se koristi u programskim sustavima za lokalne mreže, filtriranje prema temi koristi se za programske sustave većih razmjera. Osim toga, s obzirom da se za preplaćivanje koriste tekstuallne teme, omogućen je i zajednički rad korisnika različitih platformi. Proširenje koje uvodi mehanizam filtriranja prema temi je hijerarhija tema, koja se definira pomoću formata sličnog URL formatu, npr. „/e-Kupovina/Knjige/Računarstvo“. Korištenjem hijerarhije tema, odnosno podtema moguće je postizanje veće zrnatosti filtriranja događaja. Svaka podtema pruža preciznije grupiranje događaja od nadteme u kojoj se nalazi. Preplata na temu automatski uključuje i preplatu na sve podteme u hijerarhiji. U primjeru na slici 19, preplatnik P_1 preplaćen je na temu /e-Kupovina i sve ostale podteme. Kada objavitelj O_1 objavi događaj d_1 na podtemu „/e-Kupovina/Knjige“ on se proslijeđuje do oba preplatnika. Događaj d_2 objavljen od O_2 proslijeđuje se samo preplatniku P_1 . Također, većina sustava omogućava uporabu zamjenskih znakova (engl. wildcards) u imenima tema prilikom preplaćivanja, npr. „/e-Kupovina/*Rač*“. Preplata zadana na taj način omogućuje primanje događaja objavljenih na podteme koje su ugniježđene unutar tema Knjige i Uredaji, a počinju s prefiksom Rač.



Slika 19: Filtriranje prema temi

U usporedbi s filtriranjem prema kanalu, filtriranje prema temi nudi veću mogućnost izražavanja, no preplaćivanje je ograničeno na statički raspored hijerarhije tema. Također, veliki broj podtema može dovesti do bujanja hijerarhijskog stabla što je značajan problem za programsko ostvarenje, ako je ono zasnovano na razašiljanju u mrežnom sloju. U tom slučaju svaka podtema trebala bi biti ostvarena kao zasebna grupa za razašiljanje, a broj takvih grupa je konačan [13]. Zbog statičnosti ostvarenja, filtriranje prema temi je vremenski učinkovito jer se objavljuje i preplaćuje na unaprijed definirane

teme. Prema tome, prilikom objave posrednik proslijeđuje događaj preplatnicima i nikakva međuobrada događaja nije potrebna.

3.3.3.3 Filtriranje prema sadržaju

Usprkos unapređenjima poput hijerarhije tema te zamjenskih znakova, filtriranje putem tema predstavlja statički mehanizam s ograničenom mogućnošću izražavanja. Mehanizam filtriranja zasnovan na sadržaju pruža daljnje unapređenje filtriranja prema temama tako što omogućuje preplaćivanje koje uzima u obzir sam sadržaj promatranih događaja. Naime, događaji nisu raspoređeni prema unaprijed definiranom vanjskom kriteriju nego prema samim svojstvima događaja. U sustavima Gryphon [23], SIENA [25], Elvin [24] i JEDI [26] svojstva su interni atributi podatkovnih struktura. Također, svojstva mogu biti i meta-podaci i takav pristup koristi sustav *Java Message Service* [27]. Preplatnici se preplaćuju na odabране događaje koristeći jezik za opisivanje preplata. Preplata predstavlja filter u obliku definiranih ograničenja kojima se ispituje valjanost pojedinog događaja. Ograničenja su najčešće zadana kao trojke *atribut-operator-vrijednost* koristeći relacijske operatore ($=$, $<$, $>$, ...). Vezanje više ograničenja omogućeno je pomoću logičkih operatora ($\&$, $|$, ...) čime se dobivaju logički predikati koji se još nazivaju *preplatnički uzorci* (engl. *subscription pattern*). Prilikom preplaćivanja, pozivom metode *preplati* preplatnik dostavlja i *preplatnički uzorak* kao jedan od parametara. Postoje razni načini ostvarivanja preplatničkih uzoraka: tekstualni predikati i upiti, objekt predložak i izvodljivi kod.

Tekstualni predikati i upiti

Preplatnički uzorci najčešće se izražavaju pomoću tesktualnih poruka koje su logički predikati ili upiti. Mehanizam za filtriranje i preplaćivanje objava/preplata sustava sadrži ugrađeni analizator pomoću kojeg parsira preplate i događaje. Također, sadrži filter za provedbu ispitivanja koje sve preplate zadovoljava pojedini objavljeni događaj.

Na primjer, jezik za opisivanje upita u JMS sustavu zasnovan je na podskupu sintakse za postavljanje uvjeta WHERE konstrukta SQL-92 jezika [27]. Pri radu s bazama podataka taj konstrukt služi za postavljanje uvjeta pri dohvaćanju podataka. Stoga je njegova općepoznatost iskorištena i za opisivanje preplata, koje se u JMS sustavu zovu *selektori poruka* (engl. *message selector*). Npr. upit se zadaje na sljedeći način: „*symbol='IBM' AND količina>1000*“.

Zajedno s pojavom XML-a kao standarda za razmjenu informacija putem mreže Internet, javlja se potreba za mehanizmima preplaćivanja i filtriranja XML dokumenata koji omogućuju veliku izražajnost. Jedan od jezika na kojima se zasniva ostvarivanje mehanizama koji iskorištavaju i strukturu i sadržaj objavljenih XML dokumenata je XPath [29]. Jezik je standardizirala udruga W3C, a namijenjen je za adresiranje dijelova XML dokumenta. Također, u mnogim sustavima za rasprostiranje XML dokumenata koristi se kao jezik za preplaćivanje i filtriranje. U [29] opisan je

sustav za filtriranje XML dokumenata *XFilter*, koji omogućava visoko učinkovito uparivanje XML dokumenata s velikim brojem korisničkih preplata. Korisničke preplate oblikovane su kao upiti koji se zadaju pomoću *XPath* jezika.

Među poznatijim jezicima još je i *OMG Default Filter Constraint Language* [31] koji koristi *CORBA Notification Service* sustav.

Objekt predložak

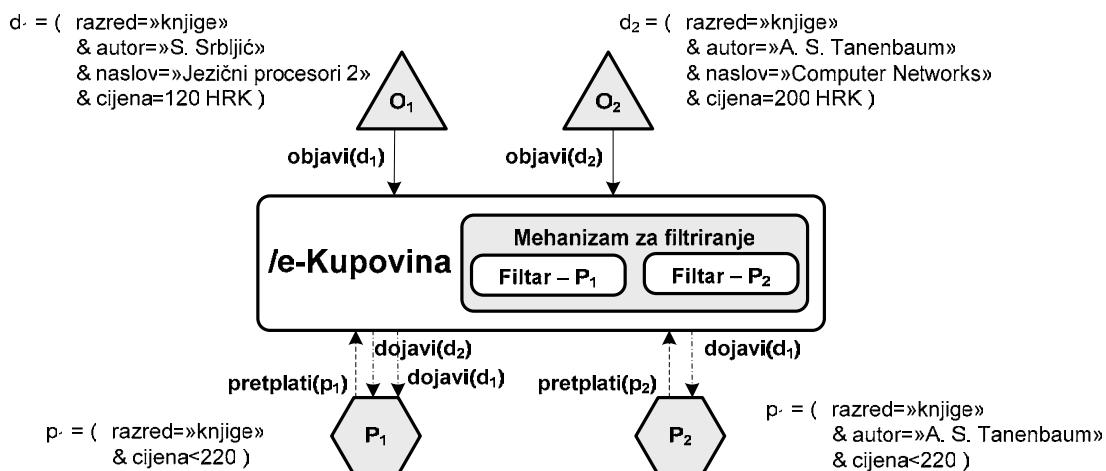
Slično kao kod dijeljenih podatkovnih prostora, filtriranje je zasnovano na uparivanju n-torki. Preplata je definirana pomoću objekta-predloška t koji je strukturiran kao n-torka parova $atribut=vrijednost: t=(a_1=v_1, \dots, a_n=v_n)$. Objekt označava pretplatnikov interes za događaje istog tipa kao i t te one čiji su atributi jednaki odgovarajućim atributima od t . Ako je neki atribut od t nedefiniran (engl. *null*), onda taj atribut događaja može poprimiti bilo koju vrijednost.

Izvodljivi kôd

Korisnici se preplaćuju putem filtarskog predikatnog objekta koji ima mogućnost filtriranja događaja. Filtarski objekt je prilikom preplaćivanja već preveden u izvodljivi kôd, te njegovo parsiranje kao kod tekstualnih preplata nije potrebno. S obzirom da filtarski objekt oblikuje i ostvaruje razvijatelj u strogo obilježenom (engl. *strongly typed*) objektno orijentiranom jeziku, poseban jezik za preplaćivanje nije potreban. Dodatno, detalji korisničke preplate učahureni su u izvodljivom kodu objekta što je jedna od prednosti ovakvog pristupa. Također, postoji ostvarenje [32] u kojem se preplate zadaju kombiniranjem *filtarskih objekata* opće namjene iz zadane knjižnice. Korisnik definira metode koje se dinamički pozivaju nad objektima događajima putem mehanizma introspekcije. Dodatno, korisnik definira semantiku vrednovanja rezultata izvođenja tih metoda koja određuje koji objekti događaji zadovoljavaju preplatu. Optimiranje preplata ostvarenih izvodljivim kodom je složeno. Osim toga, preplate se primjenjuju slijedno na svaki događaj posebno. Zbog navedenog, rješenje ostvareno preplatama u obliku izvodljivog kôda može biti vremenski zahtjevno i neučinkovito [18].

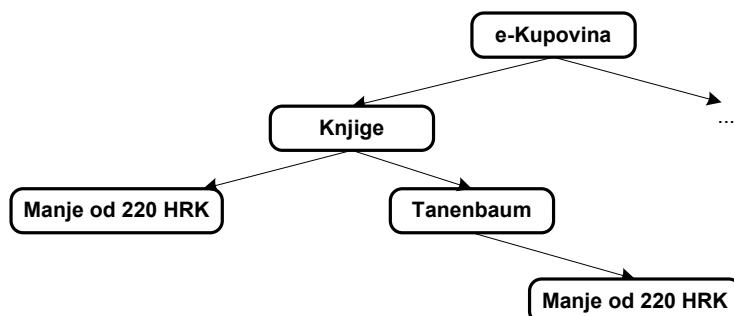
Usporedba filtriranja prema sadržaju i filtriranja prema temi

U primjeru na slici 20 prikazano je preplaćivanje i filtriranje pomoću predikata koje preplatnici oblikuju kao tekstualne poruke. Cjelokupni prostor događaja spada u jednu temu, a događaji se filtriraju prema preplatničkim tekstualnim predikatima. Preplatnika P_1 zanimaju knjige jeftinije od 220 HRK. Preplata je određena tekstualnim predikatom ($\text{razred} = \text{"knjige"} \& \text{cijena} < 220 \text{ HRK}$). Preplatnika P_2 također zanimaju knjige jeftinije od 220 HRK no samo one od autora A.S. Tanenbaum, a pripadni predikat je ($\text{razred} = \text{"knjige"} \& \text{autor} = \text{"A. S. Tanenbaum"} \& \text{cijena} < 220 \text{ HRK}$).



Slika 20: Filtriranje prema sadržaju

Filtriranje prema sadržaju nudi veću mogućnost izražavanja u odnosu na filtriranje prema temi. Za ostvarivanje jednakih mogućnosti izražavanja pomoću filtriranja prema temi postojećim temama dodaju se podteme za svako ograničenje.

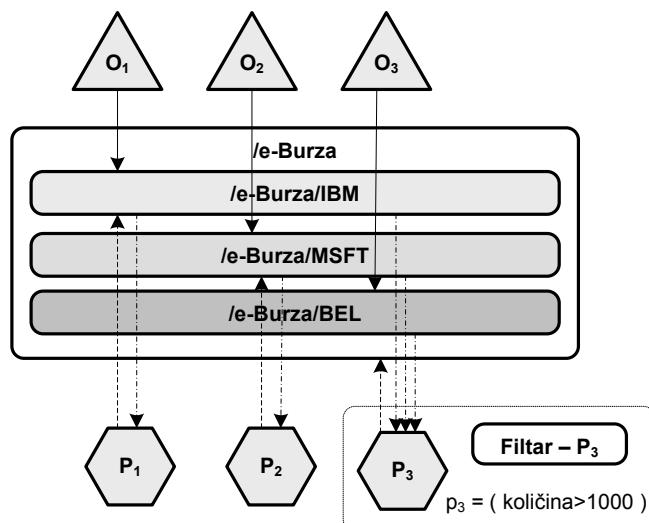


Slika 21: Ekvivalentno stablo tema za predikatno zadatu preplatu

Primjerice na slici 21 prikazan je postupak izgradnje ekvivalentnog stabla tema za preplate P_1 i P_2 . Za svako ograničenje u preplati dodaje se zasebna podtema u podstablo koje odgovara toj preplati. Osim toga, moguće je pojavljivanje iste podteme u različitim podstablima. Zbog toga, izgradnja stabla takvim postupkom može učiniti stablo vrlo složenim i zalihosnim. Nadalje, nije moguće predvidjeti

sve interese preplatnika te unaprijed izgraditi statično stablo tema. Stoga je svojstvo dinamičnosti jedno od prednosti filtriranja prema sadržaju nad filtriranjem prema temi.

Umjesto izgradnje statičkog stabla s podtemama za svako ograničenje, alternativno rješenje je prepustiti filtriranje preplatnicima. Slika 22 prikazuje sustav u kojem posrednik ostvaruje filtriranje prema temi. Objavitelji objavljuju na tri teme, odnosno objavljuju događaje vezane uz tri vrste dionica. Preplatnici P_1 zanimaju događaji u temi $/e\text{-Burza}/IBM$, a P_2 događaji u temi $/e\text{-Burza}/MSFT$. Poseban slučaj čini preplatnik P_3 kojeg zanimaju događaji uz sve tri vrste dionica, uz uvjet da je količina ponuđenih dionica veća od 1000. Preplatnik P_3 preplaćuje se na sve tri teme jer se u svakoj može pojaviti događaj u kojem je količina ponuđenih dionica veća od 1000. Stvara se znatan mrežni promet između preplatnika P_3 i posrednika jer mu se proslijeduju svi objavljeni događaji. Također, preplatnik P_3 dodatno je opterećen jer mora sam filtrirati sve pristigle događaje ispitivanjem svakog od njih. Stoga je nedostatak i dodatni trošak ovakvog pristupa vrijeme preplatnika koje se troši na ispitivanje velikog broja događaja koji su njemu nezanimljivi.



Slika 22: Filtriranje prema temi s dodatnim filtriranjem prema sadržaju koje obavlja preplatnik

Stablo odlučivanja za filtriranje prema sadržaju

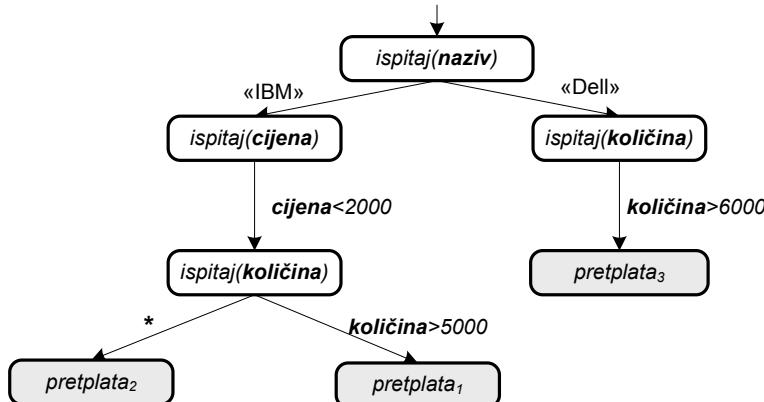
Određivanja skupa preplata čiji su predikati zadovoljeni svaki put kada se objavi novi događaj je usko grlo u sustavima s filtriranjem prema sadržaju koji se koriste u stvarnim primjenama gdje postoji znatan broj preplata i događaja. Jednostavan algoritam uparivanja prilikom objave novog događaja za sve preplate ispituje da li su pripadni predikati zadovoljeni. Vrijeme izvođenja algoritma linearno ovisi o broju preplata te je stoga algoritam neprimjenljiv u okruženjima gdje postoji mnogo preplatnika i objavitelja. Osnovna podloga za poboljšanje algoritma je velika vjerojatnost da više preplatničkih predikata ima zajednički podizraz. Navedena činjenica koristi se u postupku ispitivanja pa se zadovoljenost zajedničkog podizraza ne ispituje za svaki predikat posebno nego samo jednom. Jedan od načina ostvarenja takvog postupka ispitivanja je izgradnjom stabla odlučivanja. Zajednički

podizrazi predikata spajaju se u podstabla, pa se istovremeno ispituje zadovoljenje ograničenja za više pretplata odjednom. Primjer na slici 23 prikazuje izgradnju stabla odlučivanja za sljedeće tri preplate.

$$\text{preplata}_1 \equiv (\text{naziv} = "IBM") \wedge (\text{cijena} < 2000) \wedge (\text{količina} > 5000)$$

$$\text{preplata}_2 \equiv (\text{naziv} = "IBM") \wedge (\text{cijena} < 2000)$$

$$\text{preplata}_3 \equiv (\text{naziv} = "Dell") \wedge (\text{količina} > 6000)$$



Slika 23: Primjer stabla odlučivanja

Čvorovi stabla predstavljaju poziv procedure koja ispituje vrijednosti jednog atributa događaja, dok grane predstavljaju zadovoljavajuće ishode ispitivanja. Ako je grana označena sa * onda se ispitivanje u pripadnom podstablu nastavlja neovisno o ishodu (engl. *don't care edge*). U primjeru u korijenu stabla ispituje se vrijednost atributa *naziv*. Dvije preplate, *preplata₁* i *preplata₂*, zahtijevaju istu vrijednost atributa *naziv* („IBM“) i za tu vrijednost se daljnje ispitivanje provodi u lijevom podstablu. Ako je vrijednost atributa *naziv* jednaka „Dell“ onda se ispitivanje nastavlja u desnom podstablu. Preplata *preplata₂* ne postavlja ograničenje na količinu, što je u stablu označeno s granom *. Stoga se ispitivanje u podstablu ispod grane nastavlja bez obzira na ishod ispitivanja. Ako algoritam uspije ispitivanjem doći do nekog lista stabla, onda se događaj dojavljuje pretplatniku čijom je preplatom označen pripadni list.

Stablo odlučivanja ažurira se prilikom primanja nove preplate tako da se novi čvorovi dodaju na pripadna mjesta u stablu. U sustavu gdje je učestalost nastanka novih preplate velika može doći do pada učinkovitosti zbog obrade preplate i dodavanja čvorova u veliko stablo.

Svaki od mehanizama filtriranja ima različite izražajne mogućnosti preplaćivanja te različitu kvalitetu. Filtriranje prema temi je statično i jednostavno, no može se ostvariti vrlo učinkovito. S druge strane, filtriranje prema sadržaju je dinamično i prilagodljivo te ima velike izražajne mogućnosti preplaćivanja, ali uključuje složene mehanizme koji su vremenski vrlo zahtjevni. Zbog toga se nastoji upotrijebiti statične teme kad god je to moguće. U slučaju da osnovni atributi događaja poprimaju konačan skup diskretnih vrijednosti, npr. atribut *vrsta_akcije* poprima vrijednosti „ponuda“ i „potražnja“, za ostvarenje je moguće upotrijebiti teme. Dodatna izražajnost postiže se kombiniranim modelom koji koristi i teme i predikate. Filtriranje prema sadržaju koristi se u kontekstu statički

definiranih tema, kako bi se izrazila ograničenja nad atributima koji ne poprimaju diskretan skup vrijednosti (npr. *cijena dionica*).

3.3.3.4 Filtriranje prema uzorcima događaja

Mnogi objava/preplata sustavi ograničavaju pretplaćivanje samo na individualne događaje pa im stoga nedostaje mogućnost izražavanja interesa za pojavljivanje *uzoraka događaja* (engl. *patterns of events*). Međutim, u mnogim primjenama, posebice velikih razmjera, pretplatnici primaju veliku količinu jednostavnih događaja niske razine, te bi im dodatno koristio uvid u događaje na višoj razini. Pregled pojavljivanja događaja na višoj razini postiže se pomoću *složenog događaja* (engl. *composite event*, CE) koji nastaje pri pojavljivanju određenog skupa jednostavnih događaja po zadanim uzorku. U današnjim ostvarenjima uglavnom je zadaća pretplatnika ostvariti mehanizam za otkrivanje složenih događaja i zbog toga je pretplatnik nepotrebno složen i sklon dodatnim greškama.

Sustav SIENA omogućuje pretplaćivanje na *uzorce događaja*. Uzorci događaja zadaju se kao niz filtra $f_1 \ f_2 \dots f_N$. Svaki od filtera f_i zadan je kao skup ograničenja nad atributima događaja. Uzorak događaja zadovoljen je ako se na objava/preplata posredniku pojavi niz događaja koji redom zadovoljavaju filtre f_1 do f_N .

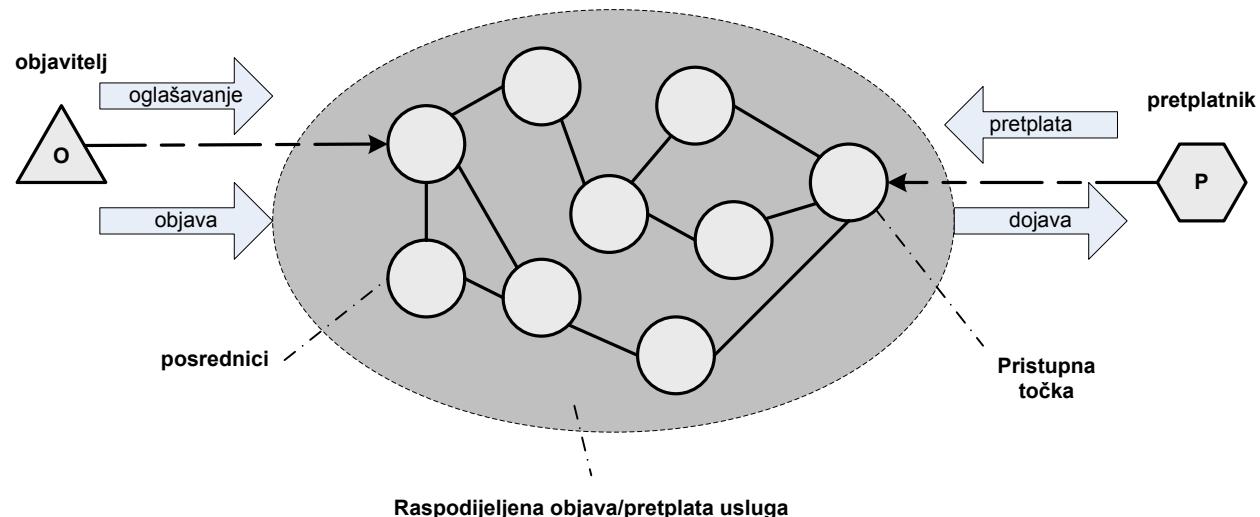
U radu [28] predložen je općeniti radni okvir za otkrivanje složenih događaja ostvaren korištenjem postojećih objava/preplata sustava. Radni okvir uključuje općeniti jezik za opisivanje složenih događaja te mehanizme za otkrivanje složenih događaja koje je moguće ostvariti na raspodijeljeni način.

Neki sustavi kao što je *Cambridge Event Architecture* (CEA) dodatno pružaju pretplaćivanje na korelaciju više događaja. Preplatom se definiraju logičke kombinacije osnovnih događaja. Također, zadaju se i vremenski intervali u kojima se događaji trebaju pojaviti. Mehanizam za otkrivanje složenih događaja ostvaruje se pomoću konačnog automata.

3.4. Raspodijeljena arhitektura komunikacijskog sustava zasnovanog na modelu objava/preplata

Do sada su razvijeni mnogi objava/preplata sustavi, no većina je namijenjena uporabi u lokalnim mrežama. Arhitekture takvih sustava većinom su centralizirane, odnosno oslanjaju se na centralnog posrednika koji obavlja sakupljanje i filtriranje događaja te njihovo dostavljanje preplatnicima. Osim toga sustav se teško prilagođava razmernom rastu broja preplatnika i objavitelja, ali i povećanju njihove fizičke rasprostranjenosti. Nadalje, sustav zasnovanom na centralnom posredniku nije primjenjiv za mreže širokih razmjera kao što je Internet [17]. Da bi sustav imao svojstvo razmernog rasta sve njegove funkcionalnosti moraju imati svojstvo rasta, a mehanizmi koji upravljaju sustavom moraju biti ostvareni tako da ne zahtijevaju poznavanje globalnog stanja sustava [16]. Centralni posrednik ima ograničenu radnu memoriju, mogućnosti obrade podataka te propusnost mreže. Stoga, ako se sustav oslanja na centralnog posrednika koji sam ostvaruje glavne funkcionalnosti sustava, onda je centralni posrednik usko grlo sustava.

Mnogi sustavi rješavaju problem razmernog rasta tako da funkcionalnost sustava raspodjeljuju na više ravnopravnih posrednika u mreži kao što prikazuje slika 24. Nadalje, nastoji se da sustavi koriste mehanizme kojima je dovoljno lokalno poznavanje sustava i lokalna komunikacija sa susjednim čvorovima sustava. U općenitom obliku, raspodijeljena arhitektura objava/preplata sustava sastoji se od povezanih ravnopravnih posrednika (engl. *brokers*), a svaki od njih poslužuje samo podskup svih korisnika sustava. S obzirom da su funkcionalnosti objavljivanja i pretplaćivanja jednostavne i ne zahtijevaju mnogo računalnih sredstava, jedan korisnik sustava može po potrebi obavljati obje uloge. Korisnici međudjeluju sa posrednikom koji im je u lokalnom dosegu.



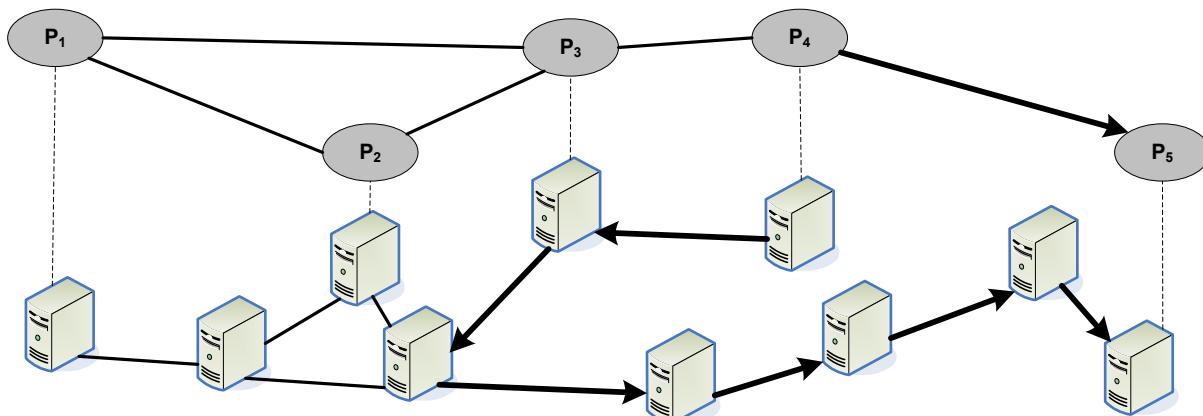
Slika 24: Raspodijeljena arhitektura objava/preplata sustava

Osim toga, on im služi kao pristupna točka za cijelokupnu objava/pretplata uslugu koja je rasprostranjena na području cijele mreže. U stvarnim sustavima arhitektura objava/pretplata usluge ostvarena je pomoću prekrivne mreže (engl. *overlay network*) širokih razmjera sastavljene od posrednika za obavljanje filtriranja i usmjeravanje dogadaja. Prekrivna mreža ostvaruje se povrh neke fizičke komunikacijske mreže poput Interneta. Jedan od mogućih razmještaja posrednika je smještanje po jednog posrednika u svaku administrativnu domenu unutar fizičke komunikacijske mreže.

3.4.1. Prekrivna mreža za usmjeravanje

Prekrivna mreža za usmjeravanje logička je mreža ostvarena u primjenskom sloju povrh mrežnog sloja koji je zasnovan na IP protokolu. Topologiju prekrivne mreže čini graf susjedstva koji je definiran skupom čvorova i vezama među njima. Susjedni čvorovi u fizičkoj mreži nisu nužno susjedni i u prekrivnoj mreži, odnosno logička topologija ne mora nužno očrtavati fizičku topologiju. Čvorovi prekrivne mreže usmjeravaju poruke šaljući ih jednom od svojih susjeda u prekrivnoj mreži. S obzirom da je mehanizam usmjeravanja ostvaren u primjenskom sloju mreže moguće je koristiti složenije algoritme za usmjeravanje zasnovane na uslugama nižih slojeva mreže. Postoji mnogo ostvarenja prekrivnih mreža za usmjeravanje koje pružaju uslugu više razine nego što to pruža usmjeravanje na mrežnom sloju. Dodatna svojstva koja nudi usluga više razine su višestruko razšiljanje, otpornost na pogreške (engl. *fault-tolerance*), neovisnost o položaju te anonimnost.

Veze između čvorova u prekrivnoj mreži posjeduju različiti trošak slanja poruke. Naime, jedna veza u prekrivnoj mreži ne mora nužno biti ostvarena jednom vezom fizičke mreže, odnosno može biti ostvarena i s više veza fizičke mreže. Također, latencija fizičkih veza može varirati od brzih LAN veza do sporih WAN veza [19, 20]. U primjeru na slici 25, posrednik P_4 ima za susjeda P_5 u prekrivnoj mreži no njihova je geografska udaljenost mnogo veća u fizičkoj mreži. Najkraći put među njima u fizičkoj mreži dug je 6 bridova. Stoga je komunikacija između ta dva čvora vrlo skupa jer koristi velik dio fizičke mreže.



Slika 25: Prekrivna mreža

3.5. Postojeći sustavi zasnovani na modelu objava/preplata

S obzirom da je osnovna zadaća objava/preplata sustava omogućavanje filtriranja i dostave događaja preplatnicima, pri oblikovanju stvarnog sustava postavlja se izazov maksimiziranja izražajnosti mehanizma filtriranja bez žrtvovanja svojstva razmernog rasta cijelog sustava. U pododjeljku 3.3.3 opisano je kako svojstvo izražajnosti određuje mogućnosti jezika za opisivanje preplata, događaja i mehanizma filtriranja. Također, ovisno o svojstvu izražajnosti moguće je veće ili manje optimiranje mehanizma dostave događaja.

Svojstvo razmernog rasta ne uključuje samo rast broja objavitelja događaja i broja preplatnika nego i primjenjivost sustava na mreže širokih razmjeru. Naime, neki sustavi ovise o mnogim prepostavkama koje su uvedene zbog uporabe sustava u lokalnim mrežama. Primjerice, podrazumijeva se niska latencija, velika propusnost mreže, homogena platforma, kontinuirano i pouzdano povezivanje sudionika, te centralizirano upravljanje sustavom.

U [17] naveden je radni okvir za razredbu sustava prema topologiji sustava koja utječe na svojstvo razmernog rasta sustava, te prema mehanizmu preplaćivanja koji utječe na svojstvo izražajnosti.

topologija			
	centralizirana	hijerarhijska	ravnopravni sudionici
prema kanalu	CORBA ES Java ES	IBM MQS, CORBA ES (IONA)	IP višestruko razašiljanje, iBUS, IRC
prema temi	ToolTalk	NNTP, JEDI	TIB-Rv, SmartSockets, BusinessWare
prema sadržaju	CORBA NS, JMS, Elvin	Keryx	Gryphon
prema sadržaju i uzorcima	Yeast, GEM, aktivna baza podataka		SIENA

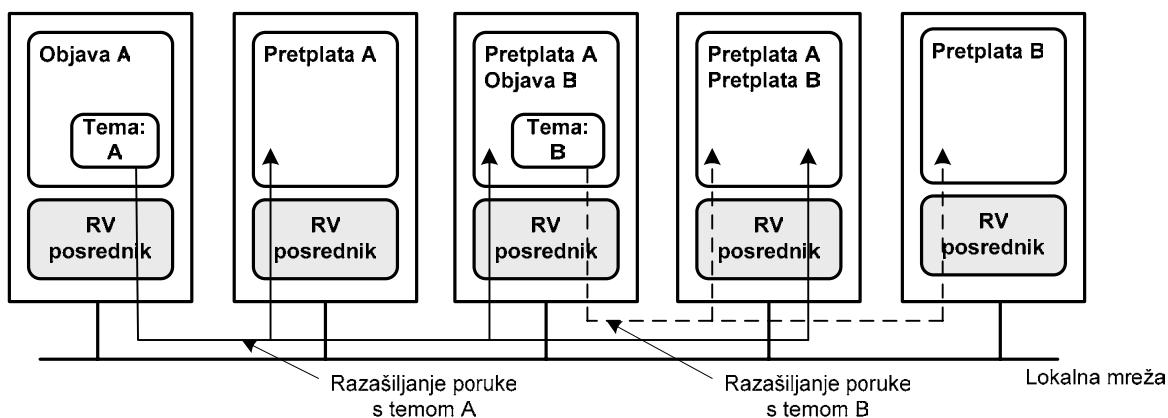
Tablica 4: Razredba sustava prema svojstvu izražajnosti i svojstvu razmernog rasta

U tablici 4 prikazana je razredba postojećih objava/preplata sustava prema svojstvu izražajnosti i svojstvu razmernog rasta. Svojstvo razmernog rasta ovisi o topologiji mreže posrednika. U poglavlju 4 opisani su postupci oblikovanja mreže posrednika te postojeće topologije mreža.

3.5.1. TIBCO TIB/Rendezvous Bus

TIB/Redezvous sustav razvila je tvrtka TIBCO kao posrednički komunikacijski sustav zasnovan na razmjerni poruka [2]. Sustav se često opisuje kao *informacijska sabirnica* (engl. *information bus*) jer pruža osnovnu komunikacijsku funkcionalnost te skriva detalje mreže od korisnika. Sustav omogućava razmjenu poruka po obrascima zahtjev/odgovor i objava/preplata. Poruke koje sustav upotrebljava neovisne su o primjeni te su samoopisne. Primjenski program koji koristi sustav ispituje poruku kako bi otkrio strukturu i sadržaj poruke. Sustav je ostvaren prema inaćici objava/preplata obrasca gdje su procesi vremenski povezani. Ključna značajka sustava je suradnja među procesima zasnovana na sadržaju, dok se preplaćivanje na poruke odvija prema temi.

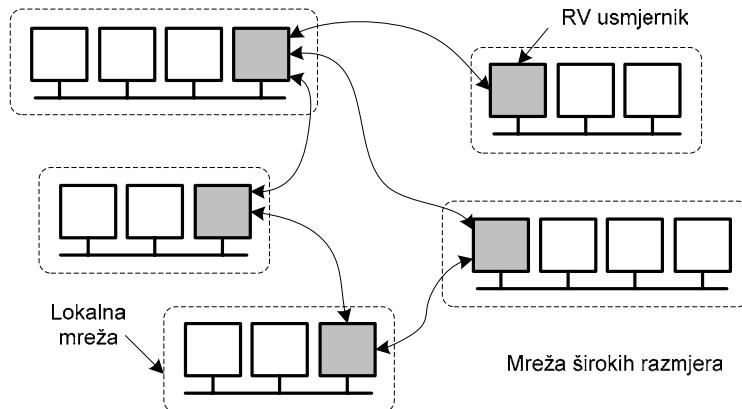
Prilikom objavljuvanja poruke, proces pošiljatelj ne navodi odredište poruke, umjesto toga postavlja poruci oznaku teme kojoj poruka pripada te predaje poruku komunikacijskom sustavu za dostavu poruke. S druge strane, primatelji poruka ne određuju od kojih procesa žele primati poruke nego obznanjuju sustavu koje ih teme zanimaju, odnosno preplaćuju se.



Slika 26: Objavljuvanje putem razašiljanja u TIB/Rendezvous sustavu

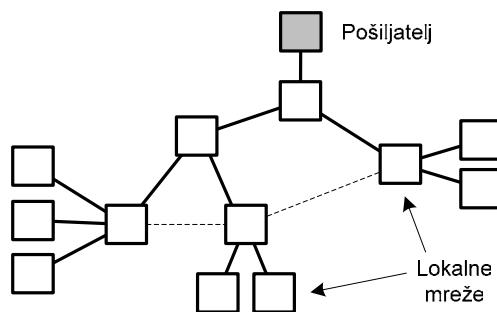
Arhitektura TIB/Rendezvous sustava je jednostavna. Osnovna funkcionalnost korištena u ostvarenju je višestruko razašiljanje koje je moguće ostvariti u mrežnom sloju pomoću protokola IP. Svaki sudionik u komunikaciji sadrži komponentu *RV komunikacijski posrednik* (engl. *rendezvous daemon*) čija je zadaća slanje poruka te filtriranje pristiglih poruka prema temi. Prilikom objavljuvanja poruke, RV posrednik razašilje poruku ostalim RV posrednicima u lokalnoj mreži. Na jednom računalu u mreži može se izvoditi više procesa korisnika koji objavljaju poruke i preplaćuju se na određene teme. Postupak preplaćivanja korisnik provodi obznanjivanjem preplate s oznakom teme lokalnom RV posredniku. Lokalni RV posrednik gradi tablicu sa zapisima oblika (*proces X, tema Y*), koji označavaju da je proces X pretplaćen na temu Y. Pri primanju poruke s temom Y, RV posrednik provjerava zapise u tablici te prosljeđuje poruku procesima pretplaćenim na temu Y, ako postoje takvi.

Kako bi se omogućilo korištenje sustava u mrežama širokih razmjera (WAN) koriste se *RV usmjernici* (engl. *rendezvous router daemons*). U svakoj lokalnoj mreži nalazi se po jedan RV usmjernik, koji komunicira s RV usmjernicima udaljenih lokalnih mreža. Nadalje, RV usmjernici tvore prekrivnu mrežu, u kojoj su spojeni *od-točke-do-točke* pomoću TCP spoja.



Slika 27: Cjelokupna arhitektura TIB/Rendezvous sustava širokih razmjera

Svaki usmjernik poznaje topologiju prekrivne mreže te izračunava stablo višestrukog razašiljanja (engl. *multicast tree*) koje služi za objavljivanje poruka udaljenim mrežama. Za razliku od ostvarenja u lokalnim mrežama, višestruko razašiljanje u mreži širokih razmjera ostvareno je u prijenosnom sloju mreže koristeći TCP protokol. Po objavljivanju poruke u lokalnoj mreži, RV usmjernik koji ju zastupa šalje poruku susjednim RV usmjernicima u stablu razašiljanja. Pri tome, poruka primljena od susjednog usmjernika prosljeđuje se dalje po stablu za razašiljanje kojemu je korijen usmjernik pošiljatelj poruke. Osim toga, poruka primljena od susjednog usmjernika razašilja se RV posrednicima u lokalnoj mreži.



Slika 28: Stablo razašiljanja poruka u mreži širokih razmjera

Nedostatak opisanog postupka razašiljanja je to što svaki usmjernik prima poruku bez obzira da li je ijedan korisnik u pripadnoj mreži pretplaćen na temu te poruke

4. Mreže zasnovane na sadržaju

Sustavi objava/preplata koriste se u mnogim raspodijeljenim sustavima u kojima korisnici imaju mogućnost izražavanja interesa za određene sadržaje. Neki od primjera su sustavi za praćenje poslovnih aktivnosti, sustavi za dojavljivanje potrošačima, mobilni alarmni sustavi, senzorske mreže, poosobljena dostava vijesti te sustavi za rasprostiranje informacija. Većina raspodijeljenih sustava koji koriste sustav objava/preplata zahtijevaju svojstvo razmjernega rasta i primjenjivost na mreže širokih razmjera kao što je Internet. Stoga su razvijeni objava/preplata sustavi s raspodijeljenom arhitekturom. Rasprostiranje događaja od objavitelja do preplatnika u raspodijeljenim objava/preplata sustavima ostvaruje se pomoću mehanizma usmjeravanja zasnovanog na sadržaju koji se danas brzo razvija.

Jednostavan sustav za dojavljivanje događaja koji ne pruža mogućnosti filtriranja sadržaja ostvaruje se pomoću mehanizma sličnim usmjeravanju u mrežama s višestrukim razašiljanjem. Postoje mnogi sustavi za dojavljivanje događaja sa svojstvom razmjernega rasta koji ne pružaju mogućnosti filtriranja prema sadržaju. Međutim, suvremeni raspodijeljeni sustavi velikih razmjera zahtijevaju napredne mogućnosti filtriranja događaja, odnosno sadržaja. U tom slučaju ukupna učinkovitost sustava te složenost usmjeravanja događaja ovisi o složenosti i razini izražajnosti jezika kojim se zadaju preplate i opisuju događaji. Zajedno s rastom izražajnosti jezika raste i složenost obrade i usmjeravanja događaja. Stoga su u praksi svojstvo razmjernega rasta i izražajnost dva oprečna svojstva između kojih je potrebno pronaći kompromis.

Većina raspodijeljenih posredničkih sustava zasnovanih na modelu objava/preplata koristi prekrivnu mrežu posrednika kojom se usmjeravaju događaji kroz mrežu. Složenost mehanizama usmjeravanja zasnovanih na sadržaju je velika jer mehanizmi sadrže brojne napredne funkcionalnosti. Budući da je napredne funkcionalnosti teško ostvariti u mrežnom ili prijenosnom sloju mreže, usmjeravanje zasnovano na sadržaju provodi se u primjenskom sloju mreže. [20]

Mreže zasnovane na sadržaju za potrebe adresiranja sudionika i usmjeravanja sadržaja ne oslanjaju se na uobičajene mrežne adrese sudionika. Odnosno, tok događaja od objavitelja do preplatnika u mreži određen je implicitnim svojstvima preplatnika, a ne eksplicitnom adresom preplatnika koji je primatelj događaja. Za određivanje primatelja sadržaja koriste se apstraktni i složeni mehanizmi. Na taj način *adresa* primatelja određena je preplatom, odnosno interesom preplatnika. Slično, adresa objavitelja je oglas koji opisuje događaje koje objavitelj generira. [9]

4.1. Oblikovanje mreže posrednika

Pri oblikovanju mreže posrednika (poslužitelja, zastupnika) koja pruža uslugu čija je arhitektura raspodijeljena postoje tri glavna problema [17]:

- **Topologija povezanosti:** kakva je konfiguracija mreže, te na koji način su sudionici povezani
- **Algoritmi usmjeravanja:** koje informacije sudionici trebaju razmjenjivati za osiguranje ispravne i učinkovite dostave poruka
- **Strategija obrade:** na kojim mjestima u mreži i prema kakvoj heuristici poruke trebaju biti obradivane s ciljem optimalnog korištenja mrežnog kapaciteta

Topologija povezanosti

Topologija povezanosti može se odabrati između tri široka razreda topologija. Najjednostavnija je centralizirana topologija, odnosno usluga ostvarena jednim centralnim posrednikom. Sljedeće moguće topologije su hijerarhijska topologija te najopćenitija, topologija ravnopravnih sudionika. Postojeći objava/preplata sustavi poput JEDI te TIB/Rendezvous koriste hijerarhijsku topologiju. Međutim, analiza provedena u [17] pokazuje da hijerarhijska topologija ima slabiju učinkovitost od topologije ravnopravnih sudionika. U sustavima SIENA [17] i Hermes [19] posrednici su organizirani u topologiju općenitog grafa, a mreža koju čine naziva se mreža ravnopravnih sudionika (engl. *peer to peer*, P2P). Također, moguće su i hibridne topologije koje kombiniraju hijerarhijsku i općenu topologiju. Primjerice, topologija hijerarhije grupa posrednika. U grupi su posrednici ravnopravni i spojeni su u općeniti graf. Grupe posrednika dalje se spajaju u hijerarhijsku topologiju. Uporabom hibridnih topologija moguće je ishoditi veću učinkovitost, no potrebno je *a priori* znanje o prirođenoj strukturi i primjeni programskog sustava koji koristi posrednički objava/preplata sustav. Pomoću *a priori* znanja moguće je na optimalan način razdijeliti mrežu u hijerarhije i grupe posrednika. No, posjedovanje *a priori* znanja o primjeni objava/preplata sustava narušava njegovu namjenu za općenite potrebe.

Mehanizmi usmjeravanja

Slično kao što pri oblikovanju topologije nije moguće pretpostaviti posjedovanje *a priori* znanja o primjeni sustava, tako za oblikovanje mehanizama usmjeravanja nije moguće pretpostaviti da objavitelji posjeduju ikakvo znanje o pretplatnicima i obratno. Stoga se problem usmjeravanja i adresiranja ne može jednostavno riješiti kao kod sustava za dostavu električke pošte. Nadalje, nije moguće pretpostaviti neki poseban razmještaj pretplatnika i objavitelja u mreži jer on ovisi o topologiji pa je mehanizme usmjeravanja potrebno prilagoditi topologiji. Međutim, može se iskoristiti razmještaj i struktura prometa događaja. Uzimajući u obzir navedena ograničenja, rješavanju problema

usmjeravanja može se pristupiti na tri alternativna načina. Svim načinima zajedničko je da zahtijevaju rasprostiranje određenih informacija svim posrednicima u mreži. Informacije koje se rasprostiru su događaji, pretplate ili oglasi. Rasprostiranje je nužno s obzirom da ne postoji *a priori* znanje o razmještaju posrednika i korisnika u mreži.

Prvo rješenje zasniva se na rasprostiranju svih događaja kroz mrežu. Stoga je nužno obavljanje filtriranja svih događaja na svakom pristupnom posredniku. Lokalnom pristupnom posredniku dospijevaju svi događaji koje on filtrira i dostavlja pretplatnicima spojenim na njega. Nedostatak ovog rješenja je velika količina mrežnog prometa prema pristupnim posrednicima jer primaju sve događaje. Također, s obzirom da se obrađuju svi događaji, pristupni posrednici su i vrlo procesno opterećeni. Drugo i treće alternativno rješenje koristi trenutno stanje sustava koje je određeno razmještajem i strukturonim pretplata ili oglasa u sustavu. Posebice, u drugom rješenju se rasprostranjuju pretplate svim posrednicima u sustavu. Širenjem pretplata kroz mrežu uspostavlja se najkraći put između pristupnog posrednika i svih ostalih posrednika. Po objavljivanju, događaji se usmjeravaju u suprotnom smjeru, odnosno od objavitelja prema pretplatniku i koristi se uspostavljeni najkraći put pa se na taj način događaji dostavljaju samo posrednicima koji sadrže odgovarajuće pretplate. Treće rješenje zasnovano je na rasprostiranju oglasa kroz mrežu. Oglas se rasprostire mrežom od objavitelja preko pristupnog posrednika do svih ostalih posrednika po razapinjućem stablu kojem je korijen pristupni posrednik koji je zaprimio oglas od objavitelja. Kada neki lokalni pristupni posrednik dobije pretplatu koja odgovara zaprimljenom oglasu, aktivira se put do posrednika koji je oglas odasiao. Događaji se šalju samo aktiviranim putovima, koji se nazivaju virtualne linije (engl. *virtual circuit*).

Drugo i treće rješenje uključuju trošak pohranjivanja svih pretplata i oglasa na svim posrednicima. Svako od rješenja prikladno je za određenu vrstu primjene sustava objava/pretplata. No pri oblikovanju sustava objava/pretplata opće namjene svako od rješenja usmjeravanja je podoptimalno za određene primjene.

Strategija obrade

Mehanizme usmjeravanja moguće je znatno poboljšati ako se na odgovarajući način rasporede zadaci filtriranja u mreži. Raspored zadatka filtriranja određuje cjelokupnu strategiju obrade pretplata u mreži. U praksi mnogo korisnika ima *slične* pretplate, odnosno neke pretplate se preklapaju jer su korisnicima od interesa slični događaji. Iz tog razloga razvijeni su mehanizmi koji iskorištavaju sličnosti među pretplatama, a koriste se za strategiju obrade pretplata u mreži. Mehanizmi iskorištavaju sličnosti među pretplatama tako što grupiraju njihove zajedničke podizraze. Primjerice, kada neki posrednik primi pretplatu, bilo od korisnika ili drugog posrednika, on ju prosljeđuje ako ona određuje nove događaje, odnosno događaje koji nisu u skupu događaja određenog već prije prosljeđenim pretplatama. Navedena strategija donosi više prednosti. Prvo, smanjuje se mreži promet ograničavanjem prosljeđivanja pretplata. Drugo, smanjuju se zahtjevi na kapacitet spremnika pretplata

pri posredniku. Treće, smanjivanjem broja pretplata koje spremi svaki posrednik smanjuje se i potrebno vrijeme za filtriranje događaja na posredniku.

Utjecaj izražajnosti jezika za preplaćivanje na oblikovanje mehanizma usmjeravanja

Svojstva jezika za preplaćivanje koja utječu na njegovu izražajnost značajno određuju oblikovanje mehanizama usmjeravanja. Dva su svojstva jezika za preplaćivanje koja utječu na ukupnu izražajnost jezika:

- **Doseg** preplatničkih predikata. Svojstvo se odnosi na uvid koji pretplata ima u sadržaj događaja. Primjerice, za događaje koji su strukturirani u obliku zapisa (engl. *recordlike*), doseg određuje koji se atributi mogu koristiti u zadavanju pretplata.
- **Snaga** preplatničkih predikata. Svojstvo koje određuje raznovrsnost i složenost operatora kojima se oblikuju preplatnički izrazi i uzorci događaja.

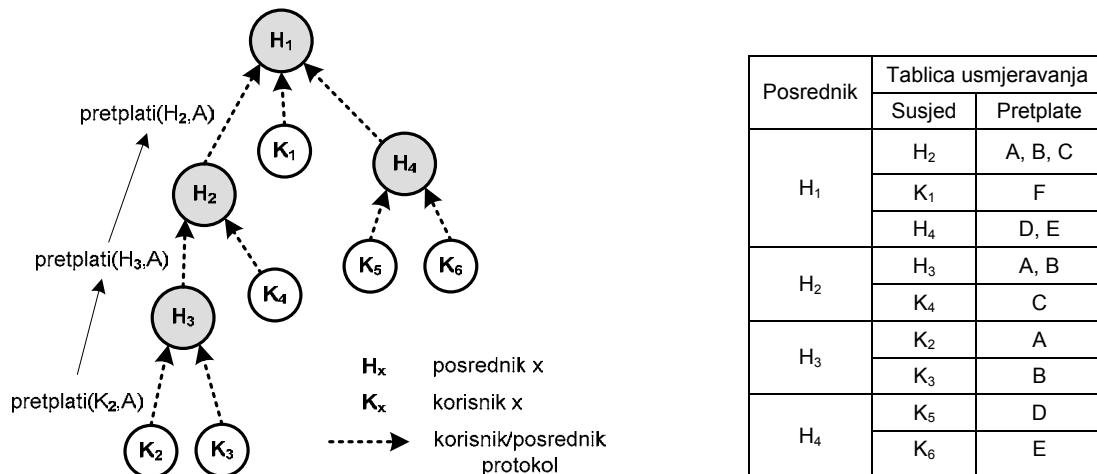
Jezik za oglašavanje također posjeduje svojstva dosega i snage, no iz perspektive objavitelja.

4.2. Arhitektura sustava

Topologija povezanosti i komunikacijski protokoli među posrednicima zajedno čine arhitekturu objava/preplata sustava. Protokol po kojem se odvija komunikacija između pojedinog posrednika objava/preplata sustava i korisnika naziva se objava/preplata protokol. Sačinjavaju ga sučelja posrednika koja omogućavaju preplaćivanje, objavljivanje i oglašavanje, te sučelje korisnika koje služi za dojavljivanje događaja. Protokol se još naziva korisnik/posrednik protokol čime se naglašava odvijanje komunikacije između korisnika i pristupnog posrednika. Nadalje, posrednici komuniciraju i međusobno kako bi zajednički raspodijelili zadatke obrade i dostave događaja na području cijele mreže. Komunikacija između dva posrednika odvija se po posrednik/posrednik protokolu. S obzirom da je arhitektura ostvarena pomoću prekrivne mreže, posrednici nisu nužno spojeni izravnom fizičkom vezom ili stalnom vezom poput TCP/IP spoja. Protokol posrednik/posrednik također je moguće ostvariti pomoću standardnih protokola primjenskog sloja kao što su HTTP ili SMTP. Svaki posrednik komunicira s nekolicinom susjednih posrednika i korisnika po posrednik/posrednik i korisnik/posrednik protokolima.

4.2.1. Hijerarhijska korisnik/posrednik arhitektura

Primjer hijerarhijske topologije prikazan je na slici 29. Dva susjedna posrednika komuniciraju na asimetričan način pomoću korisnik/posrednik protokola, odnosno jedan od posrednika se ponaša kao korisnik. Stoga se hijerarhijska topologija prikazuje usmjerenim grafom, a arhitektura se naziva hijerarhijska korisnik/posrednik arhitektura. Posrednik može imati proizvoljan broj ulaznih veza s *posrednicima korisnicima*, no samo jednu izlaznu vezu s *posrednikom gospodarom*. Posrednik koji nema *posrednika gospodara* naziva se *korijenski posrednik*. Hijerarhijska arhitektura je neposredno proširenje centralizirane arhitekture. Osnovni centralni posrednik se preinačuje tako da prosljeđuje sve poruke (pretplate i događaji) prema *posredniku gospodaru*. Iz tog razloga posrednik/posrednik protokol u hijerarhijskoj arhitekturi je jednak korisnik/posrednik protokolu između korisnika i posrednika. Posrednici spojeni na posrednika gospodara njemu se doimaju kao vanjski korisnici, odnosno posrednik gospodar od spojenih posrednika prima pretplate i događaje, a posrednik gospodar njima šalje događaje. Svaki posrednik je pretplaćen kod posrednika gospodara sa svim pretplata koje se nalaze u podmreži ispod njega. Stoga gospodar posrednik omogućuje komunikaciju između podmreža spojenih na njega. Ako gospodar posrednik dobije događaj iz jedne podmreže onda on usmjerava događaj u sve ostale podmreže pomoću tablice usmjeravanja. Svi događaji prosljeđuju se sve do korijena hijerarhije.

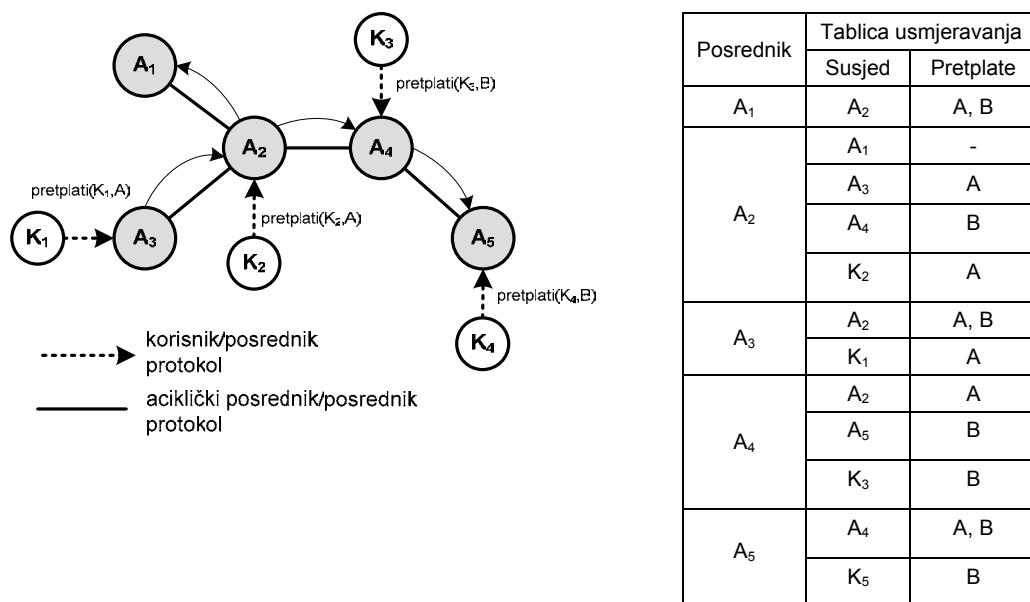


Slika 29: Primjer hijerarhijske korisnik/posrednik arhitekture

U tablici usmjeravanja u prikazanom primjeru, pretplate pri korijenskom posredniku H₁ čine skup svih pretplata u sustavu. Osnovni nedostatak hijerarhijske arhitekture je mogućnost preopterećenja posrednika smještenih visoko u hijerarhiji jer dobivaju sve događaje objavljene ispod njih u hijerarhiji. Pri tome, svaki posrednik je kritični čvor i može uzrokovati kvar većeg dijela mreže. Ispad nekog posrednika odspaja iz mreže sve podmreže koje se nalaze ispod njega.

4.2.2. Aciklična arhitektura ravnopravnih sudionika

U acikličkoj arhitekturi ravnopravnih sudionika, posrednici komuniciraju simetrično i stoga su oni ravnopravni. Također, zahtijeva se da konfiguracija veza među posrednicima čini aciklički neusmjeren graf. Komunikacija se odvija po protokolu koji dopušta dvosmjeran tok pretplate, oglasa i dojava, a protokol se naziva aciklički posrednik/posrednik protokol. Kao i kod hijerarhijske arhitekture, vanjski korisnici komuniciraju s pristupnim posrednicima pomoću korisnik/posrednik protokola. Na slici 30 prikazan je primjer acikličke arhitekture ravnopravnih sudionika. Postupci konfiguriranja veza među posrednicima koordiniraju posrednike da bi se održalo svojstvo acikličnosti topologije. Naime, algoritmi usmjeravanja oslanjaju se na to svojstvo i podrazumijevaju da postoji samo jedan put između svaka dva posrednika. Međutim, acikličnost nije jednostavno osigurati pa osiguranje acikličnosti zahtijeva zнатne troškove ako je sustav širokih razmjera, decentralizirane i autonomne administracije.



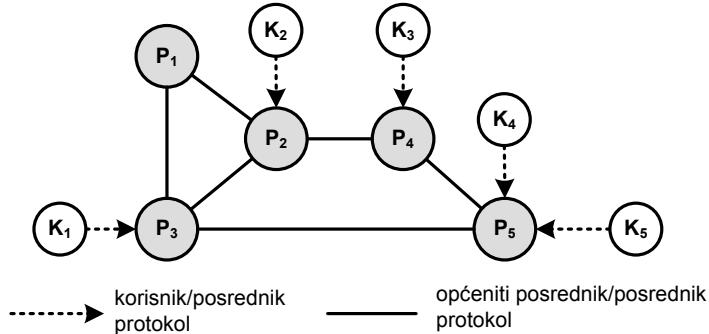
Slika 30: Primjer acikličke arhitekture ravnopravnih sudionika

Slično kao u hijerarhijskoj arhitekturi, zbog nedostatka zalihosti u topologiji ograničeno je osiguranje stalne povezanosti mreže. Npr. ispad posrednika A izolira sve podmreže koje su izravno spojene na njega.

4.2.3. Općenita arhitektura ravnopravnih sudionika

Općenita arhitektura ravnopravnih sudionika dobiva se uklanjanjem zahtjeva acikličnosti iz aciklične arhitekture posrednika. Jednako kao aciklična arhitektura, općenita arhitektura omogućava dvosmernu komunikaciju između dva posrednika, no topologija tvori općeniti neusmjeren graf, koji može

sadržavati više puteva između dva posrednika. Prednost općenite arhitekture nad prije navedenim arhitekturama je da zahtijeva manje koordinacije i pruža veću prilagodljivost konfiguracije veza među posrednicima.



Slika 31: Primjer općenite arhitekture ravnopravnih sudionika

Nadalje, postojanje zalihosnih veza arhitekturu čini otpornijom na ispadanje pojedinih posrednika. Loša strana zalihosnih veza je potreba ostvarenja složenijih algoritama usmjeravanja koji izbjegavaju cikluse i odabiru optimalne putove. Porukama se dodaje *time-to-live* brojač, a putovi se uspostavljaju izgradnjom minimalnog razapinjućeg stabla. Stoga je posrednik/posrednik protokol u općenitoj arhitekturi ravnopravnih sudionika potrebno prilagoditi uključivanjem dodatnih informacija.

4.3. Algoritmi usmjeravanja i strategije obrade događaja

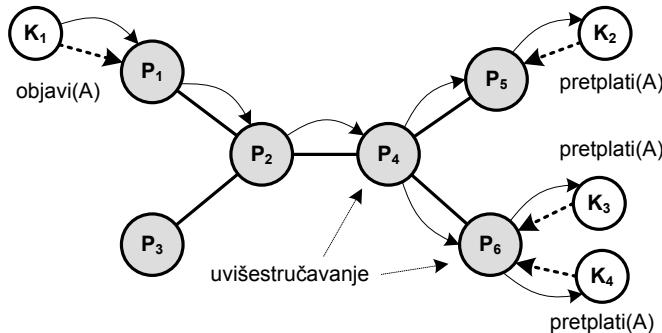
Ovisno o topologiji, posrednici uspostavljaju odgovarajuće puteve za usmjeravanje događaja kako bi se osiguralo ispravno dostavljanje objavljenih dojava svim preplaćenim korisnicima. Stoga je potrebno da se dojave i preplate *sastanu* negdje u mreži. Na mjestu *sastanka* događaji se filtriraju i usmjeravaju prema preplatnicima. Navedeno načelo osnova je oblikovanja većine algoritama usmjeravanja.

4.3.1. Načela usmjeravanja

Razvoj učinkovitih algoritama usmjeravanja zasniva se na načelima preuzetim iz algoritama usmjeravanja korištenim pri višestrukom razašiljanju u mrežnom sloju. Ideja strategije usmjeravanja je slanje dojava samo prema posrednicima koji imaju zainteresirane korisnike. Pri tome se događaji nastoje usmjeravati najkraćim putem. Dva općenita načela korištena su za oblikovanje algoritama usmjeravanja: načelo nizvodnog uvišestručavanja i načelo uzvodnog ispitivanja.

Načelo nizvodnog uvišestručavanja

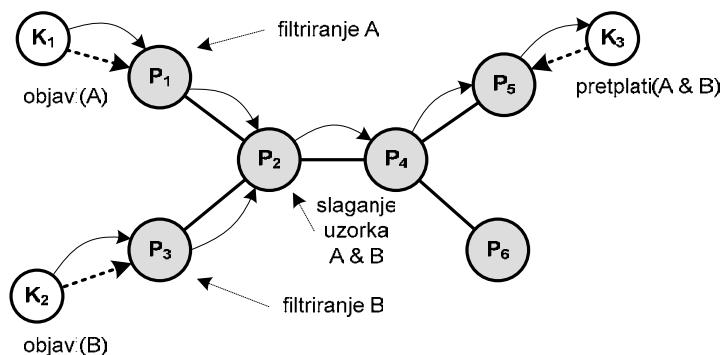
Načelo nizvodnog uvišestručavanja (engl. *downstream replication*): izvorni događaj usmjerava se kao jedna preslika najdalje moguće, a uvišestručava se samo nizvodno, odnosno što bliže preplatnicima koji su preplaćeni na događaj, a što dalje od izvořita događaja. U primjeru na slici 32 posrednici P_4 i P_6 udvostručavaju događaj A.



Slika 32: Primjer načela nizvodnog uvišestručavanja

Načelo uzvodnog ispitivanja

Načelo uzvodnog ispitivanja (engl. *upstream evaluation*): filtri se primjenjuju, a uzorci događaja slazu (engl. *pattern assembling*) uzvodno, odnosno što bliže izvorima dojava. Korisnik K_3 u primjeru na slici 33 preplaćen je na uzorak događaja $A \& B$. Filtriranje se obavlja uzvodno, dakle posrednici P_1 i P_3 obavljaju filtriranje događaja A i B, a posrednik P_2 ispituje mogućnost slaganja uzorka događaja A & B te prosljeđuje uzorak susjednom posredniku.



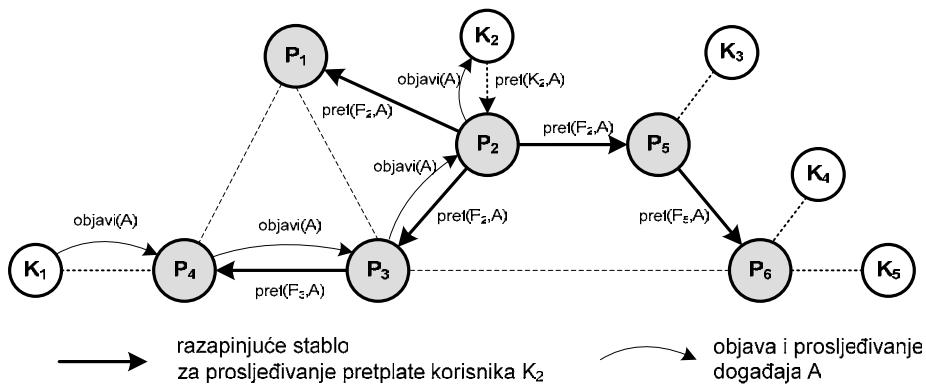
Slika 33: Primjer načela uzvodnog ispitivanja

4.3.2. Razredi algoritama usmjerenja

Navedena načela koriste se za oblikovanje dva općenita razreda algoritama usmjerenja. Prvi razred algoritama zasniva se na proslijedivanju pretplata, a drugi razred algoritama zasniva se na proslijedivanju oglasa.

Algoritmi zasnovani na proslijedivanju pretplata

Algoritmi zasnovani na proslijedivanju pretplata (engl. *subscription forwarding*) ne koriste oglašavanje, putevi usmjerenja dojava postavljaju se samo na osnovi pretplata. Pretplate se rasprostranjuju kroz cijelu mrežu čime se gradi stablo koje spaja pretplatnika sa svim posrednicima u mreži. Prilikom objave događaja na nekom mjestu u mreži koji zadovoljava pretplatu, događaj se usmjerava prema pretplatniku u suprotnom smjeru po stablu koje je izgradeno rasprostiranjem pretplate.



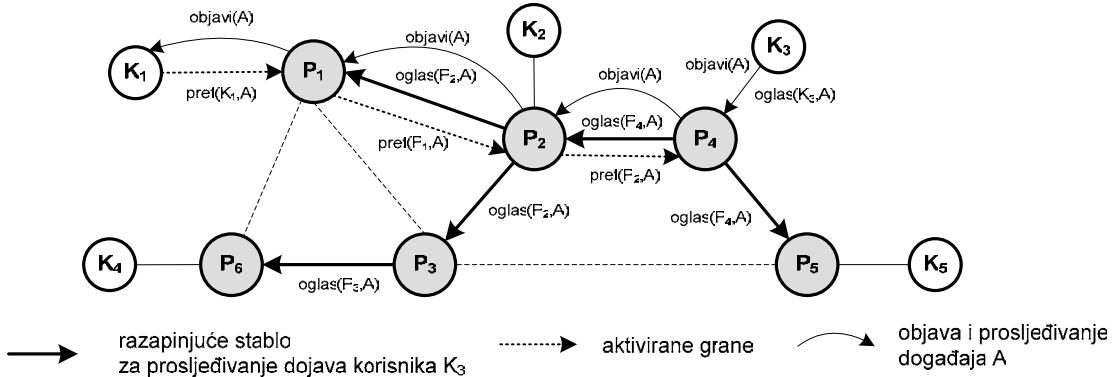
Slika 34: Primjer proslijedivanja pretplata

Slika 34 prikazuje primjer rada algoritma usmjerenja proslijedivanjem pretplata. Korisnik K_2 preplaćuje se na događaj A pri posredniku P_2 . Pretplata se širi do svih posrednika i gradi se stablo kao što je prikazano na slici. Nakon što korisnik K_1 objavi događaj A on se usmjerava po razapinjućem stablu u smjeru suprotnom rasprostiranju pretplate, odnosno prema posredniku P_2 . Posrednik P_4 usmjerava događaj prema posredniku P_3 , ovaj prema posredniku P_2 . Konačno, posrednik P_2 usmjerava događaj A korisniku K_2 .

Algoritmi zasnovani na proslijedivanju oglasa

Algoritmi zasnovani na proslijedivanju oglasa (engl. *advertisement forwarding*) osiguravaju slanje pretplata samo prema korisnicima koji objavljuju relevantne događaje za te pretplate. Putevi za slanje pretplata uspostavljaju se izgradnjom stabla rasprostiranjem oglasa. Novonastali oglasi rasprostire se mrežom od pristupnog posrednika do svih ostalih posrednika po razapinjućem stablu kojem je korijen lokalni pristupni posrednik koji je zaprimio oglas. Kada pristupni posrednik dobije pretplatu koja

odgovara nekom od zaprimljenih oglasa on ju rasprostranjuje u suprotnom smjeru te tako aktivira put do posrednika koji je oglas odaslao. Pri objavi, događaji se šalju samo aktiviranim putevima, koji se nazivaju virtualne linije (engl. *virtual circuit*).



Slika 35: Primjer prosljeđivanja oglasa

Na slici 35 prikazan je primjer rada algoritma usmjeravanja prosljeđivanjem oglasa. Korisnik K₃ najprije oglašava događaj A koji se rasprostire mrežom čime se gradi razapinjuće stablo. Potom korisnik K₁ objavljuje pretplatu na događaj A. Pretplata se prosljeđuje samo po granama razapinjućeg stabla. Pretplata se usmjerava preko posrednika P₁, P₂ i P₄, aktivirajući put prema korisniku koji je oglasio odgovarajući događaj A. Kada korisnik K₃ objavi događaj A on se usmjerava prema pretplatnicima prethodno aktiviranim putevima. U opisanom slučaju događaj A se usmjerava prema korisniku K₁ preko posrednika i P₄, P₂ i P₁.

4.3.3. Algoritmi i podatkovne strukture

Algoritmi usmjeravanja nadograđuju se radi poboljšanja učinkovitosti smanjivanja komunikacije među sudionicima, količine pohranjenih podataka te troškova obrade i izračunavanja. Poboljšanje se postiže pomoću *postupka podrezivanja* razapinjućih stabala koja nastaju rasprostiranjem pretplata i oglasa. Algoritam prosljeđivanja pretplata rasprostranjuje pretplate do svih posrednika po razapinjućem stablu kojem je korijen pretplatnik. Novodošlu pretplatu posrednik ne prosljeđuje susjednim posrednicima ako je već proslijedena općenitija pretplata koja pokriva novodošlu pretplatu. Stoga se dobiva učinak podrezivanja razapinjućeg stabla, odnosno ograničava se širenje stabla jer se pretplata prosljeđuje samo ako definira nove događaje. Prosljeđivanje oglasa odvija se prema sličnom načelu. U hijerarhijskoj arhitekturi algoritmi usmjeravanja su jednostavniji jer se pretplate ne prosljeđuju po razapinjućem stablu, nego po jedinstvenom putu prema vrhu hijerarhije.

4.3.3.1 Formalni opis podatkovnih struktura, relacija i algoritama

Dojava se definira kao skup atributa, a svaki pojedini atribut zadaje se *tipom podatka, imenom* i *vrijednošću*. Imena atributa su znakovni nizovi. Podatkovni tipovi atributa pripadaju unaprijed definiranom skupu koji se sastoji od standardnih jednostavnih podatkovnih tipova programskih jezika i baza podataka. Za jednostavne podatkovne tipove definiran je i skup operatora. Atribut α se zapisuje kao uredena trojka $\alpha = (tip_\alpha, ime_\alpha, vrijednost_\alpha)$.

Preplata ili filter je skup ograničenja nad vrijednostima atributa. Pojedino ograničenje sastoji se od tipa atributa, imena atributa, binarnog operatora i vrijednosti. Ograničenje ϕ zadaje se kao uređena četvorka $\phi = (tip_\phi, ime_\phi, operator_\phi, vrijednost_\phi)$. Osim osnovnih relacijskih operatora, koriste se operatori prefiks ($>*$) i sufiks ($*<$) nad znakovnim nizovima. Također, koristi se poseban znak $*$ za označavanje da pripadni atribut može poprimiti bilo koju vrijednost.

Atribut α zadovoljava ograničenje ϕ ako i samo ako:

$$(tip_\alpha = tip_\phi) \wedge (ime_\alpha = ime_\phi) \wedge operator_\phi(vrijednost_\alpha, vrijednost_\phi).$$

Oznakom $\alpha \prec \phi$ označava se da atribut α zadovoljava ograničenje ϕ . Zadovoljenje ograničenja još se naziva i relacija *pokrivanja*, odnosno kaže se da ograničenje ϕ pokriva atribut α (ϕ pokriva α).

Preplata ili filter je skup ograničenja među kojima vrijedi konjunkcija, odnosno sva moraju biti zadovoljena. Kaže se da dojava n zadovoljava filter f , ili istovjetno, f pokriva n . Oznaka je $n \prec_s^N f$.

Filter f pokriva dojavu n , odnosno $n \prec_s^N f$, ako i samo ako za svako ograničenje ϕ iz f postoji atribut α iz n , tako da vrijedi $\alpha \prec \phi$.

$$n \prec_s^N f \Leftrightarrow \forall \phi \in f : \exists \alpha \in n : \alpha \prec \phi$$

dojava		preplata		
<i>znakovni niz</i>	opis = „opterećenje poslužitelja“	vrijedi	<i>znakovni niz</i>	opis $>*$ „opterećenje“
<i>vrijeme</i>	trenutak = 05:01:03	\prec_s^N	<i>cijeli broj</i>	razina > 2
<i>cijeli broj</i>	razina = 3			
<i>znakovni niz</i>	opis = „opterećenje poslužitelja“	ne vrijedi	<i>znakovni niz</i>	opis $>*$ „opterećenje“
<i>cijeli broj</i>	razina = 10	\prec_s^N	<i>cijeli broj</i>	razina > 2
			<i>cijeli broj</i>	razina < 7

Tablica 5: Primjeri relacije \prec_s^N

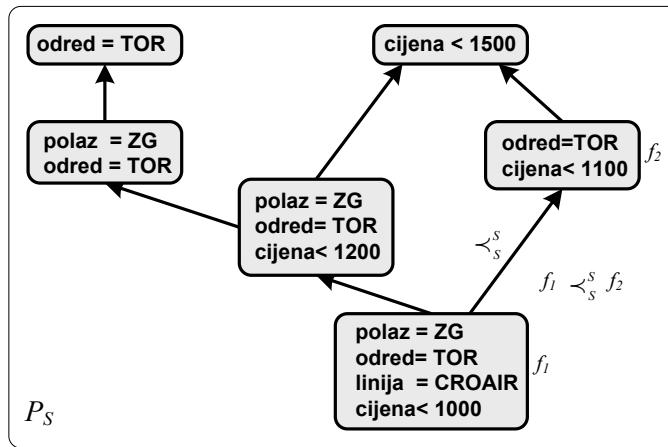
Tablica 5 prikazuje primjere relacije \prec_s^N . U prvom primjeru preplata pokriva dojavu jer su oba ograničenja zadovoljena, dok u drugom primjeru preplata ne pokriva dojavu jer ograničenje „razina < 7 “ nije zadovoljeno zbog atributa „razina = 10“.

Ako za preplate f_1 i f_2 vrijedi da preplata f_2 određuje podskup dojava koje određuje preplata f_1 , onda preplata f_1 pokriva preplatu f_2 . Odnosno, ako vrijedi da svaka dojava n koja je pokrivena s preplatom f_2 ujedno pokrivena i sa preplatom f_1 :

$$f_2 \prec_s^S f_1 \Leftrightarrow \forall n : n \prec_s^N f_2 \Rightarrow n \prec_s^N f_1$$

Djelomično uređen skup preplata P_S

U svrhe pohranjivanja postojećih preplata, njihovih izvorišta te odredišta na koja su proslijedena, posrednici sadrže podatkovnu strukturu koja je zajednička za različite algoritme i topologije. Također, podatkovna struktura služi za određivanje veza među preplatama. Podatkovna struktura predstavlja djelomično uređen skup preplata, odnosno filtra. Skup je djelomično uređen ako relacija uređenja ne vrijedi za svaka dva člana skupa. Djelomično uređenje definira se relacijom pokrivanja \prec_s^S za preplate, a relacijom pokrivanja \prec_A^A za oglase. Podatkovna struktura koja pohranjuje preplate označava se sa P_S . Dijagrama skupa P_S je grafički prikaz skupa P_S te prikazuje samo neposredne relacije pokrivanja, koje su predstavljene vezama u stablu između para filtra kao što je prikazano u primjeru na slici 36. U memorijskom zapisu podatkovne strukture također se pohranjuju samo neposredne relacije. U skupu P_S prema relaciji \prec_s^S , filter f_1 je izravni prethodnik filtra f_2 , a f_2 je izravni sljedbenik f_1 ako i samo ako $f_1 \prec_s^S f_2$ i ne postoji filter f_3 u P_S takav da vrijedi $f_1 \prec_s^S f_3 \prec_s^S f_2$. Filtri najviše razine, koji se nazivaju *korjeni*, jesu oni koji nemaju sljedbenika.



Slika 36: Primjer dijagrama djelomično uređenog skupa preplata P_S

Slika 36 prikazuje primjer djelomično uređenog skupa preplata P_S . Preplata na dnu skupa označena je sa f_1 i izravni je prethodnik preplate f_2 , jer preplata f_2 širi skup dojava. Preplate „*odred = TOR*“ i „*cijena < 1500*“ korjeni su skupa.

Prilikom dodavanja novog filtra f u skup P_S moguća su tri slučaja koja su značajna za algoritam proslijedivanja preplata:

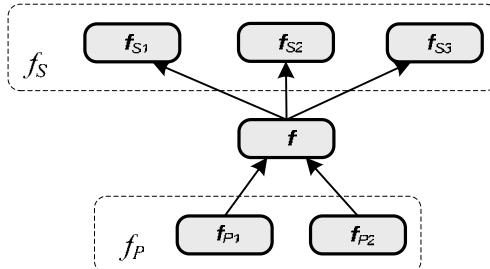
- filter f se dodaje u skup P_S kao korijen

- filtr f već postoji u skupu P_S
- filtr f se ubacuje na određeno mjesto u skup P_S i ima neprazan skup sljedbenika

Izgradnja skupa P_S u hijerarhijskoj arhitekturi

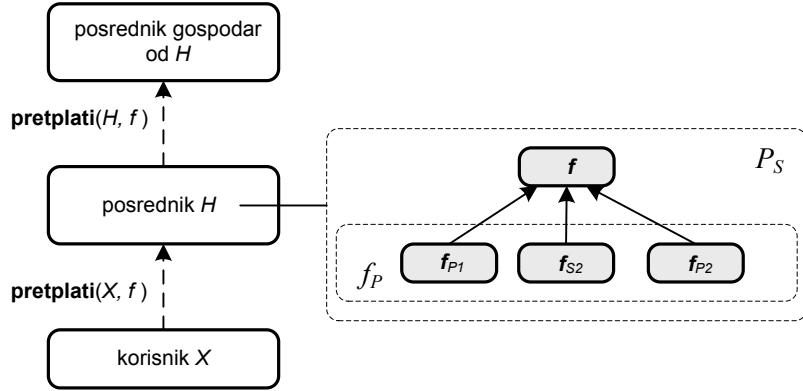
Hijerarhijski posrednik H pohranjuje pretplate u djelomično uređenom skupu pretplata P_S . Dodatno, svaka pretplata iz skupa P_S ima pridružen skup $\text{preplatnici}(f)$ koji sadrži nazive pretplatnika na pripadnu pretplatu.

Posrednik po primitku poruke $\text{preplati}(X, f)$ koja određuje pretplatu f od korisnika X pokreće algoritam koji obilazi skup P_S , počevši iz svih korijenskih pretplata. Obilaženjem skupa traži se filtr f' koji pokriva novi filtr f te sadrži X u skupu pretplatnika: $f \prec_s^s f' \wedge X \in \text{preplatnici}(f')$. Ako je pronađen takav filtr f' u P_S onda pretraživanje prestaje i ne poduzima se nikakva akcija jer je korisnik X već pretplaćen s općenitijom pretplatom f' . Ako se ne pronađe filtr f' tada pretraživanje završava određivanjem dva skupa: skupa izravnih sljedbenika f_S te skupa izravnih prethodnika f_P . Ako je $f_S = f_P = \{f\}$, odnosno ako filtr f već postoji u P_S , onda algoritam umeće X u skup $\text{preplatnici}(f)$. U suprotnom, f se umeće u skup P_S između filtara skupa f_S i filtara skupa f_P . Oznaka pretplatnika X umeće se u skup $\text{preplatnici}(f)$. U primjeru na slici 37 prikazano je umetanje filtra f između skupova f_S i f_P .



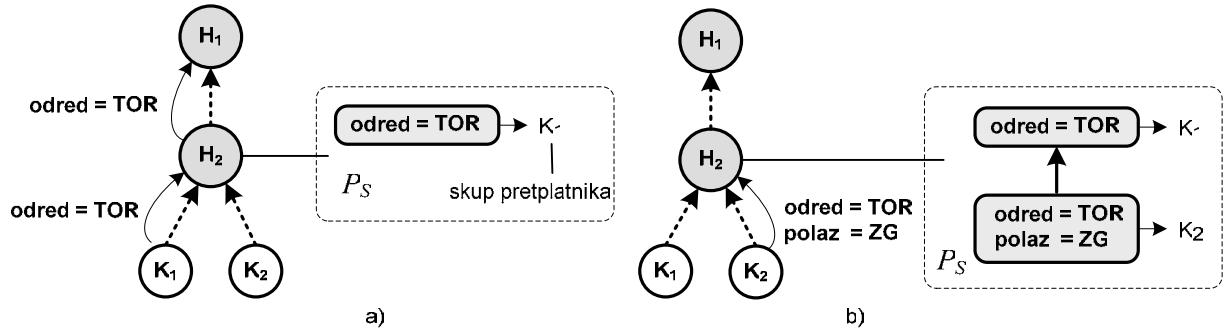
Slika 37: Ubacivanje pretplate f između skupova f_S i f_P

Ako je skup $f_S = \emptyset$ onda se pretplata f umeće kao korijenska pretplata, a posrednik H prosljeđuje f svom posredniku gospodaru, odnosno šalje gospodaru $\text{preplati}(H, f)$ kao što je prikazano na slici 38. Jedino korijenski filtri uzrokuju mrežni promet, jer se oni prosljeđuju posredniku gospodaru. Stoga *oblik* dijagrama skupa P_S grubo održava razinu učinkovitosti strategije obrade pretplata.



Slika 38: Ubacivanje korijenske pretplate f i proslijedjivanje pretplate posredniku gospodaru

Ako je $f_P \neq \emptyset$ tada algoritam uklanja X iz skupova $preplatnici(f'')$ svih pretplata f'' koje su pokrivenе sa f . Postupak uklanjanja obavlja se obilaskom skupa P_S u širinu počevši od pretplate u skupu f_P . Kada se pri pretraživanju u skupu $preplatnici(f'')$ neke pretplate f'' pronađe X , on se uklanja iz skupa, a pretraživanje se ne širi dalje iz pretplate f'' . U postupku uklanjanja pretplatnika X moguće je da skup $preplatnici(f'')$ neke pretplate f'' postane prazan, u tom slučaju takva se pretplata uklanja iz skupa P_S .



Slika 39: Primjer scenarija preplaćivanja u hijerarhijskoj arhitekturi

U primjeru na slici 39. prikazan je jednostavan scenarij preplaćivanja, proslijedjivanja pretplate i izgradnje skupa P_S . Primjer prikazuje jednostavnu hijerarhijsku arhitekturu koja sadrži posrednik H_2 na koji su spojena dva korisnika K_1 i K_2 . Nadalje, posrednik H_2 spojen je na svojeg posrednika gospodara H_1 . Posrednik H_2 prima i obrađuje pretplatu „ $odred = TOR$ “ pristiglu od korisnika K_1 . Na početku je skup P_S prazan pa se pristigla pretplata umeće kao korijen skupa te proslijeduje posredniku gospodaru H_1 . Postupak je prikazan na dijagramu (a). Sljedeća pretplata koju posrednik H_1 prima jest pretplata „ $odred = TOR, polaz=ZG$ “ od korisnika K_2 . S obzirom da novu pretplatu pokriva već proslijedena korijenska pretplata, posrednik H_2 ne proslijedi novu pretplatu.

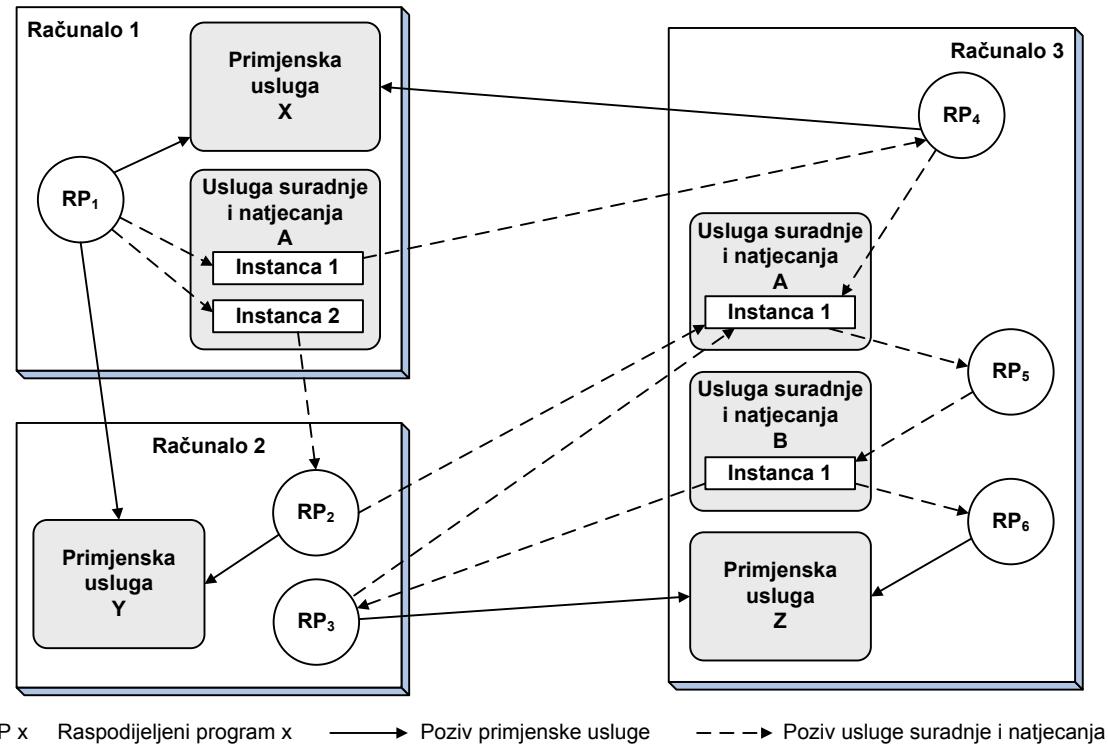
Kada posrednik zaprimi dojavu n , pokreće se postupak pretraživanja pretplate iz skupa P_S koje n zadovoljava, odnosno onih za koje vrijedi $n \prec_s^N s$. Postupak pretražuje obilaskom u širinu i započinje sa korijenskim pretplatama koje se stavljuju u rep Q . Sljedeće, postupak prolazi kroz rep Q i ispituje svaki član s . Ako za dojavu n i član s vrijedi $n \prec_s^N s$, onda se dodaju u rep svi izravni prethodnici od s koji već nisu obrađeni. U suprotnom, ako ne vrijedi $n \prec_s^N s$, pretplata s uklanja se iz repa. Po okončanju

postupka, Q sadrži sve preplate koje pokrivaju n . Posrednik šalje presliku dojave n svim preplatnicima preplata u skupu Q . Neovisno o ishodu pronalaženja pripadnih preplata, ako posrednik ima posrednika gospodara i on nije poslao dojavu n onda se njemu prosljeđuje dojava n .

5. Okolina raspodijeljenog sustava zasnovanog na uslugama

U sustavima zasnovanim na uslugama *programiranje na veliko* ostvaruje se pomoću kompozicije usluga. Postupak kompozicije usluga zasniva se na povezivanju i spajanju neovisnih usluga s ciljem ostvarivanja raspodijeljenih programskih sustava ili stvaranja novih, kompozitnih usluga. Postoje jezici posebne namjene poput WS-CDL, OWL-S i WS-BPEL jezika, koji služe za opis postupka kompozicije usluga. Najčešće korišteni je WS-BPEL kojim se opisuje postupak kompozicije usluga zasnovan na procesima. Razvijeni su mnogi sustavi za kompoziciju usluga od strane vodećih tvrtki proizvođača programskih sustava kao što su Microsoft, IBM i Oracle. Međutim, svi sustavi za kompoziciju usluga zasnovani su na centralnom poslužitelju koji upravlja i nadzire izvođenje raspodijeljenog programskog sustava. Nedostatak korisnik/poslužitelj arhitekture tih sustava je ograničenje razmjerne rasta sustava pa ih je teško primijeniti za izgradnju složenih raspodijeljenih sustava.

Kako bi se omogućila kompozicija usluga s raspodijeljenim upravljanjem, u okviru projekta *CRO-GRID Posrednički sustavi* razvijena je raspodijeljena arhitektura sustava nazvana *raspodijeljeni sustav zasnovan na suradnji i natjecanju* (engl. *Coopetition-based distributed system*, CBDS) [7]. Sustav CBDS izvodi skup raspodijeljenih programa koji pozivaju, koordiniraju i sinkroniziraju usluge te na taj način omogućuju njihovu suradnju i natjecanje. Arhitektura sustava CBDS je proširiva i prilagodljiva (engl. *open-ended*), zasnovana na uslugama, te je poboljšanje raspodijeljene arhitekture zasnovane na koordinaciji. Korištenje zasebnih mehanizama suradnje i natjecanja u arhitekturi sustava CBDS poboljšanje je u odnosu na postojeće sustave s raspodijeljenom arhitekturom zasnovanom na koordinaciji. Osnovni koncepti CBDS-a jesu odvajanje i međusobna neovisnost logike primjenskog programa, koordinacijske logike te mehanizama suradnje i natjecanja. Sve funkcionalnosti primjenskog programa, primjerice logika za izračunavanje, ostvarene su kao zasebne samostojeće (engl. *stand-alone*) usluge. Skup raspodijeljenih programa povezuje i sinkronizira individualne usluge. Raspodijeljeni programi ostvaruju logiku za koordinaciju usluga zapisanu pomoću jezika za opisivanje procesa izведенog iz WS-BPEL jezika. Koordiniranje aktivnosti u raspodijeljenom sustavu izvodi se tako što raspodijeljeni programi pozivaju primjenske usluge te koriste zasebne mehanizme suradnje i natjecanja.



Slika 40: Primjer raspodijeljenog sustava zasnovanog na suradnji i natjecanju

Mehanizmi suradnje i natjecanja pružaju općenite mehanizme za sinkronizaciju, komunikaciju, suradnju i natjecanje raspodijeljenih usluga. Nadalje, mehanizmi suradnje i natjecanja ostvareni su kao usluge s održavanjem stanja. Na slici 40 prikazan je primjer raspodijeljenog primjenskog programa koji se izvodi u CBDS sustavu. Složeni raspodijeljeni primjenski program raščlanjen je na skup primjenskih usluga i skup raspodijeljenih programa. Primjenska usluga X postavljena je na računalu 1, primjenska usluga Y postavljena je na računalu 2, a primjenska usluga Z postavljena je na računalu 3. Na navedenim računalima izvode se i raspodijeljeni programi RP₁, RP₂, RP₃, RP₄, RP₅ i RP₆. U svrhe sinkronizacije i koordinacije, mehanizmi suradnje i natjecanja postavljeni su na računalo 1 i računalo 3. Budući da su mehanizmi suradnje i natjecanja ostvareni kao mrežne usluge s održavanjem stanja, moguće je postojanje više instanci određenog mehanizma na jednom računalu. Dva različita mehanizma suradnje i natjecanja A i B postavljena su na računalu 3 i od svakog je stvorena po jedna instance. S druge strane, na računalu 1 postoje višestruke instance istog mehanizma suradnje i natjecanja (instance 1 i 2). Raspodijeljeni programi pozivaju primjenske usluge te instance mehanizama suradnje i natjecanja kao što je prikazano u primjeru.

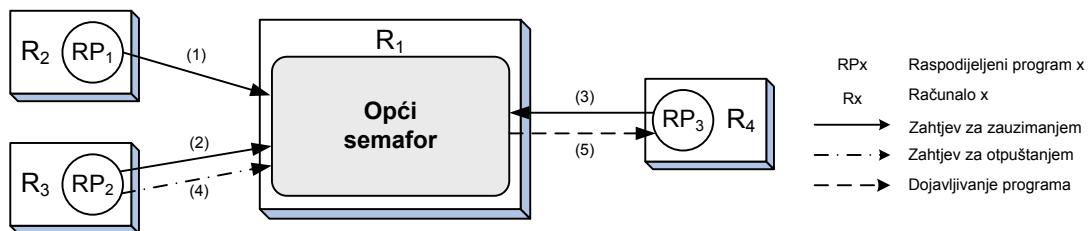
5.1. Mehanizmi suradnje i natjecanja

Raspodijeljeni sustav zasnovan na suradnji i natjecanju oslanja se na općenite mehanizme koji omogućuju suradnju i natjecanje usluga. Stoga je razvijen skup mehanizama suradnje i natjecanja koji su ostvareni kao usluge s održavanjem stanja. Skup mehanizama čine binarni semafor (engl. *binary*

semaphore), opći semafor (engl. *counting semaphore*), spremnik poruka (engl. *mailbox*) te usmjernik događaja (engl. *event channel*). Ostvarenje mehanizama koristi *Web services*, *WS-Resource Framework* i *WS-Addressing* standarde. Stoga svaka platforma koja podržava WS-* skup standarda može koristiti ostvarene mehanizme. Svaki od mehanizama ostvaren je kao *WS-Resource* pa se stoga na jednom računalu može stvoriti više instanci istog mehanizma. Svaka usluga pruža dva sučelja, sučelje tvornice koje služi za stvaranje i uništavanje instanci, te sučelje za pristup funkcionalnostima mehanizma. U nastavku poglavlja opisani su semafori i spremnik poruka, dok je u poglavlju 6 detaljno opisan usmjernik događaja.

5.1.1. Binarni i opći semafor

Binarni i opći semafor koriste se za sinkronizaciju raspodijeljenih programa te upravljanje pristupom sredstvima u raspodijeljenom sustavu. Binarni semafor omogućuje međusobno isključivanje raspodijeljenih programa tako što dozvoljava samo jednom raspodijeljenom programu da ga zauzme u određenom trenutku. Opći semafor proširuje funkcionalnost binarnog semafora tako što omogućava da ga zauzme konačan broj raspodijeljenih programa u isto vrijeme.

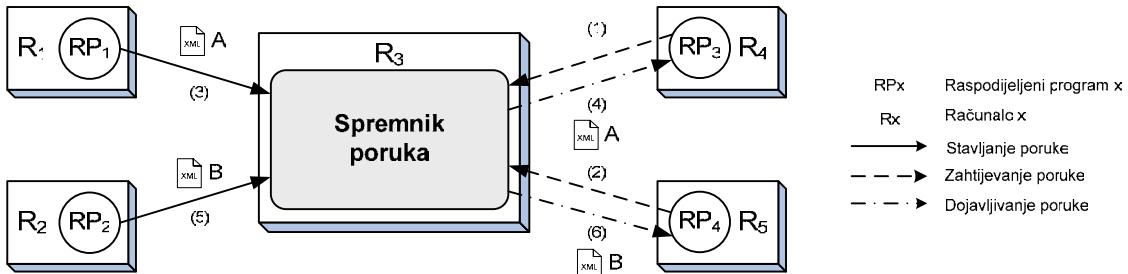


Slika 41: Primjer uporabe općeg semafora

Na slici 41 prikazan je primjer uporabe općeg semafora za sinkronizaciju tri raspodijeljena programa. Raspodijeljeni program RP₃ ovisi o raspodijeljenim programima RP₁ i RP₂. Izvođenje raspodijeljenog programa RP₃ započinje tek nakon što jedan od raspodijeljenih programa RP₁ i RP₂ otpusti semafor. Na početku RP₁ i RP₂ zauzimaju semafor (1), (2). Opći semafor u primjeru konfiguriran je tako da ga odjednom mogu zauzeti najviše dva raspodijeljena programa. Stoga kada RP₃ pokuša zauzeti opći semafor, isti vraća raspodijeljenom programu poruku o neuspjehu i dodaje zahtjev raspodijeljenog programa RP₃ u rep čekanja (3). Kasnije, kada program RP₂ otpusti opći semafor, raspodijeljeni program RP₃ koji je prvi u repu čekanja zauzima semafor. Opći semafor istovremeno dojavljuje raspodijeljenom programu RP₃ obavijest o zauzimanju općeg semafora (5).

5.1.2. Spremnik poruka

Spremnik poruka programsko je ostvarenje stavi/dohvati komunikacijskog obrasca koji omogućuje postojanu asinkronu komunikaciju među raspodijeljenim programima. Sudionici komunikacije putem spremnika poruka razmjenjuju općenite poruke koje su oblikovane kao XML dokumenti te nose proizvoljne podatke. Zbog navedenog, spremnik poruka podržava model komunikacije i obrade poruka neovisan o korištenoj platformi.



Slika 42: Primjer uporabe spremnika poruka

Na slici 42 prikazan je primjer uporabe spremnika poruka za uspostavljanje komunikacije zasnovane na porukama između četiri raspodijeljena programa. Spremnik poruka omogućava vremensku i referencijsku nepovezanost raspodijeljenih programa RP_1 i RP_2 od raspodijeljenih programa RP_3 i RP_4 . Raspodijeljeni programi RP_1 i RP_2 generiraju poruke i stavljaju ih u spremnik poruka, a raspodijeljeni programi RP_3 i RP_4 dohvaćaju poruke iz spremnika te ih obrađuju. Budući da oba raspodijeljena programa RP_3 i RP_4 nastoje dohvatiti nove poruke, dobiva se učinak njihovog međusobnog natjecanja. Scenarij u primjeru započinje tako što raspodijeljeni programi RP_3 i RP_4 šalju zahtjev za porukom spremniku poruka (1), (2). Budući da je spremnik poruka prazan, odnosno nema raspoloživih poruka, zahtjevi za porukama raspodijeljenih programa RP_3 i RP_4 dodaju se u rep čekanja. Nakon toga, raspodijeljeni program RP_1 dodaje poruku A u spremnik poruka (3). S obzirom da postoji zahtjevi u repu čekanja, spremnik poruka pristiglu poruku šalje raspodijeljenom programu RP_1 čiji se zahtjev nalazi prvi u repu (4). Konačno, kada raspodijeljeni program RP_1 doda poruku B u spremnik poruka (5), spremnik poruka ponavlja opisani postupak slanjem poruke B raspodijeljenom programu RP_4 (6).

6. Usmjernik događaja zasnovan na uslugama

Usmjernik događaja (engl. *event channel*, EC) ostvaren je kao objava/preplata komunikacijski posrednički sustav koji pruža napredne mogućnosti tumačenja događaja te je jedan od mehanizama suradnje i natjecanja razvijenih u sklopu CBDS sustava. Budući da je usmjernik događaja napredan komunikacijski mehanizam zasnovan na uslugama, uporabom usmjernika događaja moguće je razvoj raspodijeljenih sustava poticanih događajima, raspodijeljenih sustava zasnovanih na dokumentima te mreža zasnovanih na sadržaju.

U većini postojećih sustava objava/preplata načinjen je određeni kompromis između svojstva razmjernog rasta sustava i svojstva izražajnosti mehanizma preplaćivanja. Primjerice, *CORBA Notification Service* (CORBA NS) i *Java Message Service* sustavi su ograničeni jer koriste unaprijed definiran jezik za preplaćivanje zasnovan na atributima. Također, sustavi ostvareni kao CORBA NS ili JMS nisu predviđeni za međusobno spajanje u mreže. Dodatno, sustavi su strogo centralizirani jer jedna komponenta koja je smještena na jednom računalu u isto vrijeme obrađuje događaje te prima događaja i preplate. Zbog navedenih razloga, raspodijeljeni sustavi ostvareni pomoću CORBA i JMS sustava nemaju svojstvo razmjernog rasta. Osim toga, sustav CORBA NS za komunikaciju koristi protokol IIOP, a JMS komunicira pomoću protokola Java RMI pa je komunikacija kod oba sustava ovisna o računalnoj platformi.

S druge strane, SIENA i HERMES sustavi namijenjeni su za izgradnju mrežna zasnovanih na sadržaju velikih razmjera. Međutim, poput CORBA NS i JMS sustava, zasnovani su na posebnim protokolima niske razine pa ni oni nisu neovisni o platformi. Sustavi SIENA i HERMES pružaju učinkovit mehanizam usmjeravanja događaja u mreži na uštrb svojstva izražajnosti mehanizma preplaćivanja. Navedeni sustavi koriste preplaćivanje zasnovano na atributima. SIENA dodatno podržava jednostavni model preplaćivanja na *uzroke događaja* koji se zadaju kao vremenski slijed događaja. Navedenim sustavima zajednička je ograničena prilagodljivost primjeni te ovisnost o platformi. Također, komponente navedenih sustava čvrsto su povezane pa je teško proširiti ili izmijeniti funkcionalnosti sustava.

Prilikom razvoja usmjernika događaja jedan od ciljeva je bio nadilaženje ograničenja koje posjeduju navedeni sustavi. Usmjernik događaja omogućuje korisnicima da sami odrede razinu svojstva izražajnosti i razinu svojstva razmjernog rasta ovisno o raspodijeljenom sustavu koji koristi usmjernik događaja. Korisnici objavitelji objavljaju događaje općenitog tekstuallnog zapisa na usmjernik događaja, a korisnici preplatnici preplaćuju se zadajući kriterije tumačenja događaja također putem općenitog tekstuallnog zapisa. Ispitivanje zadovoljenosti kriterija nad skupom objavljenih događaja ne

obavlja sam usmjernik događaja. Naime, ispitivanje obavljaju vanjski interpretatori koje poziva usmjernik događaja. Ostvarenje interpretatora i ugrađenog mehanizma ispitivanja događaja prepušteno je korisniku ili nekoj trećoj osobi. Na taj način moguće je prilagoditi i proširiti funkcionalnost interpretatora, a jezik za pretplaćivanje moguće je definirati ovisno o primjeni i potrebi. Međutim, složenost tumačenja događaja nije ograničena jezikom za pretplaćivanje. Za izgradnju interpretatorovog mehanizma tumačenja događaja moguće je koristi napredna svojstva programskog jezika i razvojne platforme pomoću kojih se interpretator programski ostvaruje.

Ako se na usmjerniku događaja pojavi skup događaja koji zadovoljava kriterije korisnika pretplatnika onda usmjernik događaja šalje dojavu pripadnim korisnicima pretplatnicima. Funkcionalnost usmjernika događaja izložena je putem sučelja za pretplaćivanje i sučelja za objavljivanje događaja. Stvaranje i uništavanje instanci usmjernika događaja omogućeno je putem *sučelja tvornice*.

Učinkovitost obrade događaja moguće je povećati po potrebi budući da je na pojedini usmjernik događaja moguće spojiti proizvoljan broj interpretatora. Također, usmjernike događaja moguće je povezati u mrežu i na taj način je omogućeno svojstvo razmernog rasta.

Programsko ostvarenje usmjernika događaja koristi *Web services*, *WS-Resource Framework* i *WS-Addressing* standarde. Usmjernik događaja ostvaren je kao *WS-Resource* pa je na jednom računalu moguće stvoriti više instanci usmjernika događaja.

6.1. Arhitektura usmjernika događaja

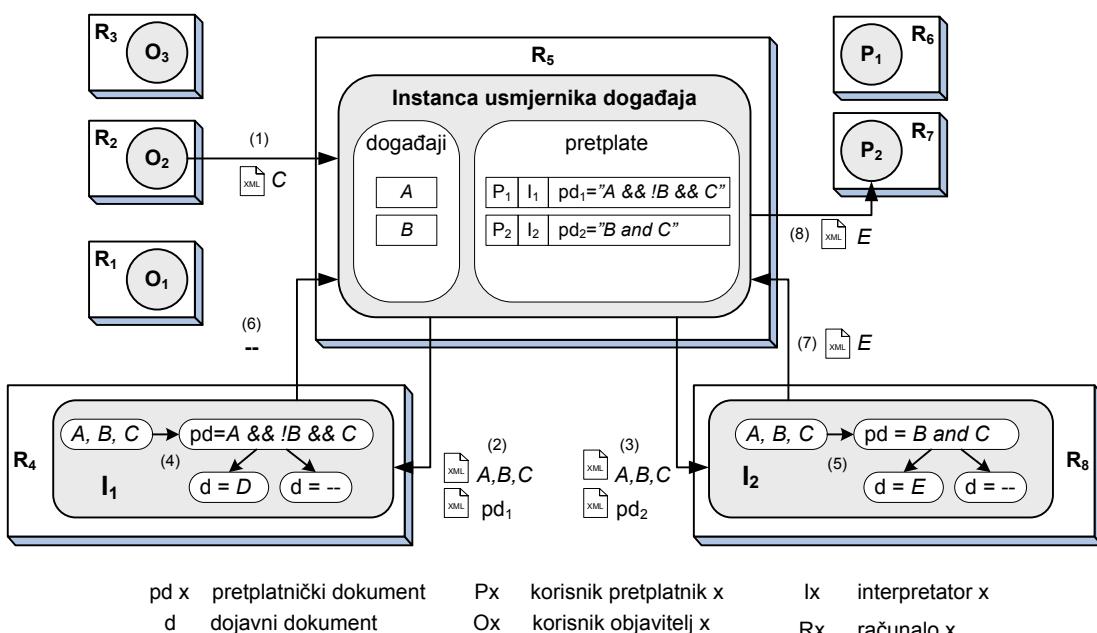
Prilikom pretplaćivanja, korisnik pretplatnik navodi adresu vanjskog interpretatora kojeg želi koristiti za tumačenje događaja te pretplatnički dokument kojim dodatno specificira kriterije preplate. Pretplatnički dokumenti i objavljeni događaji oblikovani su kao općeniti XML dokumenti. Vanjski interpretatori ostvaruju mehanizam za parsiranje pretplatničkog dokumenta i skupa događaja. Također, vanjski interpretatori ostvaruju mehanizam za ispitivanje skupa događaja prema kriterijima zadanim pretplatničkim dokumentom, te prema kriterijima ugrađenim u programsku logiku interpretatora. Ako trenutni skup objavljenih događaja zadovoljava zadane kriterije, interpretator vraća pozitivan odgovor usmjerniku događaja u obliku novog generiranog događaja. Nakon toga usmjernik događaja dojavljuje generirani događaj pripadnom pretplatniku.

Kako bi se omogućilo dojavljivanje pretplatniku, pretplatnik ostvaruje mehanizam za dojavljivanje (engl. *callback mechanism*) te pripadno sučelje za dojavljivanje kojim se izlaže funkcionalnost mehanizma. Sučelje korisnika za dojavljivanje nije unaprijed definirano, odnosno korisnik sučelje za dojavljivanje prilagođava vlastitim potrebama. Na taj način omogućeno je spajanje raznih korisnika na usmjernik događaja čime se osigurava slaba povezanost i dinamička prilagodljivost sustava. Da bi

usmjernik događaja saznao detalje o sučelju za dojavljivanje korisnik preplatnik šalje opis sučelja prilikom preplaćivanja.

Kada usmjernik događaja primi novi događaj on poziva interpretatore putem unaprijed dogovorenog sučelja. Interpretatori su ostvareni kao zasebne usluge i može ih pružiti neki treći sudionik. Iz tog razloga interpretatore je moguće prilagoditi za bilo koju primjenu, bez utjecanja na detalje programskog ostvarenja usmjernika događaja. Budući da objavljeni događaj, preplatnički dokument te generirani događaj imaju općeniti tekstualni zapis, pomoću usmjernika događaja moguće je ostvariti jezik za preplaćivanje i opisivanje događaja s bilo kojom semantikom. Kako usmjernik događaja ne obrađuje niti ispituje razmijenjene dokumente on je potpuno je neovisan o semantici dokumenata i njihovoj primjeni. Stoga usmjernik događaja pruža općeniti objava/preplata mehanizam koji omogućuje ostalim komponentama sustava potpunu prilagodljivost ovisno o primjeni.

Usmjernik događaja moguće je koristi u zahtjevnim i složenim primjenama jer se vremenski zahtjevni zadaci obradivanja dokumenata prebacuju na interpretatore, čiji broj nije ograničen. Na taj način povećava se raspoloživost usmjernika događaja za primanje zahtjeva preplaćivanja i objavljinja jer je trajanje obrade zahtjeva skraćeno budući da usmjernik događaja nije opterećen tumačenjem događaja. Prijavljeni interpretatori pozivaju se tako da rade paralelno čime se postiže učinkovitost obrade velikog broja događaja i preplatničkih dokumenata. Budući da događaje obrađuju prilagodljivi interpretatori, moguće je ostvariti razne vrste filtriranja događaja. Interpretatori mogu biti ostvareni tako da filtriraju prema temi ili prema sadržaju.



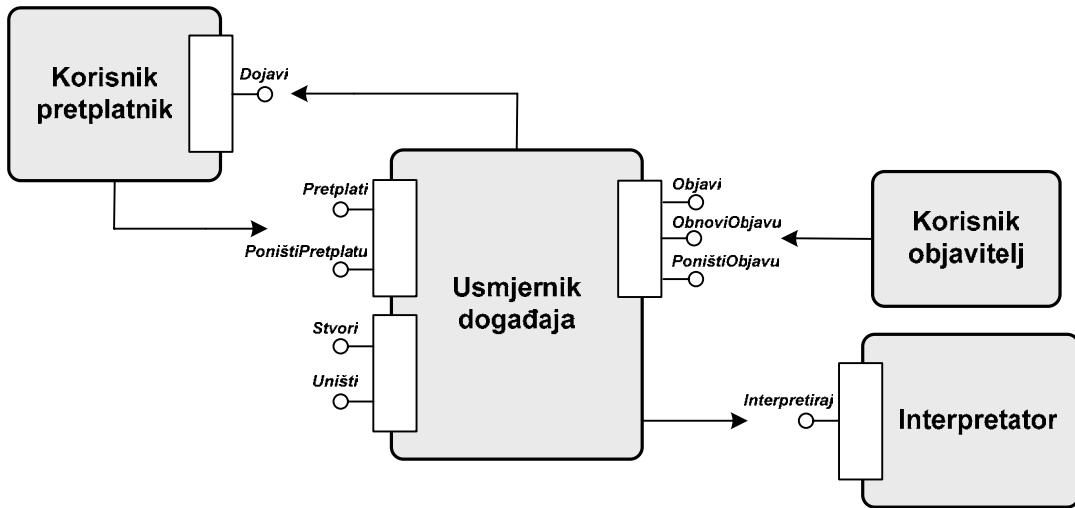
Slika 43: Primjer scenarija rada usmjernika događaja

Također, budući da interpretatori obrađuju cijeli skup objavljenih događaja, moguće je ostvariti filtriranje prema uzorcima i složenim događajima. Dodatno, usmjernik događaja omogućuje

definiranje postojanosti događaja ovisno o potrebama primjene. Podržane su tri vrste događaja s obzirom na postojanost: postojani događaji, potrošljivi događaji i dojavni događaji. Na slici 43 prikazan je primjer scenarija rada usmjernika događaja. Sustav se sastoji od jedne instance usmjernika događaja postavljene na računalo R_5 , tri korisnika objavitelja O_1 , O_2 i O_3 na računalima R_1 , R_2 i R_3 , dva korisnika preplatnika P_1 i P_2 na računalima R_6 i R_7 te dva interpretatora I_1 i I_2 na računalima R_4 i R_8 . U trenutku koji je prikazan u primjeru, usmjernik događaja sadrži dva događaja i dvije preplate. Preplate su definirane trojkom koja se sastoji od adrese i opisa sučelja preplatnika, adrese interpretatora te preplatničkog dokumenta. U ovom slučaju, preplatnički dokumenti definiraju uzorce događaja koji su zadani logičkim operacijama nad događajima. Interpretator I_1 koristi logičke izraze poput onih u programskom jeziku C . Ako skup objavljenih događaja zadovoljava preplatnički dokument onda interpretator I_1 generira dokument D . Interpretator I_2 pruža sličnu funkcionalnost, no uzorci događaja zadaju se prema sintaksi sličnoj sintaksi programskog jezika Pascal. Nadalje, interpretator I_2 za razliku od interpretatora I_1 generira dokument E ako su uvjeti zadovoljeni. Slijed akcija u sustavu označen je brojevima (1)-(8). Na početku, korisnik O_2 objavljuje događaj C na usmjernik događaja (1). Događaj se dodaje u skup događaja čime se mijenja stanje skupa događaja. Promjenom stanja skupa događaja moguć je nastanak uvjeta pri kojima će određeni interpretatori generirati nove događaje. Stoga se pri zaprimanju novog događaja izvodi ciklus interpretiranja. U ciklusu interpretiranja usmjernik događaja svakom interpretatoru šalje skup događaja te pripadni preplatnički dokument. Interpretatoru I_1 se uz skup događaja šalje preplatnički dokument pd_1 (2), a interpretatoru I_2 preplatnički dokument pd_2 (3). Nakon toga, interpretator I_1 analizira parametre zahtjeva te ispituje da li skup događaja zadovoljava preplatnički dokument pd_1 (4). Skup događaja ne zadovoljava preplatnički dokument pa se ne generira novi događaj. S druge strane, skup događaja zadovoljava preplatnički dokument pd_2 poslan interpretatoru I_2 stoga on generira novi događaj E (5). Potom, interpretator I_1 šalje negativan odgovor usmjerniku događaja (6), dok interpretator I_2 šalje generirani događaj E (7). Konačno, usmjernik događaja šalje generirani događaj E preplatniku P_2 (8).

6.1.1. Okolina usmjernika događaja

Okolinu usmjernika događaja čine korisnik preplatnik, korisnik objavitelj, interpretator i sam usmjernik događaja. Na slici 44 prikazana je okolina usmjernika događaja te sučelja svakog sudionika okoline. Usmjernik događaja sadrži sučelje tvornice, sučelje za preplaćivanje i sučelje za objavljivanje. Sučelje tvornice sastoji se od metoda *Stvori* i *Uništi*. Metode *Pretplati* i *PoništiPretplatu* čine sučelje za preplaćivanje, a metode *Objavi*, *ObnoviObjavu* i *PoništiObjavu* čine sučelje za objavljivanje. Korisnik preplatnik sadrži sučelje za dojavljivanje koje pruža metodu *Dojavi*. Interpretator ostvaruje sučelje za interpretiranje koje sadrži metodu *Interpretiraj*. Konačno, korisnik objavitelj ne pruža vanjska sučelja.



Slika 44: Okolina usmjernika događaja

Sučelja usmjernika događaja

Sučelje tvornice opisano u tablici 6 služi za stvaranje i uništavanje instanci usmjernika događaja. Korisnik izdaje zahtjeve za stvaranje nove instance usmjernika događaja pozivanjem metode *Stvari*. Ulazni parametri metode su *identifikator instance usmjernika događaja* te *krajnja točka korisnika*. Krajnja točka korisnika jednoznačno određuje korisnika i sadrži adresu korisnika. Ako korisnik na jednoj adresi sadrži više sjednica koje koriste isti usmjernik događaja onda je potrebna dodatna informacija za jednoznačno određivanje korisnika. U tom slučaju, u krajnju točku dodaje se identifikator sjednice korisnika. Usmjernik događaja ne ovisi o ostvarenju korisnika i s njegove strane je svejedno da li postoji više sjednica korisnika na jednoj adresi. Povratna vrijednost metode *Stvari* je vrijednost koja određuje uspješnost stvaranja nove instance te identifikator instance usmjernika događaja.

sučelje	metoda	parametri	
Tvornica	<i>Stvari</i>	ulazne vrijednosti	identifikator instance usmjernika događaja krajnja točka korisnika
		povratne vrijednosti	identifikator instance usmjernika događaja uspješnost stvaranja
	<i>Uništi</i>	ulazne vrijednosti	identifikator instance usmjernika događaja krajnja točka korisnika
		povratna vrijednost	uspješnost uništenja

Tablica 6: Sučelje tvornice usmjernika događaja

Pomoću metode *Uništi* uništavaju se prije stvorene instance usmjernika događaja. Dopushta se da određenu instancu uništi jedino korisnik koji ju je stvorio. Stoga je osim identifikatora instance parametar metode *Stvari* i krajnja točka korisnika.

Sučelje za pretplaćivanje sastoji se od metoda *Pretplati* i *PoništiPretplatu* čiji je opis prikazan u tablici 7. Pomoću metode *Pretplati* korisnik se pretplaćuje, a pomoću metode *PoništiPretplatu* poništava prije izdanu pretplatu. Parametri koji određuju korisnikovu pretplatu jesu krajnja točka interpretatora, XML pretplatnički dokument te identifikator instance usmjernika događaja. Krajnja točka interpretatora određuje adresu interpretatora kojeg korisnik želi koristiti za tumačenje događaja, a pomoću XML pretplatničkog dokumenta korisnik specificira detaljne kriterije za ispitivanje događaja. *Identifikator instance usmjernika događaja* jednoznačno određuje postojeću *instancu usmjernika događaja* pri usmjerniku događaja koji se poziva. Također, prilikom pretplaćivanja šalje se WSDL opis korisnikovog sučelja za dojavljivanje. Povratna vrijednost je identifikator preplate koji jednoznačno određuje pretplatu pri pripadnoj instanci usmjernika događaja. Prilikom poziva metode *PoništiPretplatu* korisnik šalje kao parametar identifikator preplate kojim određuje pretplatu koju želi poništiti.

Sučelje	metoda	parametri	
<i>Pretplaćivanje</i>	<i>Pretplati</i>	ulazne vrijednosti	identifikator instance usmjernika događaja krajnja točka korisnika WSDL opis sučelja pretplatnika krajnja točka interpretatora XML pretplatnički dokument
		povratna vrijednost	identifikator preplate
	<i>PoništiPretplatu</i>	ulazne vrijednosti	identifikator instance usmjernika događaja krajnja točka korisnika identifikator preplate
		povratna vrijednost	uspješnost poništenja

Tablica 7: Sučelje usmjernika događaja za pretplaćivanje

Metode *Objavi*, *ObnoviObjavu* i *PoništiObjavu* sačinjavaju sučelje usmjernika događaja za objavljinje opisano u tablici 8.

sučelje	metoda	parametri	
<i>Objavljinje</i>	<i>Objavi</i>	ulazne vrijednosti	identifikator instance usmjernika događaja krajnja točka korisnika XML dokument događaja
		povratna vrijednost	identifikator događaja
	<i>ObnoviObjavu</i>	ulazne vrijednosti	identifikator instance usmjernika događaja krajnja točka korisnika identifikator događaja XML dokument događaja
		povratna vrijednost	identifikator događaja
	<i>PoništiObjavu</i>	ulazne vrijednosti	identifikator instance usmjernika događaja krajnja točka korisnika identifikator događaja
		povratna vrijednost	uspješnost poništenja

Tablica 8: Sučelje usmjernika događaja za objavljinje

Korisnik objavljuje događaje pomoću metode *Objavi* te kao parametar šalje XML dokument koji predstavlja događaj. Povratna vrijednost je identifikator događaja kojeg korisnik šalje prilikom obnavljanja objave putem metode *ObnoviObjavu* te prilikom poništavanja objave putem metode *PoništiObjavu*. Pozivom metode *ObnoviObjavu* uz identifikator događaja koji se obnavlja šalje se i novi XML dokument događaja.

Sučelje interpretatora

Sučelje interpretatora opisano je u tablici 9. Sučelje se sastoji od jedne metode *Interpretiraj* koja za parametre ima *preplatnički dokument* i *skup objavljenih događaja*. Preplatnički dokument i događaji predstavljeni su XML dokumentima. Interpretator prema pravilima preplatničkog dokumenta te prema pravilima ugrađenim u programsku logiku obrađuje skup objavljenih događaja. Obradom skupa objavljenih događaja prema postavljenim pravilima generira se novi događaj. *Generirani događaj* i *lista iskorištenih događaja* povratne su vrijednosti interpretatora. Ako je prilikom obrade neki događaj iz skupa objavljenih događaja iskorišten za generiranje novog događaja onda se on dodaje u listu iskorištenih događaja.

sučelje	metoda	parametri	
<i>Interpretiranje</i>	<i>Interpretiraj</i>	ulazne vrijednosti	skup objavljenih događaja (XML dokument) preplatnički dokument (XML dokument)
		povratna vrijednost	generirani događaj (XML dokument) lista iskorištenih događaja

Tablica 9: Sučelje za interpretiranje

Sučelje korisnika preplatnika

Sučelje za dojavljivanje korisnika preplatnika sastoji se od jedne metode *Dojavi*. Pomoću nje usmjernik događaja šalje preplatniku jedan generirani dojavni događaj. Osim XML dokumenta koji predstavlja dojavni događaj, kao parametri šalju se identifikator sjednice te krajnja točka usmjernika događaja. Identifikator sjednice označava sjednicu korisnika kojoj je dojava namijenjena ako iza usluge korisnika postoji više sjednica koje koriste usmjernik događaja. Da bi korisnik znao izvorište generiranog događaja, prilikom dojavljivanja događaja šalje se krajnja točka usmjernika događaja koja sadrži adresu usmjernika događaja i identifikator pripadne instance. Na taj način jednoznačno je određena instanca usmjernika događaja od koje je pristigla dojava preplatniku.

sučelje	metoda	parametri	
<i>Dojavljivanje</i>	<i>Dojavi</i>	ulazne vrijednosti	identifikator sjednice korisnika krajnja točka usmjernika događaja dojavni događaj (XML dokument)
		povratna vrijednost	uspješnost dojavljivanja

Tablica 10: Sučelje za dojavljivanje

6.1.2. Vrste postojanosti događaja

Postojanost događaja je svojstvo koje određuje vijek trajanja i način poništavanja događaja u usmjerniku događaja. Usmjernik događaja podržava tri vrste događaja s obzirom na njihovu postojanost: postojane događaje, potrošljive događaje i dojavne događaje. Objavitelj prilikom objavljivanja na usmjernik događaja definira vrstu postojanosti događaja. U korijenski element XML dokumenta koji predstavlja događaj dodaje se atribut koji definira vrstu postojanosti događaja.

Postojani događaji

Postojani događaji služe za opisivanje stanja sustava ili događaje čiji je vijek trajanja dug. Koriste se za dokumente i podatke koji se ne mijenjaju učestalo, primjerice podatak koji sadrži iznos temperature. Postojani događaji aktivni su u usmjerniku događaja sve dok korisnik ne poništi ili obnovi objavljeni događaj pomoću metoda *PoništiObjavu* i *ObnoviObjavu*.

Potrošljivi događaji

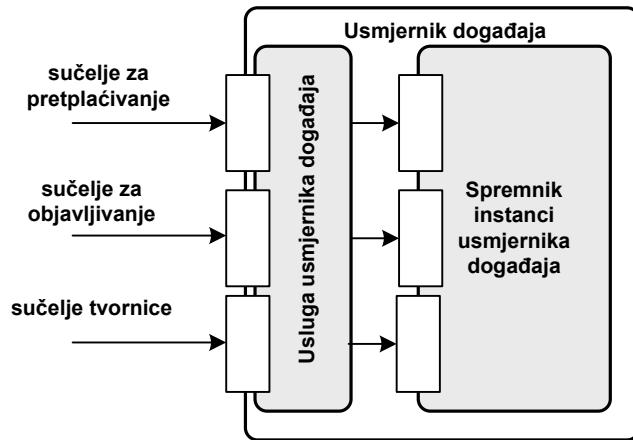
Za razliku od postojanih događaja, potrošljivi događaji automatski se uklanjaju iz usmjernika događaja ako ih neki interpretator iskoristi za generiranje novog događaja prema pravilima pretplate određenog korisnika. Potrošljive događaje je također moguće poništiti i obnoviti za vrijeme dok su aktivni u usmjerniku događaja.

Dojavni događaji

Dojavni događaji predstavljaju važnu dojavu koja služi kao signal usmjerniku događaja za pokretanje ciklusa interpretiranja. Dojavni događaji su privremeni, odnosno ne ostaju u usmjerniku događaja bez obzira da li su iskorišteni za generiranje novih događaja.

6.1.3. Održavanje stanja usmjernika događaja

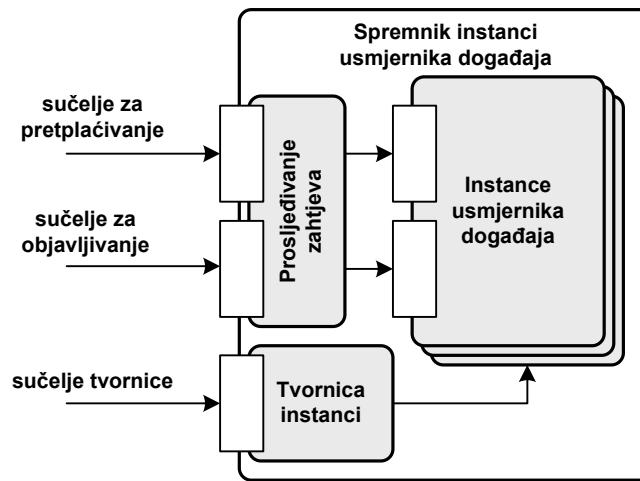
Arhitektura usmjernika događaja sastoji se od mrežne usluge koja pruža javno sučelje usmjernika događaja te spremnika instanci usmjernika događaja koji omogućuje održavanje stanja usmjernika događaja. Spremnik instanci kao i mrežna usluga pruža sučelje tvornice, sučelje za pretplaćivanje te sučelje za objavljivanje događaja. Stoga mrežna usluga obavlja preslikavanje vanjskih javnih sučelja na istoimena sučelja spremnika instanci.



Slika 45: Arhitektura usmjernika događaja

Spremnik instanci usmjernika događaja

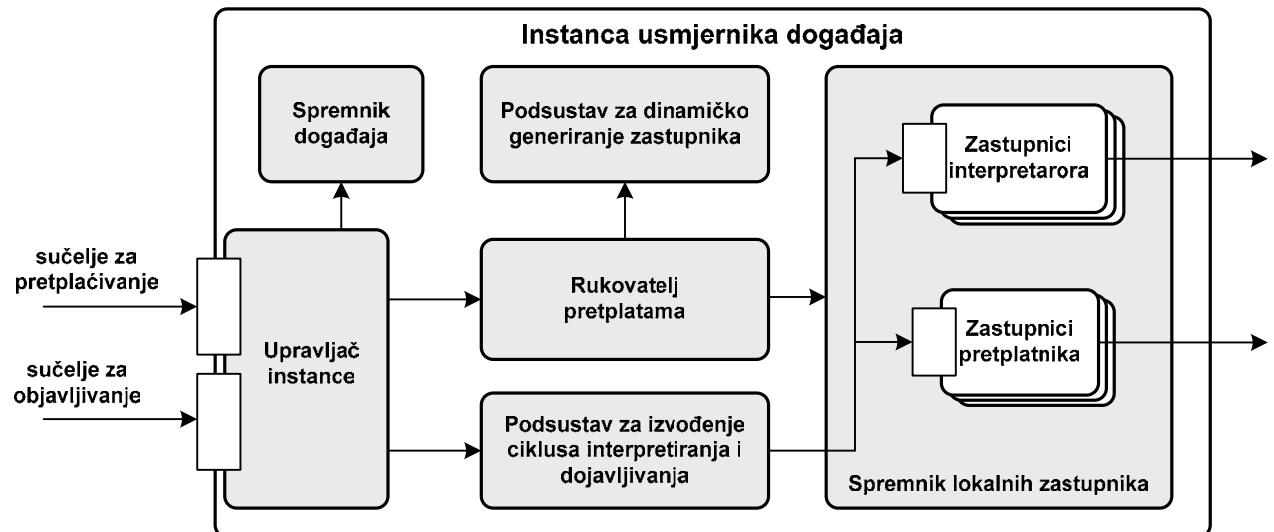
Spremnik instanci usmjernika događaja ostvaruje mehanizme za upravljanje instancama usmjernika događaja, odnosno ostvaruje mehanizme za njihovo stvaranje, dohvaćanje te uništavanje. Sučelje spremnika usmjernika događaja sadrži istoimene metode kao i sučelje mrežne usluge usmjernika događaja. Na slici 46 prikazana je arhitektura spremnika usmjernika događaja. Podsustav tvornice instanci obavlja zadaću stvaranja i uništavanja instanci. Prilikom poziva neke od metoda sučelja za preplaćivanje ili objavljivanje, podsustav za proslijedivanje zahtjeva dohvaća instancu određenu *identifikatorom instance* te joj proslijedi zahtjev sadržan u pozivu metode nad spremnikom. Nakon što je zahtjev proslijeđen instanci, spremnik je raspoloživ za primanje sljedećeg zahtjeva, pa na taj način spremnik omogućava pristup i korištenje više instanci istovremeno.



Slika 46: Arhitektura spremnika instanci usmjernika događaja

6.1.4. Arhitektura instance usmjernika događaja

Arhitektura instance usmjernika događaja prikazana je na slici 47. Osnovni podsustavi instance jesu: *Upravljač instance*, *Rukovatelj preplatama*, *Spremnik događaja*, *Podsustav za izvođenje ciklusa interpretiranja i dojavljivanja* te *Podsustav za dinamičko generiranje zastupnika*.



Slika 47: Arhitektura instance usmjernika događaja

Upravljač instance

Upravljač instance pruža sučelje putem kojeg spremnik poziva instancu. Upravljač instance obrađuje parametre zahtjeva te ovisno o zahtjevu poziva ostale podsustave i koordinira njihov rad. Ovisno da li je pozvana metoda sučelja za preplaćivanje ili objavljivanje, upravljač poziva *Spremnik događaja* ili *Rukovatelja preplatama*. Budući da se pozivom metode sučelja za preplaćivanje mijenja skup preplata, a pozivom metode sučelja za objavljivanje mijenja skup događaja, pri svakom pozivu upravljač instance poziva i *Podsustav za izvođenje ciklusa interpretiranja i dojavljivanja*.

Spremnik događaja

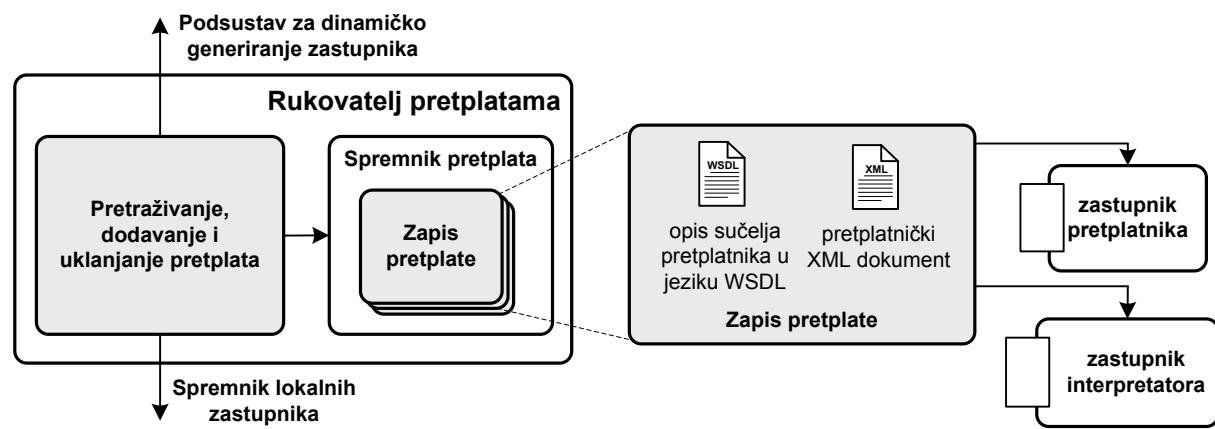
Spremnik događaja upravlja objavljenim događajima, odnosno njihovim dodavanjem, obnavljanjem i uklanjanjem iz skupa događaja. Prilikom poziva metode *Objavi*, *ObnoviObjavu* ili *PoništiObjavu* upravljač instance poziva *Spremnik događaja* koji ažurira skup događaja. Također, ažuriranje se obavlja nakon svakog ciklusa interpretiranja i ovisi o postojanosti događaja. Interpretatori vraćaju liste iskorištenih događaja, a *potrošljivi događaji* koji se nalaze na nekoj listi uklanjuju se iz spremnika.

Spremnik lokalnih zastupnika

Usmjernik događaja poziva udaljene mrežne usluge vanjskih interpretatora i pretplatnika. Postupak pozivanja vanjskih interpretatora i pretplatnika obavlja se pomoću lokalnih zastupnika koji su spremljeni u *spremniku lokalnih zastupnika*. Sučelje interpretatora je dobro poznato (engl. *well known*), odnosno unaprijed je definirano pa je izvodljivi kôd lokalnog zastupnika interpretatora moguće generirati za vrijeme razvoja usmjernika događaja. Za vrijeme izvođenja dovoljno je po potrebi stvarati instance lokalnih zastupnika. S druge strane, svi dijelovi sučelja za dojavljivanje pretplatnika nisu poznati za vrijeme razvoja. Stoga korisnik prilikom pretplaćivanja šalje WSDL opis sučelja za dojavljivanje, a usmjernik događaja za vrijeme izvođenja dinamički generira izvodljivi kôd lokalnog zastupnika pretplatnika.

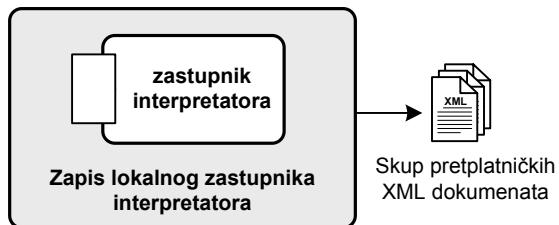
Rukovatelj preplatama

Rukovatelj preplatama brine se o dodavanju i uklanjanju pretplata te poziva *Podsustav za dinamičko generiranje lokalnih zastupnika*. Prilikom poziva metode *Pretplati* ili *PoništiPretplatu* na *Upravljaču instance* zahtjev se prosljeđuje komponenti *Rukovatelj preplatama*. Zahtjev za pretplaćivanje *Rukovatelj preplatama* obrađuje stvaranjem zapisa pretplate u spremniku pretplata. Zapis pretplate sastoji se od opisa sučelja pretplatnika u jeziku WSDL i pretplatničkog XML dokumenta koji su proslijedeni kao parametri metode. Također, zapis pretplate povezuje se s pripadnim lokalnim zastupnikom interpretatora i pripadnim lokalnim zastupnikom pretplatnika. *Rukovatelj preplatama* prvo utvrđuje da li već postoji zapis pretplate s prijavljenim istim interpretatorom. Stoga se na osnovu krajnje točke interpretatora pretražuje spremnik pretplata. Ako krajnja točka već postoji, onda se pretplata povezuje na postojeći lokalni zastupnik interpretatora, inače se instancira novi lokalni zastupnik u spremniku lokalnih zastupnika. Svakom zapisu pretplate pridružen je i lokalni zastupnik pretplatnika pomoću kojeg se pripadnom pretplatniku dojavljuju događaji.



Slika 48: Arhitektura rukovatelja preplatama i zapis pretplate u spremniku

Slično kao i prilikom povezivanja zapisa preplate i pripadnog zastupnika interpretatora, provodi se pretraživanje spremnika preplata. Ako postoji zapis preplate koji sadrži isti WSDL dokument opisa sučelja preplatnika, onda se nova preplata povezuje s postojećim lokalnim zastupnikom te preplate. Inače, poziva se podsustav za dinamičko generiranje lokanog zastupnika preplatnika. Povezivanjem zapisa preplata s postojećim lokalnim zastupnicima izbjegava se uvišestručavanje istovjetnih lokalnih zastupnika. Također, poboljšava se učinkovitost jer se vremenski zahtjevan postupak dinamičkog generiranja ne provodi ako nije nužno. Osim što su preplate povezane s pripadnim lokalnim zastupnicima interpretatora, uz svaki lokalni zastupnik interpretatora pridružen je zapis koji ga povezuje sa svim preplatama koje su na njega povezane. Zapis se ažurira tijekom povezivanja preplate na lokalnog zastupnika interpretatora. Prilikom poziva lokalnog zastupnika interpretatora, pomoću zapisa se učinkovito određuju preplatnički dokumenti koje je potrebno poslati kao parametar poziva prilikom pozivanja interpretatora. Za svaki preplatnički dokument interpretator se zasebno poziva.



Slika 49: Zapis lokalnog zastupnika interpretatora u spremniku

Podsustav za dinamičko generiranje zastupnika preplatnika

Korisniku preplatniku omogućuje se konfiguiranje određenih dijelova sučelja za dojavljivanje prema vlastitoj potrebi. Primjerice, moguće je prilagoditi naziv metode, prostor imena, naziv mrežne usluge te ostale nazive korištene u sučelju mrežne usluge. Budući da je parametar metode *Dojavi* događaj koji je predstavljen XML dokumentom, korisnik ima potpunu slobodu pri izgradnji sučelja za dojavljivanje. Prilikom preplaćivanja, korisnik kao dio preplate šalje WSDL dokument s opisom sučelja koji se koristi u podsustavu za dinamičko generiranje zastupnika prikazanom na slici 50. Postupak dinamičkog generiranja zastupnika započinje analiziranjem sučelja iz kojeg se izdvajaju korisnički definirani podaci kao što su naziv metode, naziv prostora imena te naziv usluge. Izdvojeni podaci iz sučelja preplatnika umeću se u predložak izvornog kôda zastupnika preplatnika te se kao rezultat dobiva potpuni izvorni kôd zastupnika preplatnika. Predložak izvornog kôda zastupnika izrađuje se za vrijeme razvoja usmjernika događaja.



Slika 50: Organizacija podsustava za dinamičko generiranje zastupnika

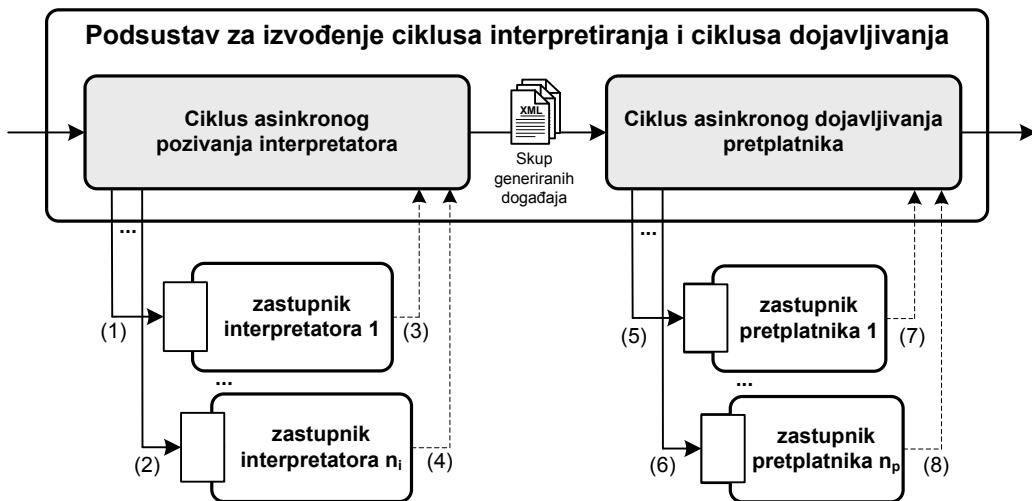
Sljedeći korak je prevođenje izvornog kôda zastupnika preplatnika, a rezultat je izvodljivi kôd lokalnog zastupnika preplatnika. Konačno, korištenjem izvodljivog kôda zastupnika instanciraju se potrebni objekti lokalnog zastupnika preplatnika. Budući da je usmjernik događaja razvijen korištenjem platforme *.NET Framework*, izvorni kôd zastupnika zapisan je u jezik C#, a ciljni kôd u jeziku CIL (*Common Intermediate Language*).

Podsustav za izvođenje ciklusa interpretiranja i ciklusa dojavljivanja

Stanje instance usmjernika događaja određeno je skupom pretplata u spremniku pretplata i skupom događaja u spremniku događaja. Rezultat ciklusa interpretiranja, skup generiranih događaja nastao pozivom prijavljenih interpretatora, jednoznačno je određen stanjem instance usmjernika događaja. Stoga se prilikom svake promjene stanja usmjernika događaja, izvodi ciklus interpretiranja, odnosno pozivaju se prijavljeni interpretatori. Potom se izvodi ciklus dojavljivanja, odnosno dojavljivanje generiranih događaja pretplatnicima. Promjena stanja usmjernika događaja uzrokovana je pozivom jedne od metoda sučelja za pretplaćivanje ili objavljivanje koja mijenja stanje instance usmjernika događaja, odnosno skup pretplata ili skup događaja. *Upravljač instance* prilikom poziva jedne od metoda sučelja za pretplaćivanje ili dojavljivanje poziva podsustav za izvođenje ciklusa interpretiranja i ciklusa dojavljivanja poziva.

Pozivom metoda *Pretplati* izvodi se interpretiranje samo za novopristiglu pretplatu. Naime, na skup događaja nova pretplate ne utječe, odnosno dodavanje nove pretplate u skup pretplata ne utječe na ostale pretplate koje se u nalaze u skupu pretplata. Stoga nije potrebno izvoditi interpretiranje za ostale pretplate. Ako pozvani interpretator generira događaj, pripadnom pretplatniku se dojavljuje generirani događaj. Prilikom poziva metode *PoništiPretplatu* ne izvodi se interpretiranje jer se pozivom te metode naznačuje da pripadna pretplata više nije važeća.

S druge strane, pozivom jedne od metoda *Objavi*, *ObnoviObjavu* te *PoništiObjavu* mijenja se skup događaja. Promjena skupa događaja utječe na sve aktivne pretplate pa se izvodi interpretiranje za svaku pretplatu u spremniku pretplata pozivanjem pripadnog interpretatora. Pojedinom interpretatoru šalje se skup svih događaja te pridruženi pretplatnički dokument. Da bi se osigurala učinkovita obrada događaja, interpretatori se pozivaju asinkrono. Na taj način omogućava se paralelni rad svih interpretatora. Nakon što se svi interpretatori pozovu, podsustav čeka na završetak rada svih interpretatora. Potom podsustav prikuplja rezultate izvođenja pojedinog interpretatora.

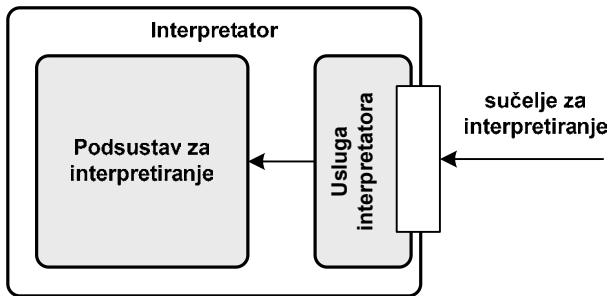


Slika 51: Organizacija podsustava za izvođenje ciklusa interpretiranja i ciklusa dojavljivanja

Ukupan rezultat izvođenja svih interpretatora je skup generiranih događaja. Nakon ciklusa interpretiranja, pozivaju se pretplatnici putem njihovih lokalnih zastupnika. Dojavljuju se oni pretplatnici za koje su u ciklusu interpretiranja generirani događaji. Zbog učinkovitosti, pozivanje pretplatnika također se odvija asinkrono pa se svi pretplatnici dojavljuju istovremeno. Podsustav za izvođenje ciklusa interpretiranja i ciklusa dojavljivanja te slijed pozivanja prikan je na slici 51.

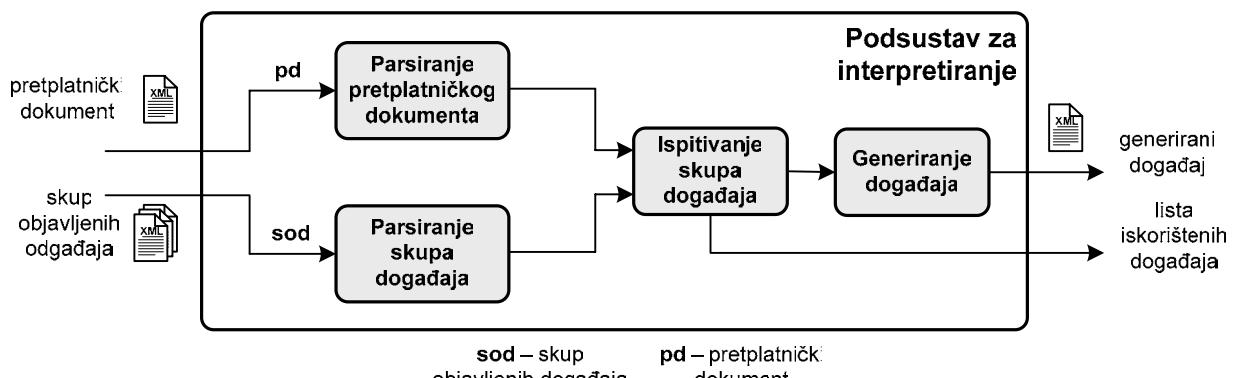
6.1.5. Arhitektura interpretatora

Arhitektura i ostvarenje interpretatora ne ovisi o arhitekturi usmjernika događaja. Na slici 52 prikazana je arhitektura interpretatora ostvarena u ovom radu. Za interpretator nije nužno posjedovanje mogućnosti održavanja stanja, odnosno između uzastopnih poziva nije potrebno postojanje logičke veze. Stoga je interpretator *obradivač poruka* (engl. *message processor*), jer rezultat obrade događaja ovisi jedino o ulaznim podacima koje čine skup događaja i pretplatnički dokument. Interpretator prikazan na slici 52 sastoji se od usluge koja izlaže javno sučelje za interpretiranje te podsustava za interpretiranje.



Slika 52: Arhitektura interpretatora

Primjer arhitekture podsustava za interpretiranje prikazan je na slici 53. Ulaz u podsustav je preplatnički dokument te skup objavljenih događaja. Prvo se obavlja parsiranje preplatničkog dokumenta i parsiranje skupa događaja. Potom se ispituje da li skup događaja zadovoljava uvjete u preplatničkom dokumentu. Ako su uvjeti zadovoljeni, onda se u sljedećem koraku generira dojavni događaj koji je povratna vrijednost interpretatora. Izlaz iz komponente za ispitivanje je lista iskorištenih događaja koja se također umeće u povratnu vrijednost interpretatora.



Slika 53: Arhitektura podsustava za interpretiranje

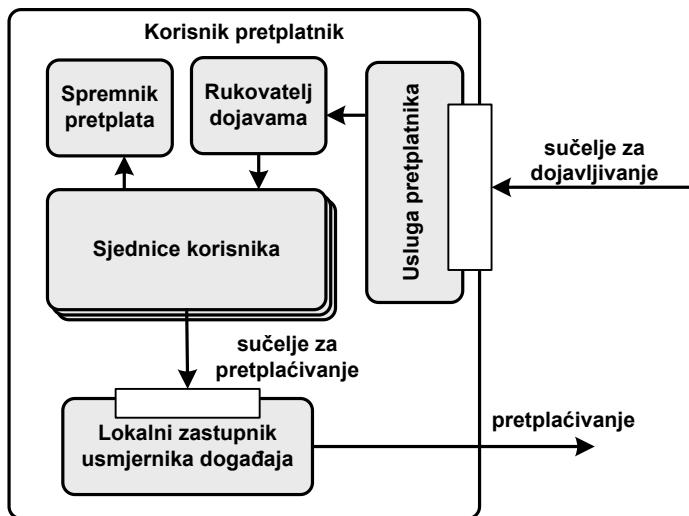
6.1.6. Arhitektura korisnika objavitelja i korisnika preplatnika

Arhitekture korisnika objavitelja i korisnika preplatnika poput arhitekture interpretatora, nisu vezane uz arhitekturu i ostvarenje usmjernika događaja. Također, način programskog ostvarenja korisnika je slobodan. Nužan uvjet za omogućavanje zajedničkog rada korisnika i usmjernika događaja je poštivanje dogovorenih sučelja. Primjer arhitekture korisnika objavitelja korištene u ovom radu prikazan je slici 54. Sastavne komponente arhitekture su skup sjednica korisnika, spremnik objava i lokalni zastupnik usmjernika događaja. Sjednice korisnika pohranjuju u spremnik objava objavljene događaje koji se kasnije koriste prilikom obnavljanja ili poništavanja događaja. Pozivanjem metoda sučelja lokalnog zastupnika korisnik objavljuje događaje na usmjerniku događaja.



Slika 54: Arhitektura korisnika objavitelja

Slično kao arhitektura objavitelja, arhitektura korisnika pretplatnika sadrži spremnik pretplata te lokalnog zastupnika usmjernika događaja. Dodatno, arhitektura sadrži uslugu sa sučeljem za dojavljivanje te komponentu *Rukovatelj dojavama*. Prilikom pristizanja dojave, usluga pretplatnika prosljeđuje dojavu *Rukovatelju dojavama* koji obrađuje pristiglu dojavu te ju šalje odgovarajućoj sjednici korisnika.



Slika 55: Arhitektura korisnika pretplatnika

6.2. Programsко ostvarenje usmjernika događaja

Usmjernik događaja programski je ostvaren koristeći .NET radnu okolinu i ASP.NET *Web services* tehnologiju. Za razvoj je korišten programski jezik C#. Usmjernik događaja razvijen je prema WS-Resource Framework standardu pa je pojedina instanca usmjernika događaja programski ostvarena kao *WS-Resource*. Za razvoj programskog ostvarenja i ispitivanje ispravnosti korišten je *Microsoft IIS 5.1* mrežni poslužitelj. Mrežne usluge prijavljene su na mrežni poslužitelj koji zahtjeve za uslugama prosljeđuje ASP.NET podsustavu. Komponente i podsustavi usmjernika događaja programski su ostvareni kao C# razredi.[8]

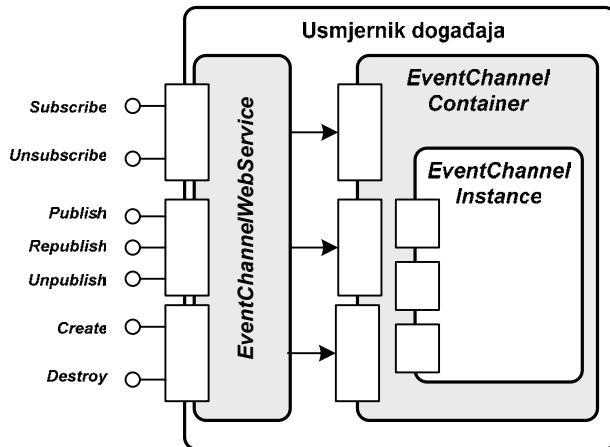
Pri razvoju mrežnih usluga s održavanjem stanja, jedna od zadaća je razvoj mehanizma za upravljanje stanjem pojedinih instanci. Potrebno je osigurati njihovu postojanost između pojedinih zahtjeva za uslugom te neovisnost o privremenim objektima koji nastaju pri svakom zahtjevu. ASP.NET podsustav sadrži radni proces unutar kojeg se stvaraju dretve pri svakom zahtjevu za mrežnom uslugom. Mrežna usluga postavljena na poslužitelj sastoji se od izvodljivog kôda razreda koji ostvaruje funkcionalnost mrežne usluge i *.asmx* dokumenta koji pokazuje na pripadni razred i pomoću njega se pokreće mrežna usluga. Mrežna usluga usmjernika događaja postavljena na poslužitelj *www.ris.fer.hr* poziva se pomoću sljedeće adrese:

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

Za mrežnu uslugu usmjernika događaja osim stvaranja dretve instancira se objekt razreda mrežne usluge. Održavanje stanja, odnosno pohranjivanje spremnika i instanci usmjernika događaja moguće je postići na dva osnovna načina. Prvi način je pomoću zasebnog procesa koji brine o održavanju stanja. Komunikacija između ASP.NET radnog procesa u kojem se stvaraju dretve zahtjeva i zasebnog procesa programski se ostvara putem *.NET Remoting* mehanizma koji omogućuje pozivanje udaljenih objekta. Drugi način je pohranjivanje spremnika i instanci u *Application* objekt ASP.NET radnog procesa [6]. *Application* objekt postoji za cijelo vrijeme života mrežne usluge, odnosno za vrijeme dok je mrežna usluga postavljena na mrežnom poslužitelju. Prvo rješenje je robusnije jer ne ovisi o ASP.NET podsustavu pa prestanak rada ASP.NET podsustava ne uzrokuje nestanak spremnika i instanci. Prednost drugog rješenja je jednostavnost i učinkovitost jer se svi objekti nalaze unutar istog procesa pa je programsko ostvarenje komunikacije među objektima jednostavno. Oba rješenja su ispitana, a odabранo je drugo rješenje za mehanizam održavanja stanja.

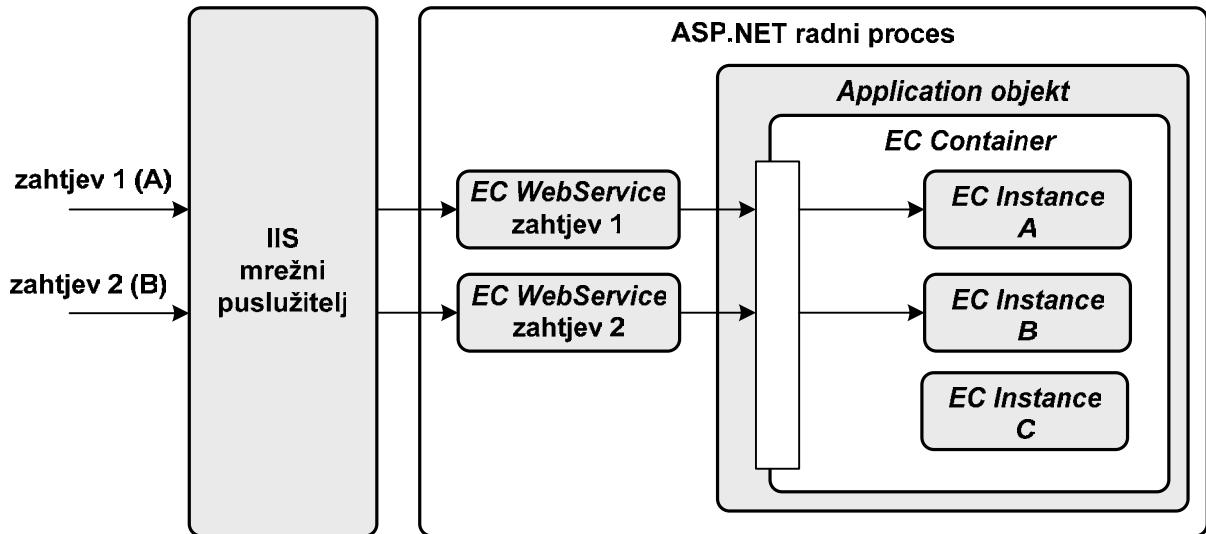
6.2.1. Programska arhitektura i osnovni razredi

Programska arhitektura usmjernika događaja prikazana je na slici 56. Mrežna usluga programski je ostvarena razredom *EventChannelWebService*, spremnik usmjernika događaja razredom *EventChannelContainer*, a instanca usmjernika događaja razredom *EventChannelInstance*. Metode sučelja za pretplaćivanje nazivaju se *Subscribe* i *Unsubscribe*, metode sučelja za objavljivanje *Publish*, *Unpublish* i *Republish*, a metode sučelja tvornice *Create* i *Destroy*.



Slika 56: Programska arhitektura usmjernika događaja

Primjer scenarija u okolini programske arhitekture usmjernika događaja prikazan je na slici 57. Mrežni poslužitelj istovremeno prima dva zahtjeva za mrežnom uslugom usmjernika događaja. Mrežni poslužitelj prepoznaje da su zahtijevane mrežne usluge zasnovane na ASP.NET podsustavu te prosljeđuje zahtjeve ASP.NET radnom procesu. Unutar ASP.NET radnog procesa, za svaki pojedini zahtjev stvara se objekt razreda *EventChannelWebService* te dretva koja obrađuje zahtjev. *Application* objekt sadrži *EventChannelContainer* objekt koji sadrži tri instance usmjernika događaja odnosno tri objekta *EventChannelInstance* razreda. *EventChannelContainer* i *EventChannelInstance* razredi ostvarenici su tako da se osigura što veća dostupnost spremnika te omogući istovremen pristup više instanci odjednom. U primjeru, prvi zahtjev pristupa instanci A dok drugi zahtjev za to vrijeme pristupa instanci B.



Slika 57: Okolina programske arhitekture usmjernika događaja

6.2.2. Sučelje mrežne usluge

Razred *EventChannelWebService* izveden je iz razreda *WebService* koji je dio programske potpore za razvoj mrežnih usluga koje pruža .NET radna okolina. Sučelje razreda *EventChannelWebService* ostvareno je tako da pruža eksplicitnu tvornicu *WS-Resourcea* (engl. *WS-Resource factory*) te eksplicitan pristup postojećim *WS-Resource* instancama. WS-Resource instance adresiraju se prema WS-Addressing standardu.

U primjeru na slici 58 prikazana je SOAP poruka poziva metode *Subscribe* koja služi za pretplaćivanje na usmjernik događaja. Prema *WS-Addressing* standardu, *WS-Resource*instancama pristupa se putem zaglavljiva SOAOP poruke. Stoga se identifikator instance usmjernika događaja prilikom poziva metode zapisuje u zaglavljivo SOAP poruke.

```

<soap:Envelope>
  <soap:Header>
    <ResourceID xmlns="http://www.ris.fer.hr/CoopetitionServices/EventChannel">
      A
    <ResourceID/>
  </soap:Header>
  <soap:Body>
    <Subscribe xmlns="http://www.ris.fer.hr/CoopetitionServices/EventChannel">
      <EndpointReference>...</EndpointReference>
      <CB>
        <EndpointReference>...</EndpointReference>
        <definitions>...</definitions>
      </CB>
      <subDocument>
        xml
      </subDocument>
    </Subscribe>
  </soap:Body>
</soap:Envelope>

```

Slika 58: SOAP omotnica poziva metode *Subscribe*

Identifikator je oblikovan XML elementom *ResourceID*. Ostali parametri metode *Subscribe* šalju se unutar tijela SOAP poruke.

.NET radna okolina omogućava dodavanje opisnih deklaracija varijablama i metodama u obliku *atributa* (engl. *attributes*) koji su posebna vrsta ključnih riječi. Obilježja dodana varijablama i metodama putem atributa predstavljaju njihova dodatna svojstva i upute prevoditelju. Metoda ostvarena u C# jeziku koja odgovara navedenoj SOAP poruci navedena je na slici 59. Metoda *Subscribe* posjeduje atribut *[WebMethod]* koji označava da je pripadna metoda sastavni dio sučelja mrežne usluge, te atribut *[SoapHeader]* koji označava parametre koji se prenose putem SOAP zaglavlja.

```
[WebMethod]
[SoapHeader("rID", Direction=SoapHeaderDirection.In)]
public long Subscribe( EndpointReferenceType intEPR, Callback CB, XmlElement subDocument )
{
    EventChannelContainer ecc = (EventChannelContainer)Application["EventChannelContainer"];
    string ecID = rID.rID;

    // Subscribe
    return ecc.Subscribe( ecID, intEPR, CB, subDocument );
}
```

Slika 59: Metoda *Subscribe*

Svi razredi C# jezika koji su parametri mrežnih usluga, odnosno predstavljaju podatke koje mrežne usluge razmjenjuju, oblikuju se u XML zapis prije slanja. WSDL opis mrežne usluge sadrži definiciju podatkovnih tipova parametara pomoću *XML Schemae*. Postupak pretvaranja objekata .NET radne okoline u XML zapis naziva se *XML serijalizacija*, a obrnuti postupak naziva se *XML deserijalizacija*.

6.2.3. Strukture podataka zasnovane na XML-u

Određeni podatkovni tipovi korišteni u programskom ostvarenju definirani su standardima WS-Resource Framework i WS-Addressing. Primjerice, podatkovni tip koji predstavlja *krajnju točku* definiran je WS-Addressing standardom. Podatkovni tipovi definiraju se pomoću XML Schema dokumenta. S adrese <http://schemas.xmlsoap.org/ws/2004/03/addressing> moguće je dohvatiti XML Schema dokument *EndpointReferenceType.xsd* koji definira podatkovni tip *EndpointReferenceType*.

.NET radna okolina sadrži XSD alat koji omogućava generiranje XML Schema dokumenta iz dobro oblikovanog XML dokumenta. Alat generira XML Schema dokument *zaključivanjem* o pravilnostima u strukturi i vrsti podatkovnih tipova koji se nalaze u XML dokumentu. Također, moguće je iz XML Schema dokumenta generirati razred jezika C#. Generiranje datoteke *EndpointReferenceType.cs* koja sadrži istoimeni razred obavlja se sljedećim pozivom XSD alata:

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

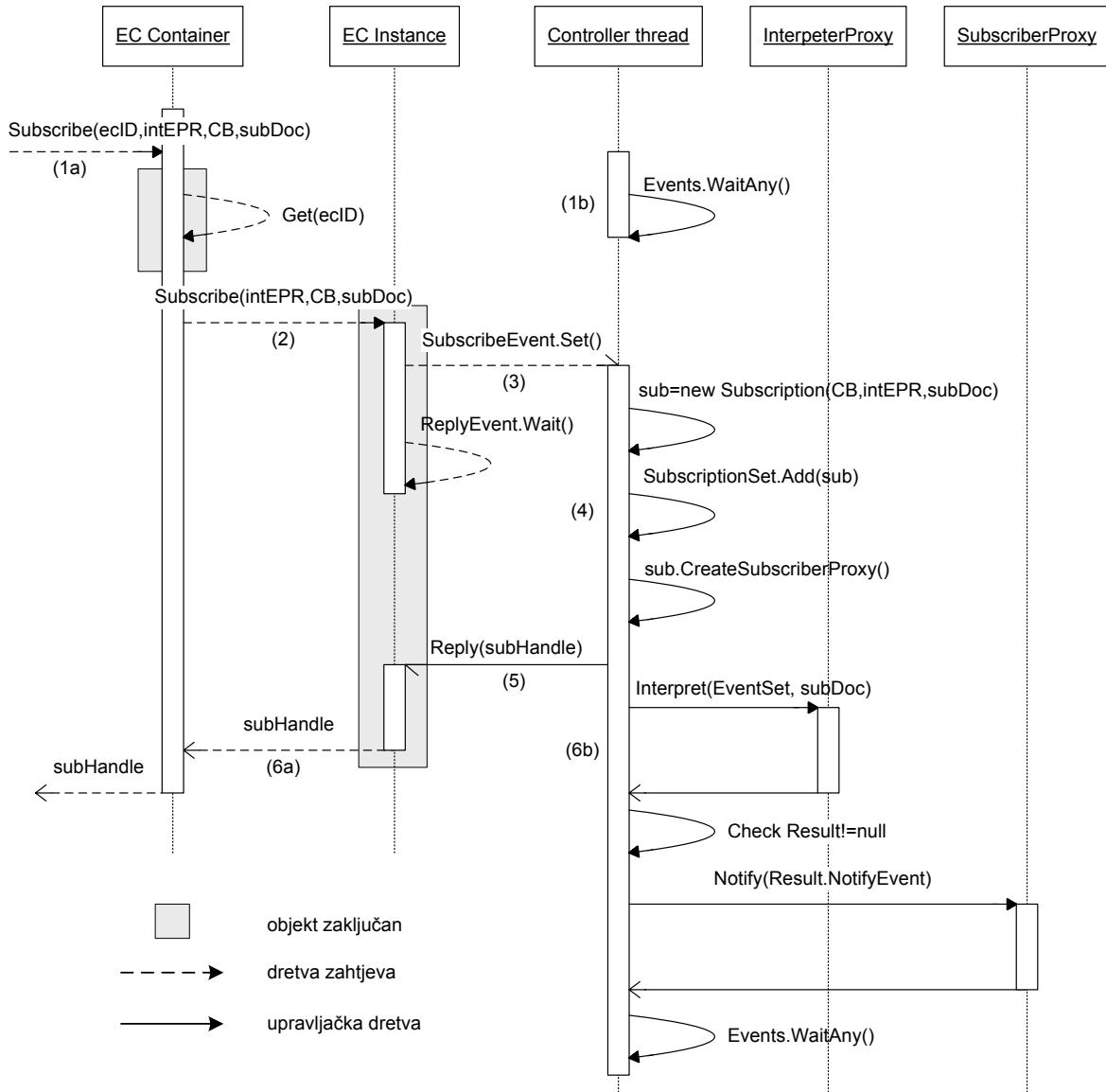
6.2.4. Spremnik usmjernika događaja

Spremnik usmjernika događaja je programski ostvaren razredom *EventChannelContainer*. Prilikom prvog pokretanja mrežne usluge usmjernika događaja instancira se jedan objekt razreda *EventChannelContainer* te se pohranjuje unutar *ASP.NET Application* objekta. Vrijeme života objekta *EventChannelContainer* prestaje zaustavljanjem rada ASP.NET podsustava. U tom trenutku provodi se čišćenje *Application* objekta pri čemu se uništava objekt razreda *EventChannelContainer*.

EventChannelContainer razred koristi objekt razreda *HashTable* za pohranjivanje objekata razreda *EventChannelInstance*. *HashTable* razred .NET radne okoline pruža učinkovit mehanizam *raspršenog adresiranja* pohranjenih objekata. Jednoznačni ključ za adresiranje objekata je identifikator instance usmjernika događaja (*resourceID*).

6.2.5. Instanca usmjernika događaja

Funkcionalnost instance usmjerenika događaja programski je ostvarena razredom *EventChannelInstance*. Pri svakom stvaranju nove instance usmjernika događaja stvara se novi objekt razreda *EventChannelInstance*. Svaka instanca usmjernika događaja sadrži zasebnu *upravljačku dretvu* koja izvodi komponentu *Upravljač*. Budući da ASP.NET radni proces stvara zasebnu *dretvu zahtjeva* koja služi za obradu svakog pristiglog zahtjeva te vraćanje odgovora pozivatelju, poželjno je što kraće vrijeme izvođenja dretve zahtjeva. Na slici 60 prikazana je obrada zahtjeva u slučaju poziva metode *Subscribe* za pretplaćivanje te način komunikacije *dretve zahtjeva* i *upravljačke dretve* kojim se postiže vremenski učinkovita obrada zahtjeva. Na početku scenarija dretva zahtjeva pristupa objektu *EventChannelContainer* (1a) koji je zaključan na kratko vrijeme pri dohvaćanju pripadnog *EventChannelInstance* objekta. Za to vrijeme upravljačka dretva je blokirana i čeka novi zahtjev (1b). Potom, dretva zahtjeva pristupa objektu *EventChannelInstance* koji se također zaključava za vrijeme obrade zahtjeva (2). Ako za to vrijeme istom *EventChannelInstance* objektu pokuša pristupiti nova dretva zahtjeva ona se blokira do završetka obrade prvog zahtjeva. Dretva zahtjeva obavještava upravljačku dretvu o pristizanju novog zahtjeva pomoću signalizacijskog objekta te predaje parametre poziva upravljačkoj dretvi (3).



Slika 60: Komunikacija dretve zahtjeva i upravljačke dretve u scenariju pretplaćivanja

Upravljačka dretva instance obavlja nužne akcije za oblikovanje odgovora (4) te predaje parametre odgovora dretvi zahtjeva (5). Nakon toga, dretva zahtjeva otključava *EventChannelInstance* objekt te šalje odgovor pretplatniku (6a), a upravljačka dretva obavlja ostale akcije koje su uzrokovane pristiglim zahtjevom (6b). U slučaju poziva metode *Subscribe*, poziva se interpretator za novoprstiglu pretplatu te se dojavljuje pretplatnik ako je interpretator generirao događaj.

6.2.6. Mehanizmi generiranja lokalnih zastupnika

Mehanizam statičkog generiranja lokalnih zastupnika

Lokalne zastupnike udaljenih mrežnih usluga moguće je generirati statički, odnosno za vrijeme razvoja programskog ostvarenja. Opis mrežne usluge WSDL dokumentom sadrži potrebne informacije za generiranje lokalnog zastupnika. Radna okolina pruža alat za generiranje razreda u jeziku C# koji predstavlja lokalnog zastupnika mrežne usluge smještene na određenoj adresi. Pomoću WSDL alata za zadani URL mrežne usluge usmjernika događaja generira se C# razred *EventChannelProxy* na sljedeći način:

```
WSDL http://www.ris.fer.hr/CoopetitionServices/EventChannel/EventChannel.asmx?WSDL  
/out:EventChannelProxy.cs
```

Generirani razred zastupnika *EventChannelProxy* izведен je iz razreda *SoapHttpClientProtocol* .NET radne okoline koji ostvaruje funkcionalnost HTTP korisnika, provodi serijalizaciju objekata u XML zapis te oblikuje SOAP poruke.

Mehanizam dinamičkog generiranja lokalnih zastupnika

Ako WSDL opis mrežne usluge nije dostupan za vrijeme razvoja programskog ostvarenja, onda se izvodljivi kôd generira dinamički, odnosno za vrijeme izvođenja. Mehanizam dinamičkog generiranja koristi se za generiranje izvodljivog kôda lokalnog zastupnika preplatnika. Iz predloška izvornog kôda zastupnika u jeziku C# i WSDL opisa sučelja stvara se potpuni izvorni kôd razreda koji ostvaruje funkcionalnost lokalnog zastupnika preplatnika. Izvorni kôd prevodi se za vrijeme izvođenja pomoću razreda *CSharpCodeProvider* i *ICodeCompiler* koji su sastavni dio .NET radne okoline. Rezultat prevodenja je objekt razreda *Assembly* koji sadrži izvodljivi kôd u jeziku CIL. Virtualni stroj .NET radne okoline izvodi CIL kôd sadržan u objektu *Assembly* prilikom instanciranja objekata i poziva metoda. *.NET Reflection* je mehanizam introspekcije koji omogućuje upravljanje izvodljivim kodom za vrijeme izvođenja. Mehanizam pruža mogućnost dohvatanja informacija o izvodljivom kôdu, odnosno o razredima, metodama i atributima koje izvodljivi kôd sadrži. Također, mehanizam omogućuje pokretanje izvodljivog kôda te korištenje istog za prevodenje i povezivanje.

6.2.7. Asinkrono pozivanje interpretatora i dojavljivanje pretplatnicima

Istovremeno pozivanje više udaljenih mrežnih usluga moguće je ostvariti stvaranjem zasebne dretve za svaki sinkroni poziv mrežne usluge. Međutim, .NET radna okolina pruža učinkovitu programsku potporu za asinkronu komunikaciju pa je u okviru programskog ostvarenja usmjernika događaja korišteno asinkrono pozivanje za postizanje istovremenog rada više mrežnih usluga. Pozivanje više mrežnih usluga na asinkroni način izvodi jedna dretva. U programskom ostvarenju instance usmjernika događaja pozivanje izvodi upravljačka dretva. Asinkronim pozivanjem mrežne usluge pozivajuća dretva se ne blokira za vrijeme čekanja na odgovor mrežne usluge. Budući da dretva nije blokirana do završetka rada mrežne usluge, nego nastavlja s radom nakon asinkronog poziva, postoje mnogi mehanizmi za određivanje završetka rada mrežne usluge, odnosno primjeka odgovora od mrežne usluge. Prilikom asinkronog pozivanja, korisnička strana koristi mehanizam *U/I vratiju završetka* (engl. *I/O completion port*) kojim se postiže učinkovito čekanje odgovora pozvane mrežne usluge. Nadalje, mrežna usluga ne zahtijeva dodatno konfiguriranje da bi bilo poduprto njen asinkrono pozivanje. Također, mrežna usluga ne zna da li je pozvana sinkrono ili asinkrono jer je poruka zahtjeva i poruka odgovora ista za oba načina pozivanja.

Uz svaku metodu za sinkrono pozivanje, razred lokalnog zastupnika mrežne usluge sadrži pripadne *Begin* i *End* metode za asinkrono pozivanje. Stoga uz metodu *Interpret* lokalnog zastupnika mrežne usluge interpretatora postoje dodatne metode *BeginInterpret* i *EndInterpret*. Asinkrono pozivanje mrežne usluge interpretatora odvija se u dva koraka. Prvo se poziva neblokirajuća metoda *BeginInterpret* koja pokreće poziv mrežne usluge, a u drugom koraku poziva se *EndInterpret* metoda koja završava poziv mrežne usluge. Metoda *EndInterpret* je blokirajuća, odnosno ako prilikom njezinog poziva odgovor još nije pristigao onda se metoda *EndInterpret* blokira. Pomoću razreda *WaitHandle* i metode *WaitAll* moguće je čekati na završetak više asinkrono pozvanih mrežnih usluga. Prilikom pozivanja više interpretatora istovremeno, za sve interpretatore poziva se neblokirajuća metoda *BeginInterpret*. Nakon toga pomoću metode *WaitAll* čeka se završetak svih poziva. Na kraju se za sve interpretatore poziva metoda *EndInterpret* koja vraća rezultat poziva mrežne usluge. Na sličan način programski je ostvareno dojavljivanje događaja pretplatnicima.

7. Zaključak

Većina današnjih objava/pretplata posredničkih sustava te sustava za izgradnju mreža zasnovanih na sadržaju razvijeni su pomoću posebnih protokola i oslanjaju se na određenu razvojnu platformu. Sustavi ne podupiru skup otvorenih standarda pa je onemogućen zajednički rad i suradnja različitih posredničkih sustava. Također, postojeći sustavi nisu primjenjivi za izgradnju raspodijeljenih sustava velikih razmjera koji koriste mrežu Internet. Nadalje, arhitektura postojećih sustava čvrsto je povezana, a komponente sustava ovise o razvojnoj platformi i programskom ostvarenju. Stoga je funkcionalnost sustava teško izmijeniti ili proširiti za potrebe određene primjene.

Razvojem računarstva zasnovanog na uslugama omogućena je izgradnja slabo povezanih raznorodnih raspodijeljenih sustava. Programske sustave izgrađuju se od usluga čije se povezivanje zasniva samo na osnovi njihovog sučelja i zbog toga je povezivanje potpuno neovisno o razvojnoj platformi. Ostvarenje arhitekture zasnovane na uslugama pomoću općeprihvaćenog *Web services* standarda potaknulo je mnoga istraživanja te brz razvoj i primjenu raspodijeljenih sustava zasnovanih na uslugama.

U ovom radu programski je ostvaren usmjernik događaja, objava/pretplata mehanizam s naprednim mogućnostima tumačenja sadržaja koje ostvaruju zasebni vanjski interpretatori. Arhitektura programskog ostvarenja zasnovana je na uslugama, a programsko ostvarenje razvijeno je kao mrežna usluga s održavanjem stanja prema *WS-Resource Framework* standardu. Stoga je usmjernik događaja neovisan o platformi i moguće ga je koristiti u okviru programskih sustava koji su izgrađeni prema WS-* skupu standarda. Povezivanjem više usmjernika događaja u mrežu omogućuje se razvoj raspodijeljenih sustava poticanih događajima, raspodijeljenih sustava zasnovanih na dokumentima te mreža zasnovanih na sadržaju. Ispitana je ispravnost programskog ostvarenja te je načinjena pripadna dokumentacija programskog ostvarenja. Proučena je metodologija i pripadne tehnologije za razvoj raspodijeljenih programskih sustava zasnovanih na uslugama.

Usmjernik događaja prilagodljiv je primjeni jer omogućava korisnicima određivanje razine svojstva izražajnosti i razine svojstva razmernog rasta ovisno o primjeni raspodijeljenog sustava. Svojstvo izražajnosti nije ograničeno razinom složenosti jezika koji se koristi za preplaćivanje i opisivanje događaja. Naime, vanjski interpretatori omogućuju napredno tumačenje objavljenog te generiranje novog sadržaja. Za tumačenje sadržaja, interpretatori koriste izražajnost i svojstva programskog jezika kojim je pojedini interpretator ostvaren. S druge strane, svojstvo razmernog rasta moguće je ostvariti povezivanjem proizvoljnog broja usmjernika događaja u mrežu. Također, korištenjem većeg broja interpretatora povećavaju se mogućnosti obrade sadržaja.

Literatura

- [1] Tanenbaum, Andrew S.: „**Computer Networks**”, Fourth Edition, Prentice Hall, New Jersey, 2003.
- [2] Tanenbaum, Andrew S., Maarten V. Steen: „**Distributed Systems: Principles and Paradigms**”, Prentice Hall, New Jersey, 2002.
- [3] Singh, Munindar P., Michael N. Huhns: „**Service-Oriented Computing: Semantics, Processes, Agents**”, John Wiley & Sons, London, 2005.
- [4] Singh, Munindar P., Michael N. Huhns: „**Service-Oriented Computing: Key Concepts and Principles**”, *IEEE Internet Computing*, siječanj/veljača 2005.
- [5] Mahmoud, Qusay H.: „**Middleware for Communications**”, John Wiley & Sons, London, 2004.
- [6] Short, S.: “**Building XML Web Services for the Microsoft .NET platform**”, Redmond, Washington, Microsoft Press, 2002.
- [7] Andro Milanović, Siniša Srbljić, Daniel Skrobo, Davor Čapalija, Saša Rešković: “**Coopetition Mechanisms for Service-Oriented Distributed System**”, članak prihvaćen za objavljivanje na *The 3rd International Conference on Computing, Communications and Control Technologies (CCCT '05)*, Austin, SAD, 2005.
- [8] Ivan Benc i ostali (pripremili): “**Exploring ICT frontiers: agents, quality of service and distributed computing**”, *proceedings of the Fourth Summer Camp 2004*, Ericsson Nikola Tesla i Fakultet elektrotehnike i računarstva, srpanj-rujan 2004.
- [9] Antonio Carzaniga, Alexander L. Wolf: “**Content-Based Networking: A New Communication Infrastructure**”, *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, listopad 2001.
- [10] S. Tai, I. Rouvellou: “**Strategies for integrating messaging and distributed object transactions**”, *In Middleware 2000, volume 1795 of Lecture Notes in Computer Science*, Springer-Verlag, 2000.
- [11] Nadia Busi, Cristian Manfredini, Alberto Montresor, Gianluigi Zavattaro: “**Towards a Data-Driven Coordination Infrastructure for Peer-to-Peer Systems**”, *NETWORKING 2002 Workshops on Web Engineering and Peer-to-Peer Computing*, 19. – 24. svibanj, 2002.
- [12] Giovanni Russello, Michel Chaudron, Maarten van Steen: “**Exploiting Differentiated Tuple Distribution in Shared Data Spaces**”, *Proc. Int'l Conf. on Parallel and Distributed Computing (Euro-Par)*, Pisa, Italija, rujan 2004.
- [13] Ivana Podnar: “**Service Architecture for Content Dissemination to Mobile Users**”, disertacija, veljača 2004., Zagreb
- [14] Ivan Skuliber: “**Raspodijeljene aplikacije javnog informacijskog sustava**”, magistarski rad, 2003., Zagreb
- [15] D. S. Rosenblum, A. L. Wolf: “**A design framework for Internet-scale event observation and notification**”, *Proceedings of the ESEC/FSE '97 - 6th European Software Engineering Conference*, Zurich, Švicarska, Rujan, 1997.
- [16] Peter R. Pietzuch: “**Event-Based Middleware: A New Paradigm for Wide-Area Distributed Systems?**”, *6th CaberNet Radicals Workshop, Funchal, Madeira Island, Portugal*, veljača 2002.

- [17] Antonio Carzaniga, David S. Rosenblum, Alexander L. Wolf: “**Design and evaluation of a wide-area event notification service**”, *ACM Transaction on Computer Systems*, 2001.
- [18] P. Th. Eugster, P. Felber, R. Guerraoui, A.-M. Kermarrec: „**The Many Faces of Publish/Subscribe**” Technical report, EPFL Lausanne, 2001.
- [19] Peter R. Pietzuch, Jean Bacon: “**Hermes: A Distributed Event-Based Middleware Architecture**”, *Proceedings of the 22nd International Conference on Distributed Computing Systems*, 2002.
- [20] Peter R. Pietzuch, Jean Bacon: „**Peer-to-peer overlay broker networks in an event-based middleware**”, *Proceedings of the 2nd international workshop on Distributed event-based systems*, 2003.
- [21] M. Castro, P. Druschel, A. Kermarrec, A. Rowstron: “**SCRIBE: A large-scale and decentralized application-level multicast infrastructure**”, *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.
- [22] Michael Franklin, Stanley Zdonik: “**A framework for scalable dissemination-based systems**”, *Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Atlanta, Georgia, SAD, 1997.
- [23] IBM T. J. Watson Research Center: “**Gryphon: Publish/subscribe over public networks**”, *White paper*, 1999.
- [24] Bill Segall, David Arnold, Julian Boot, Michael Henderson, Ted Phelps: ”**Content Based Routing with Elvin4**”, *Proceedings AUUG2K*, Canberra, Australija, lipanj 2000.
- [25] Antonio Carzaniga: “**Architectures for an Event Notification Service Scalable to Wide-area Networks**”, disertacija, 1998.
- [26] G. Cugola, E. Di Nitto, A. Fuggetta: “**The JEDI event-based infrastructure and its application to the development of the OPSS WFMS**”, *IEEE Transactions on Software Engineering, Volume 27, Issue 9*, rujan 2001.
- [27] M. Hapner, R. Burridge, R. Sharma, J. Fialli, K. Stout: “**Java Message Service**”, Sun Microsystems Inc., travanj 2002.
- [28] Peter R. Pietzuch, Brian Shand, Jean Bacon: “**A Framework for Event Composition in Distributed Systems**”, *4th ACM/IFIP/USENIX International Conference on Middleware (Middleware'03)*, Rio de Janeiro, Brazil, lipanj 2003.,
- [29] M. Altinel, M. Franklin: “**Efficient Filtering of XML Documents for Selective Dissemination of Information**”, *In Proc. of VLDB*, rujan 2000.
- [30] J. Clark, S. DeRose: “**XML Path Language (XPath) Version 1.0**”, *W3C Recommendation*, <http://www.w3.org/TR/xpath>, studeni 1999.
- [31] OMG: “**CORBA Notification Service Specification**”, kolovoz 2002.
- [32] P.T. Eugster, R. Guerraoui: “**Content-Based Publish/Subscribe with Structural Reflection**”, *In Proceedings of the 6th Usenix Conference on Object-Oriented Technologies and Systems (COOTS'01)*, siječanj 2001.
- [33] M. P. Papazoglou, W. J. van den Heuvel: “**Service-Oriented Computing: State-of-the-Art and Open Reserach Issues**”, *TI/RS/2003/123 (SOBI-1/D2.3)*, Telematica Instituut, Enschede, Nizozemska, 2003.
- [34] M. P. Papazoglou, D. Georgakopoulos: “**Service-oriented computing: Introduction**”, *Communications of the ACM, Volume 46, Issue 10*, listopad 2003.
- [35] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana: “**Unraveling the web services web: An introduction to SOAP, WSDL, UDDI**”, *IEEE Internet Computing*, ožujak-travanj 2002.

- [36] “**Web Service Conceptual Architecture (WSCA 1.0)**”, *IBM Technical White Paper*, <http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>, svibanj 2001.
- [37] T. Bray, J. Paoli, C. M. Sperberg-McQueen: “**Extensible markup language (XML) 1.0. Recommendation (Third Edition)**”, *W3C*, <http://www.w3.org/TR/REC-xml>, veljača 2004.
- [38] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, et al.: “**Simple Object Access Protocol (SOAP) 1.1**”, *W3C Note*, <http://www.w3.org/TR/SOAP>, svibanj 2000.
- [39] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana: “**Web Services Description Language (WSDL) 1.1**”, *W3C Note*, <http://www.w3.org/TR/wsdl>, ožujak 2001.
- [40] T. Bellwood, et al: “**UDDI Version 3.0.2 Published specification**”, http://uddi.org/pubs/uddi_v3.htm, listopad 2004.
- [41] D.C. Fallside, et al: “**XML Schema Part 0: Primer Second Edition**”, *W3C*, www.w3.org/TR/xmlschema-0/, listopad 2004.
- [42] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, S. Weerawarana: “**The next step in web services**”, *Communications of the ACM, volume 46, number 10*, listopad 2003.
- [43] Karl Czajkowski, Donald F. Ferguson, Ian Foster, Jeffrey Frey i ostali: “**The WS-Resource framework**”, *Globus Alliance*, <http://www.globus.org/wsrf>, siječanj 2004.
- [44] Foster, I., Frey, J., Graham, S., Tuecke i ostali: “**Modeling Stateful Resources with Web Services**”, Globus Alliance, 2004.
- [45] Chris Peltz: “**Web Services Orchestration and Choreography**”, *IEEE Computer, Volume 28, Number 10*, 2003.
- [46] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland i ostali: “**Business Process Execution Language for Web Services (BPEL4WS) 1.1**”, svibanj 2003.