SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 85

SUSTAV PRIKUPLJANJA I OBJEDINJAVANJA PODATAKA U RAČUNALNOM OBLAKU ZA POTROŠAČKO PROGRAMIRANJE

Goran Narančić

Zagreb, lipanj 2010.

Tekst diplomskog zadatka

Zahvaljujem prof. dr. sc. Siniši Srbljiću na prilici za rad na zanimljivom projektu.
Zahvaljujem mag.ing. Goranu Delaču, mr.sc. Dejanu Škvorcu, mr.sc. Miroslavu Popoviću i mag.ing. Ivanu Žužaku na pomoći i strpljenju tijekom izrade ovog rada.
Hvala obitelji i prijateljima na potpori.

Sadržaj

1. Uvod

Globalna računalna mreža Internet najveći je postojeći raspodijeljeni sustav, složen od velikog broja različitih monolitnih i raspodijeljenih sustava povezanih zajedničkom komunikacijskom infrastrukturom. Raspodijeljeni računalni sustav definira se kao sustav sastavljen od skupa nezavisnih računala kojemu korisnik pristupa kao jednom koherentnom sustavu [1]. Raspodijeljeni sustavi imaju mnoge prednosti nad monolitnim sustavima, ali i unose dodatne zahtjeve koji se ne pojavljuju kod monolitnih sustava. Neki od ključnih zahtjeva za pravilan rad raspodijeljenih računalnih sustava su ostvarenje komunikacije među elementima sustava i održavanje konzistencije podataka na različitim elementima. Prikupljanje i organizacija podataka među dijelovima sustava jedan je od zahtjeva specifičnih za raspodijeljene računalne sustave.

Sustav za potrošačko programiranje omogućava potrošaču da bez poznavanja osnovnih elemenata programskih jezika izgradi programski sustav s funkcionalnošću koja zadovoljava njegove potrebe. Sučelje sustava za potrošačko programiranje prikazuje dijelove funkcionalnosti formirane u obliku komponenata, koje potrošač spaja i uređuje u sustav sa traženom funkcionalnosti. Sustav za potrošačko programiranje na taj način skriva programske detalje i konstrukte od potrošača, zaobilazeći potrebu za znanjem programiranja. Primjer sustava za potrošačko programiranje je sustav *Geppeto* [2].

Računalni oblak za potrošačko programiranje sastoji se od poslužitelja za komponente sustava za potrošačko programiranje i potrošača koji stvaraju, mijenjaju i koriste te komponente. Poslužitelj za komponente sustava za potrošačko programiranje služi za mrežno objavljivanje komponenata. Potrošači upravljaju komponentama sustava za potrošačko programiranje objavljenim na poslužiteljima. Pretraživanje komponenata sustava za potrošačko programiranje na razini računalnog oblaka ostvaruje se *sustavom za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje*, koji je definiran u ovom diplomskom radu. Primjer komponenti sustava za potrošačko programiranje su udomljenici poput *Yahoo! Widgets* i *Google Gadgets*.

Sustav za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje je raspodijeljeni sustav za upravljanje i centralizirano pohranjivanje opisa udomljenika. Sudionici računalnog oblaka preuzimaju uloge definirane u *sustavu za prikupljanje i objedinjavanje podataka*. Uloge opisuju njihova prava i odgovornosti u sklopu sustava. Moguće uloge su imenik, poslužitelj za udomljenike, te korisnik. Imenik je centralizirani, mrežno dostupni repozitorij opisa komponenata sustava za potrošačko programiranje, jedinstven u računalnom oblaku. Imenik omogućava pretraživanje opisa udomljenika. Poslužitelj za udomljenike u sklopu *sustava za prikupljanje i objedinjavanje podataka* obavještava imenik slanjem poruka o promjenama u opisu udomljenika koje mrežno objavljuje. Sudionik računalnog oblaka s ulogom korisnika u sklopu *sustava za prikupljanje i objedinjavanje podataka* ima pravo pretraživanja imenika.

U sklopu diplomskog rada definirana je arhitektura *sustava za prikupljanje i objedinjavanje podataka* i protokol *raspodijeljenog prijavljivanja i pretraživanja udomljenika* (*RPPU*). Opisano je programsko ostvarenje imenika *sustava za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje* te su opisane korištene tehnologije. Opisane su osnove raspodijeljenih računalnih sustava i računalnih oblaka, te je iznesen pregled tehnologije *Java Servlet* korištene za programsko ostvarenje imenika.

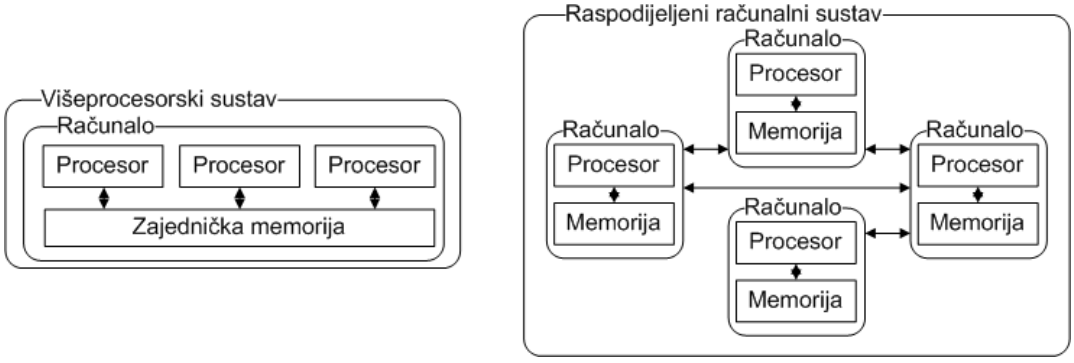
U drugom poglavlju opisani su raspodijeljeni računalni sustavi i koncept računalnog oblaka, te je dan pregled *Java Servlet* tehnologije. U trećem poglavlju opisana je arhitektura *sustava za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje*. U četvrtom poglavlju opisan je protokol *RPPU* definiran za potrebe komunikacije unutar sustava. Peto poglavlje sadrži opis programskog ostvarenja imenika *sustava za prikupljanje i objedinjavanje podataka*.

2. Pregled korištenih tehnologija

Tijekom izrade diplomskog rada korištene su tehnologije i koncepti izneseni u ovom poglavlju. *Sustav za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje* je raspodijeljeni računalni sustav koji čini dio infrastrukture računalnog oblaka. Raspodijeljeni računalni sustavi opisani su u poglavlju 2.1. U poglavlju 2.2 opisan je koncept računalnog oblaka. U sklopu diplomskog rada ostvaren je imenik *sustava za prikupljanje i objedinjavanje podataka* koristeći tehnologiju *Java Servlet*, koja je opisana u poglavlju 2.3.

2.1. Raspodijeljeni računalni sustavi

Raspodijeljeni računalni sustavi su sustavi koji spajaju više različitih računalnih sredstava u jednu cjelinu radi učinkovitijeg korištenja i lakšeg pristupa. Primjer raspodijeljenog računalnog sustava je globalna računalna mreža Internet. Globalna računalna mreža Internet sastoji se od velikog broja računala, koji komuniciraju zajedničkim jezikom, protokolom HTTP, povezujući sredstva poput Internet stranica i elektroničke pošte s korisnicima. Sredstvo u raspodijeljenom sustavu može predstavljati podatak, procesorsko vrijeme, memorijsku lokaciju ili druge oblike programskih ili sklopovskih elemenata [1].

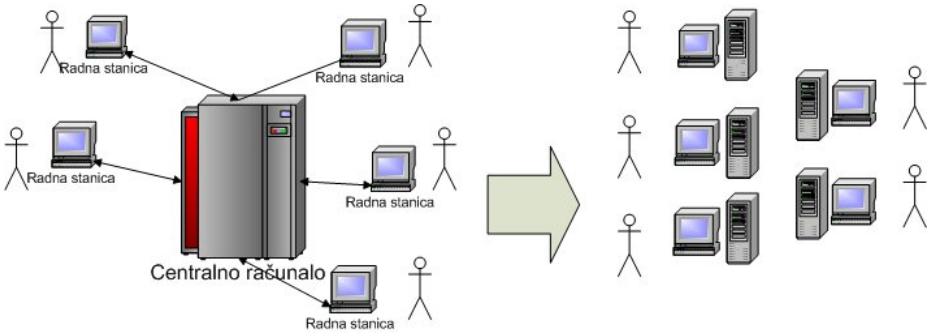


Slika 2.1 Višeprocesorski i raspodijeljeni sustavi

Raspodijeljeni računalni sustav definira se na više načina. Jedna od definicija raspodijeljenog računalnog sustava glasi: raspodijeljeni računalni sustav je skup nezavisnih računala kojima korisnik pristupa kao jednom koherentnom sustavu [1]. Druga definicija glasi: sustav u kojem programske i

sklopovske komponente umreženih računala komuniciraju i usklađuju svoje aktivnosti isključivo razmjenom poruka [3]. Definicije ističu ključnu osobinu raspodijeljenih sustava, koja ih razlikuje od paralelnih (višeprocesorskih) sustava. Raspodijeljen sustav je višeračunalni sustav, odnosno sustav koji se sastoji od dva ili više nezavisnih računala koja međusobno komuniciraju, ali ne dijele sklopovlje. Višeprocesorski sustavi dijele dio sklopovlja, dok su procesori međusobno neovisni [1]. Na slici 2.1 prikazana je arhitektura višeprocesorskih i raspodijeljenih sustava.

Razvoj raspodijeljenih računalnih sustava su omogućila dva napretka u području računalnih sustava. Razvoj brzih mikroprocesora i njihova relativno niska cijena omogućila je prelazak organizacija sa centraliziranih računala s radnim stanicama za korisnike na individualno računalo za svakog korisnika. Slika 2.2 prikazuje promjenu sustava unutar organizacija. Svaki korisnik posjeduje svoje računalo s odvojenom memorijom i procesorskim sustavom, čije su sposobnosti usporedive s prijašnjim centraliziranim računalom [1].



Slika 2.2 Prijelaz s centralnog na individualna računala

Drugi napredak u području računalnih sustava je bio ostvarenje prvih računalnih mreža velikih brzina. Lokalne računalne mreže (engl. *local area network*, LAN) služe za komunikaciju između računala u podružnici tvrtke ili ustanove, omogućavajući prijenos velikih količina podataka između računala u relativno kratkom roku. Računalne mreže širokog područja (engl. *wide area network*, WAN) omogućavaju spajanje milijuna računala za prijenos podataka, iako s manjoj brzinom nego u lokalnim računalnim mrežama [1].

Raspodijeljeni računalni sustav dijeli karakteristike, ali se razlikuje od raspodijeljenih i od mrežnih operacijskih sustava. Raspodijeljeni operacijskih sustav (engl. *distributed operating system*, DOS) namijenjen je

višeprocesorskim i homogenim višeračunalnim sustavima. Zadaća raspodijeljenog operacijskog sustava je pružanje usluge upravljanja sklopovljem i istovremenog skrivanja oblika sklopovlja. Raspodijeljeni operacijski sustav opisuje se kao čvrsto povezan (engl. *tightly-coupled*) jer održava jedinstven pogled na čitav sustav [1].

Mrežni operacijski sustav (engl. *network operating system*, NOS) je slabo povezan (engl. *loosely-coupled*) operacijski sistem namijenjen heterogenim višeračunalnim sustavima povezanim lokalnom računalnom mrežom ili mrežom širokog područja. Glavna zadaća mrežnog operacijskog sustava je pružanje lokalnih usluga udaljenim korisnicima. Mrežni operacijski sustav ostvaruje se kao skup odvojenih operacijskih sustava koji djeluju zajedno u svrhu ponude zajedničkog skupa usluga korisnicima [1].

Raspodijeljeni računalni sustav je sustav posredničkog sloja koji nadograđuje mrežni operacijski sustav sa distribucijskom prozirnošću. Mrežni operacijski sustav nudi pristup svim sredstvima neovisno o lokaciji, ali ne skriva od korisnika lokaciju sredstva. Raspodijeljeni računalni sustav zasniva se na mrežnim operacijskim sustavima, te pruža posredničke usluge korisnicima skrivajući pritom lokacije sredstva koji se koriste. Raspodijeljeni računalni sustav djeluje kao posrednik između operacijskih sustava koji upravljaju sredstvima i aplikacijama, te prikriva raspodijeljenost sredstava po računalima i omogućava suradnju sustava i uređaja [1].

Jedna od bitnih osobina raspodijeljenih sustava je heterogenost. Raspodijeljeni računalni sustavi ne ovise o tipu, obliku ili ostvarenju komponente sustava, nego samo o sposobnosti komponente da se uključi u sustav. Komponente sustava mogu biti jednake, ali mogu biti i napravljene od različitih proizvođača prema različitim principima, zahtjevima i ciljevima. Raspodijeljeni računalni sustav veže heterogene komponente u jednu cjelinu, nudeći usluge različitih oblika i principa u jednom obliku [1].

Raspodijeljeni računalni sustavi mogu se opisati sa pet obilježja: paralelne aktivnosti u sustavu, komunikacija razmjenom poruka, dijeljenje sredstava, nepoznavanje globalnog stanja i nepostojanje globalnog takta. Paralelna aktivnost u raspodijeljenom sustavu odnosi se na akcije koje nezavisne

komponente sustava istovremeno provode. Raspodijeljeni sustavi ne posjeduju zajedničku memoriju koja povezuje komponente sustava, stoga se komunikacija odvija razmjenom poruka. Komponente sustava zajedno pristupaju sredstvima dostupnim u sustavu. Globalno stanje sustava je kombinacija stanja svake komponente sustava, no niti jedna komponenta ne poznaje globalno stanje. Nezavisnost komponenti onemogućava uspostavljanje globalnog takta, što zahtijeva korištenje mehanizama za vremensko usklađivanje različitih komponenata u sustavu [3].

Zahtjevi na raspodijeljene računalne sustave definiraju osnovu na kojoj se arhitektura sustava gradi. Raspodijeljeni računalni sustavi služe za povezivanje korisnika sa sredstvima radi povećanja učinkovitosti sredstava, razmjene informacija i prevladavanja razlike u geografskim lokacijama. Prozirnost, otvorenost i razmjerni rast predstavljaju osnovne zahtjeve na raspodijeljene sustave [1].

2.1.1. Prozirnost

Zahtjev za prozirnost proizlazi iz definicije raspodijeljenih sustava: korisnik vidi jedan koherentan sustav koji nudi niz usluga, a ne skupinu sredstava postavljenih na različita računala u mreži. Prozirnost raspodijeljenih sustava podijeljena je u 8 vrsta: prozirnost pristupa, lokacijska prozirnost, migracijska prozirnost, prozirnost premještanja, replikacijska prozirnost, konkurencijska prozirnost, prozirnost na kvar i prozirnost perzistencije [1].

Prozirnost pristupa (engl. *access transparency*) predstavlja prikrivanje različitosti pristupa i zapisa istovjetnih sredstava na različitim računalima. Na primjer, različita računala mogu koristiti različite operacijske sustave i stoga različite konvencije imenovanja sredstava. Drugi primjer je korištenje *little endian* na jednom i *big endian* načina zapisa podataka na drugom računalu. Raspodijeljeni sustav s ostvarenom prozirnosti pristupa prikriva razlike i omogućuje korisnicima pristup sredstvima na jednak način, bez obzira na različitost u načinu imenovanja, pristupa ili zapisa na različitim komponentama sustava [1].

Lokacijska prozirnost (engl. *location transparency*) predstavlja prikrivanje lokacije sredstva kojeg korisnik koristi. Imenovanje igra ključnu ulogu u

lokacijskoj prozirnosti. Korištenje logičkih imena je jedan način postizanja lokacijske prozirnosti [1]. Primjer lokacijske prozirnosti ostvarene korištenjem logičkih imena su *URL* adrese sredstava na globalnoj računalnoj mreži Internet, poput *URL*-a *http://www.fer.hr/*. Lokacijska prozirnost je usko vezana sa migracijskom prozirnosti i prozirnosti premještanja [1].

Migracijska prozirnost (engl. *migration transparency*) predstavlja prikrivanje promjene lokacije sredstva. Raspodijeljeni sustav koji ostvaruje migracijsku prozirnost prikriva činjenicu da se lokacija sredstva u sustavu promijenila [1]. Primjer bi bio premještanje poslužitelja koji objavljuje stranicu na *URL*-u *http://www.fer.hr/* iz Zagreba u Split bez promjene *URL*-a stranice. Korisnik i dalje na isti način pristupa sredstvu, iako je sredstvo promijenilo lokaciju [1].

Prozirnost premještanja (engl. *relocation transparency*) predstavlja prikrivanje promjene lokacije sredstva za vrijeme korištenja sredstva. Uvjet prozirnosti premještanja je stroži uvjet od migracijske i lokacijske prozirnosti [1]. Primjer prozirnosti premještanja je pristup u stvarnom vremenu snimki kamere postavljene na vozilo koje se kreće ulicama. Prozirnost premještanja zahtijeva da se način pristupa i veza s kamerom ne prekida ili mijenja bez obzira na kretanje kamere.

Replikacijska prozirnost (engl. *replication transparency*) predstavlja prikrivanje postojanja više preslika istih podataka. Podaci se repliciraju radi povećanja dostupnosti podataka, smanjenja vremena pristupa ili povećanja učinkovitosti sustava. Radi ostvarenja replikacijske prozirnost, nužno je da sve preslike podataka imaju isto ime, što zahtijeva ostvarenje lokacijske prozirnosti, ili bi bilo nemoguće pristupiti replikama na različitim lokacijama [1].

Konkurencijska prozirnost (engl. *concurency transparency*) predstavlja prikrivanje istovremenog pristupa dva ili više korisnika istom sredstvu. Konkurencijska prozirnost proizlazi iz obilježja raspodijeljenog sustava o dijeljenju sredstava. Pri ostvarenju konkurencijske prozirnosti nužno je osigurati da istovremeno korištenje jednog sredstva od dva ili više korisnika ostavi sredstvo u konzistentnom stanju [1]. Primjer konkurencijske prozirnosti je pristupanje dva korisnika istoj datoteci smještenoj na poslužitelj. Održanje

konzistentnog stanja može se ostvariti korištenjem blokirajućih mehanizama ili ostvarenjem transakcija [1].

Prozirnost na kvar (engl. *failure transparency*) predstavlja prikrivanje kvarova tijekom rada sustava. Prikrivanje kvarova podrazumijeva sprječavanje spoznaje korisnika da se kvar dogodio, kao i oporavak od kvara. Raspodijeljeni sustav s ostvarenom prozirnosti na kvar bi trebao spriječiti da korisnik primijeti kako sredstvo koje koristi je prestalo raditi [1]. Primjer prozirnosti na kvar je preusmjeravanje korisnika na alternativni poslužitelj nakon što je prvi poslužitelj prestao raditi. Određivanje i prikrivanje kvarova jedan je od najtežih problema u ostvarenju raspodijeljenih sustava [1].

Perzistencijska prozirnost (engl. *persistence transparency*) predstavlja prikrivanje lokacija podataka s obzira na trajnu ili privremenu memoriju. Korisnik ne smije znati da li objekt nad kojim je tražio izvođenje operacije se nalazio u privremenoj memoriji računala, ili je operacijski sustav prvo morao preseliti objekt iz trajne u privremenu memoriju, pa tek onda izveo operaciju. Perzistencijska prozirnost jednako je važna za monolitne kao i raspodijeljene sustave [1].

Prozirnost raspodijeljenih računalnih sustava potrebno je uravnotežiti s učinkovitošću sustava i zahtjevima na ostvarenje [1]. Primjer nepoželjne lokacijske prozirnosti je pri putovanjima. Primjenski sustav na prijenosnom računalu je postavljen da pripremi vremensku prognozu za trenutni dan. Kada se lokacijska prozirnost ne bi strogo održavala, primjenski sustav bi mogao sam otkriti trenutnu lokaciju i pripremiti prikladnu vremensku prognozu. Učinkovitost sustava također može biti smanjena zbog prozirnosti. Na primjer, dvije komponente dijele način zapisa drukčiji od standardnog u sustavu. Pri strogom održavanju prozirnosti pristupa, pretvorba podataka iz jednog zapisa u drugi će se provesti dva puta. Ukoliko bi komponente mogle doznati da dijele način zapisa, pretvorba podataka ne bi morala biti provedena.

2.1.2. Otvorenost

Otvorenost raspodijeljenog sustava definira se kao mjera u kojoj je raspodijeljeni sustav moguće proširiti ili zamijeniti dijelove sustava, odnosno koliko je lako dodati nova sredstva i ponuditi nove usluge u sklopu

raspodijeljenog sustava. Usluge u raspodijeljenim sustavima definiraju se preko sučelja, i ta sučelja moraju biti javno dostupna radi zadovoljavanja zahtjeva otvorenosti [4].

Otvoreni raspodijeljeni računalni sustav je raspodijeljeni sustav koji nudi usluge prema normiranim pravilima koja opisuju sintaksu i semantiku usluga. Primjer normiranih pravila su definicije protokola za komunikacija u računalnim sustavima [1]. Norme su zastupane od strane normizacijskog tijela (*de jure* standard), ili su široko prihvaćene u industriji (*de facto* standard). Norme su vlasnički neovisne, dobro definirane i javno dostupne [3].

Pravilno definirana sučelja omogućuju povezivanje komponenata sustava napravljenih potpuno neovisno te zamjenu komponenata bez potrebe za promjenom programskih ostvarenja drugih komponenata. Općenito, sučelja formalno definiraju sintaksu, no semantika predstavlja problem pri formalnoj definiciji, stoga se u praksi često opisuje prirodnim jezikom. Primjer su opisi objekta. Parametri i imena javnih funkcija objekta definiraju se u formalnom zapisu, ali značenja parametara i funkcija, kao i čitavog objekta, opisani su prirodnim jezikom u neformalnom obliku [1].

Pravilno definirana sučelja su potpuna i neutralna. Potpuna definicija sučelja sadrži sve informacije potrebne za pravilno ostvarenje tražene funkcionalnosti. Potpunu definiciju sučelja teško je postići, stoga često definicije sučelja sadrže informacije koje su ovisne o ostvarenju, ali pružaju dodatni uvid na razini sučelja. Neutralna definicija sučelja ne definira programsku logiku ostvarenja. Potpunost i neutralnost su ključne za međudjelovanje i prenosivost [1].

Otvorenost u raspodijeljenom sustavu preduvjet je za tri osobine komponenata sustava: međudjelovanje, prenosivost i proširivost. Međudjelovanje (engl. *interoperability*) predstavlja mjeru u kojoj dva (ili više) sustava ili komponenti različitih ostvarenja mogu obavljati zadatke zajedno oslanjajući se isključivo na međusobne usluge definirane zajedničkim sučeljem [1]. Prenosivost (engl. *portability*) predstavlja mjeru u kojoj programsko ostvarenje namijenjeno raspodijeljenom sustavu *A* se može izvršavati, bez promjena, na drukčijem raspodijeljenom sustavu *B*, koja ostvaruje jednaka

sučelja kao i *A* [1]. Proširivost (engl. *extensibility*) predstavlja mjeru u kojoj je sustav moguće proširiti novim komponentama ili zamijeniti postojeće komponente bez utjecanja na komponente koje čine sustav [1].

Prilagodljivost (engl. *flexibility*) raspodijeljenog sustava je osobina koja nadograđuje otvorenost. Prilagodljivost otvorenog raspodijeljenog sustava predstavlja mjeru u kojoj je moguće izgraditi raspodijeljeni sustav od različitih komponenata koje ostvaruju zajednički skup sučelja, bez obzira na njihov izvor i oblik programskog ostvarenja. Prilagodljiv raspodijeljeni sustav je proširiv s komponentama koje su ostvarene na drukčijem računalnom sustavu, te podržava zamjenu komponenata s novima ostvarenim u drugom programskom jeziku. Prilagodljiv, otvoren raspodijeljeni sustav je teško ostvarljiv cilj [1].

Ključna metoda za izgradnju prilagodljivih raspodijeljenih sustava je organizacija sustava od skupine relativno malih, lako zamjenjivih i prilagodljivih komponenti. Pri uporabi ove metode, potrebno je definirati ne samo sučelja visoke razine koja vide korisnici i primjenski sustavi, nego i sučelja za interne dijelove sustava. Sučelja internih dijelova služe za opis rada sustava i omogućavaju nadogradnju i održavanje raspodijeljenog sustava kao cjeline. Razlog za zamjenu ili nadogradnju komponente sustava često je komponenta koja pruža mehanizam baziran na principu koji nije prikladan, obično zbog specifičnih potreba korisnika raspodijeljenog sustava [1].

Odvajanje principa sustava od mehanizama rada omogućava postavljanje globalnih principa sustava koji se mogu zaobići ako se pojavi potreba. Na primjer, lokacijska prozirnost važan je princip na razini sustava, no u specifičnim slučajevima ne odgovara potrebama korisnika. Rješenje problema u ovom primjeru leži u pružanju mogućnosti da se lokacijska prozirnost zaobiđe u slučaju potrebe. Princip lokacijske prozirnosti se održava na razini raspodijeljenog sustava, ali primjenski sustav korisnika koji putuje sadrži mehanizam postavljen da zaobiđe princip i dozna fizičku lokaciju [1].

2.1.3. Razmjerni rast

Porast u broju korisnika i opsegu zadataka koji se postavljaju pred računalne sustave postavlja problem pred pojedine sustave. Raspodijeljene računalne sustave je moguće napraviti proširivima sa novim korisnicima i

sredstvima, no postavlja se problem razmjernog rasta sustava. Nova sredstva, ili nove instance postojećih sredstava, ne pružaju samo povećane mogućnosti sustava, nego također postavljaju upravljački problem. Razmjerni rast raspodijeljenog sustava predstavlja osobinu prilagođavanja sustava promjenama u opsegu zadaće sustava [1].

Razmjerni rast raspodijeljenog sustava može se izraziti u tri različite dimenzije. Razmjerni rast prema veličini sustava predstavlja mogućnost sustava da prihvati nove korisnika i/ili sredstava bez značajnog utjecanja na učinkovitost sustava. Druga dimenzija razmjernog rasta je razmjerni rast prema geografskoj lokaciji. Raspodijeljeni sustav može imati dijelove i korisnike koji se nalaze na udaljenim lokacijama, i potrebno je uzeti u obzir koliko ta udaljenost utječe na učinkovitost sustava. Upravljački razmjerni rast je treća dimenzija razmjernog rasta raspodijeljenog računalnog sustava. Upravljački razmjerni rast predstavlja razinu lakoće upravljanja raspodijeljenim sustavom bez obzira na složenost sustava, koji može pokrivati mnoge neovisne organizacije. Raspodijeljeni računalni sustav koji može razmjerno rasti u jednoj ili više dimenzija svejedno može gubiti na učinkovitosti kako razina skale sustava raste [1].

Pri ostvarenju razmjernog rasta raspodijeljenog sustava nailazi se na granice i probleme koje je potrebno riješiti ili uzeti u obzir. Razmjerni rast prema veličini sustava može biti ograničena centraliziranim elementom sustava, kojeg iz određenog razloga nije moguće ili prihvatljivo raspodijeliti. Tri moguća centralizirana elementa sustava su usluge, podaci i algoritmi [1].

Primjer centraliziranih usluga sustava mogu biti usluge koje se nalaze na točno jednom poslužitelju u sklopu sustava. S rastom skale sustava, poslužitelj će u jednom trenutku postati usko grlo, zbog razine komunikacije, ograničene procesorske moći, ograničenog skladišnog prostora ili nekog drugog specifičnog elementa. Jednostavno rješenje bi bilo raspodijeliti ili replicirati usluge na drugim poslužiteljima, ali to nije uvijek moguće. Na primjer, poslužitelj može služiti kao mrežno sučelje bankovnog sustava kojeg zbog održanja sigurnosti nije dopustivo replicirati i proširiti na više lokacija [1].

Primjer centraliziranih podataka u raspodijeljenom sustavu je jedna tablica s podacima o svim *URL*-ovima koji čine globalnu računalnu mrežu Internet,

umjesto *DNS* sustava kakav se danas koristi. Poslužitelj koji bi održavao tablicu bi bio preopterećen brojem zahtjeva za razrješavanje *URL* adresa koji bi stizali na njega. Posljedica toga bi bila usporenje globalne računalne mreže Internet ispod razine iskoristivosti [1].

Centralizirani algoritmi predstavljaju treći problem pri ostvarivanju razmjernog rasta raspodijeljenih računalnih sustava. Primjer centraliziranog algoritma je skupljanje informacija o povezanosti raspodijeljenog sustava, izračun optimalnih tablica usmjeravanja pomoću algoritma iz teorije grafova i potom prosljeđivanja izračunatih tablica svakom elementu sustava. Čak i ako se uračuna da komponenta može izračunati optimalne tablice u prihvatljivom vremenu i da se oblik mreže neće promijeniti, razina komunikacije potrebna za izvršavanja algoritma bi zagušila barem dio sustava. Ovaj primjer može se poopćiti na uputu da se svaki algoritam koji prikuplja informacije sa svih elemenata sustava, izvršava na jednom računalu i onda obavještava sve elemente o rezultatima treba izbjegavati. U raspodijeljenim sustavima treba primjenjivati decentralizirane algoritme [1].

Decentralizirani algoritmi dijele četiri glavne osobine koje ih razlikuju od centraliziranih algoritama. Prvo, decentralizirani algoritmi pretpostavljaju da niti jedan element nema potpune podatke o globalnom stanju sustava. Druga osobina decentraliziranih algoritama je oslanjanje isključivo na lokalne informacije pri donošenju odluke. Treće, kvar jednog elementa ne kvari ili prekida izvršenje algoritma kao cjeline. Četvrta glavna osobina decentraliziranog algoritma je nepostojanje pretpostavke da postoji globalni takt u sustavu, odnosno da su elementi usklađeni u brzini izvršavanja dijelova algoritma [1].

Geografski razmjerni rast uzrokuje specifičnu skupinu problema. Glavni problem pri geografskom razmjernom rastu nastaje pri korištenju sinkrone komunikacije. Sinkrona komunikacija označava način komunikacije gdje zahtjevatelj šalje poruku s zahtjevom i prekida svoj rad dok ne dobije odgovor. U lokalnim mrežama sinkrona komunikacija ne stvara značajne probleme jer se zanemarivo malo vremena gubi na prijenosu zahtjeva i odgovora. Povećanjem skale udaljenosti, kao kod mreža širokog područja, gubitak vremena za

komunikaciju značajno raste i ovisno o sustavu može uzrokovati neprihvatljivo kašnjenje [1].

U lokalnim računalnim mrežama komunikacija je pouzdana i brza. Nadalje, postoje protokoli za istovremeno odašiljanje poruka većem broju primatelja. Na primjer, otkrivanje koji element pruža određenu uslugu u lokalnim mrežama izvršava se korištenjem višeodredišnog pošiljanja (engl. *broadcast*), što olakšava izgradnju raspodijeljenih sustava. U mrežama širokog područja, komunikacija nije pouzdana i brza, te je potrebno ostvariti specijalizirane usluge koje služe kao imenici koji omogućavaju pretraživanje i pronalazak potrebnih usluga, kao zamjenu za višeodredišno odašiljanje poruka. Ostvarenje tih imenika uzrokuje dodatne specifične probleme razmjernog rasta, koji se mora riješiti kako bi imenici bili u stanju ponuditi uslugu velikom broju korisnika [1].

Geografski razmjerni rast i razmjerni rast prema veličini sustava blisko su povezani s problemima koje uzrokuju centralizirana rješenja. Centralizirana rješenja uzrokuju probleme u razmjernom rastu prema veličini zbog inherentnih uskih grla koje donose u sustav, te postavljaju ograničenja za geografski razmjerni rast zbog kašnjenja i nepouzdanosti komunikacije na većim udaljenostima. Također, centralizirana rješenja na velikim područjima uzrokuju trošenje mrežnih sredstava jer poruke moraju putovati veće udaljenosti nego je potrebno zbog udaljenosti centraliziranog sustava [1].

Upravljanje raspodijeljenim sustavom koji se pruža preko većeg broja nezavisnih domena koje mogu imati različite, pa čak i suprotne principe korištenja, upravljanja i sigurnosti sredstava. Općenito, korisnici vjeruju komponentama i politici administratora koji pripadaju istoj domeni kao i korisnici, no to povjerenje se ne pruža preko granica domene, iako obje domene su dio jednog raspodijeljenog sustava. Nadalje, pri uključivanju novih domena u raspodijeljeni sustav, dvosmjerne sigurnosne mjere se moraju postaviti. Raspodijeljeni sustav mora zaštiti svoje stare domene od nepoželjnih napada iz nove domene. S druge strane, nova domena također mora postaviti mehanizme zaštite sebe od raspodijeljenog sustava. Na primjer, novi korisnici mogu biti ograničeni samo na pravo čitanja, ali ne i mijenjanja datoteka iz raspodijeljenog sustava. Nadalje, novi korisnici mogu biti ograničeni ili potpuno spriječeni u pristupu skupim sredstvima poput procesora velikih performansi [1].

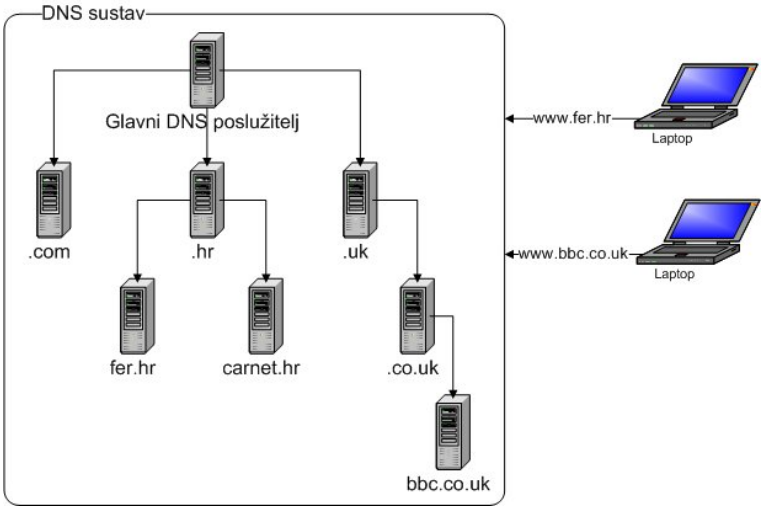
Rješenja za probleme razmjernog rasta ovise o ciljevima sustava, ostvarenjima drugih komponenata sustava i ograničenjima nametnutim drugim aspektima raspodijeljenog sustava. Skrivanje kašnjenja izazvanih komunikacijom, razdjeljivanje i replikacija su tri općenite metode koje mogu pomoći pri rješavanju, ili barem skrivanju problema razmjernog rasta od korisnika [1].

Skrivanje kašnjenja izazvanih komunikacijom (engl. *hiding communication latencies*) ne rješava problem, ali skriva ga od korisnika što je u nekim slučajevima dovoljan rezultat. Osnovna ideja leži u držanju korisnikove strane zaposlenom dok se čeka odgovor na zahtjev. Na primjer, korisnik se može proslijediti na sljedeći obrazac dok se čeka odgovor o ispravnosti prethodnog. U osnovi, sinkrona komunikacija se zamjenjuje asinkronom komunikacijom, gdje pošiljatelj zahtjeva nastavi obavljati druge dijelove posla dok se čeka odgovor na prethodni zahtjev. Asinkrona komunikacija može biti simulirana sinkronom komunikacijom, prepuštajući procesor drugom zadatku pokrenutom u nezavisnoj dretvi dok glavna dretva čeka odgovor [1].

Postoje slučajevi gdje korištenje asinkrone komunikacije, izravno ili neizravno, nije prikladno. Na primjer, ako korisnik mora pravilno ispuniti obrazac prije nego može preći na sljedeći korak. U tom slučaju, skrivanje kašnjenja zbog komunikacije može se provesti prebacujući dio ili čitav programski kôd korisniku, tako da se najveći dio posla obavlja na korisničkoj strani i smanjuje potreba za komunikacijom. Primjer je ispunjavanje obrazaca, gdje se provjera sintakse unesenih podataka može obaviti na korisničkoj strani, te se potom šalje samo jedna poruka sa provjereno dobrim obrascem, koji potom poslužitelj obrađuje i sprema [1].

Razdjeljivanje (engl. *distribution*) je tehnika razmjernog rasta kojom se zadaća jedne komponente podijeli u skup manjih zadataka. Svaki od manjih zadataka obavlja jedna, nova, komponenta, koje se razdijele po sustavu i stoga smanji opterećenje na pojedinu komponentu. Primjer tehnike razdjeljivanja je *Domain Name System* (*DNS*) sustav, prikazan na slici 2.3. *DNS* sustav je podijeljen u hijerarhiju poslužitelja zaduženih za razrješavanje *URL* adresa poslužitelja. *URL* adresa je strukturirana logička adresa određenog poslužitelja. Zahtjev za razrješavanje se izvršava iterativno. Korisnik šalje zahtjev za

razrješavanjem koji sadrži adresu. *DNS* poslužitelj vraća *IP* adresu ako je poznaje, ili adresu *DNS* poslužitelja niže razine koji je zadužen za sljedeći element primljene *URL* adrese. Proces se ponavlja dok se ne odredi *IP* adresa poslužitelja.



Slika 2.3 DNS sustav kao primjer razmjernog rasta tehnikom razdjeljivanja

Replikacija (engl. *replication*) služi za smanjivanje kašnjenja zbog komunikacije te razrješavanja uskih grla sustava. Replikacija je smještanje preslike podataka na jedno ili više novih lokacija unutar raspodijeljenog sustava, čime se povećava dostupnost podataka, balansira opterećenje komponenata, te u slučaju geografski raspršenih sustava, smanjuje udaljenost između korisnika i izvora podataka, a time i kašnjenje zbog komunikacije [1].

Privremeno skladištenje podataka (engl. *caching*) je podvrsta replikacije gdje se stvara preslika podatka općenito bliže korisniku koji je zatražio podatak. Glavna razlika u odnosu na replikaciju je u izvoru odluke za stvaranjem preslike. Privremeno skladištenje podataka ovisi o korisnicima u sustavu, dok replikacija ovisi o vlasniku podataka. Primjer privremenog skladištenja je privremeno skladištenje Internet stranica, pri čemu mrežni preglednici pri ponovnom pristupu stranici ne šalju novi zahtjev, nego prikazuju ranije primljenu stranicu [1].

Replikacija i privremeno skladištenje podataka, rješavajući jedan skup problema razmjernog rasta, stvaraju novi problem. Konzistencija (engl. *consistency*) podataka odnosi se na održavanje preslika podataka jednakima.

Promjene načinjene na jednoj preslici podataka potrebno je prenijeti na sve druge preslike [1].

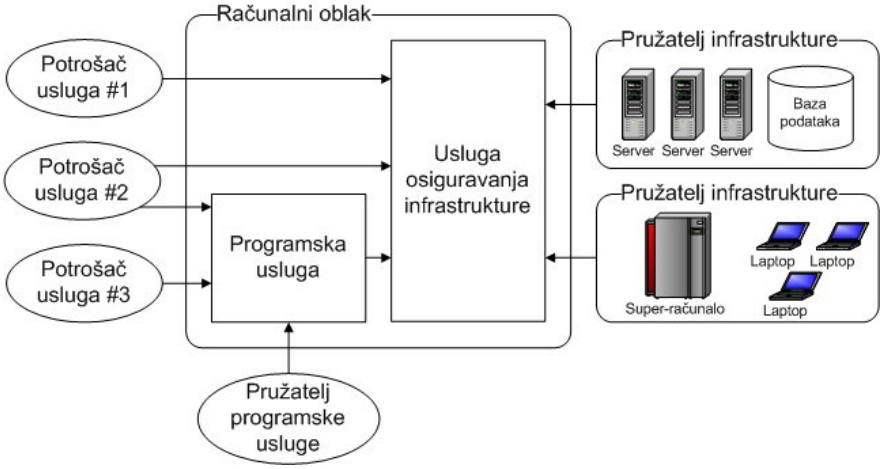
Razina ozbiljnosti problema konzistencije podataka može od nebitnog pa do ključnog. Razlike u promjeni prosječne Internet stranice u roku od nekoliko minuta rijetko su neophodne, no promjene u podacima pri elektronskoj trgovini dionicama su ključne i moraju biti što brže prenesene do svih korisnika. Problem također leži i u višestrukim promjenama podataka sa različitih izvora, stoga je nužno promijeniti sve preslike podataka jednakim redoslijedom, što zahtijeva neki oblik globalnog takta. Globalni takt je vrlo teško, ili često i nemoguće, ostvariti na način kojim može razmjerno rasti. Posljedično, rješavanje problema razmjernog rasta replikacijom može unijeti nove probleme koji inherentno nemaju svojstvo razmjernog rasta [1].

2.2. Računalni oblak

Računalni oblak je novi, zasad slabo definiran, termin u području računarstva. Oblak kao pojam u računarstvu koristi se kao metafora za globalnu računalnu mrežu Internet i potječe od ranih prikaza mreže kao promjenjivog skupa neovisnih sredstava. Pojam računalni oblak je naziv za novi model pružanja usluge iznajmljivanja računalne infrastrukture, no oblik i granice tog modela još nisu precizno definirani [5].

Definicija računalnog oblaka, sastavljena na temelju 22 definicije drugih autora i iznesena u članku [5], definira oblak kao veliki skup lako iskoristivih i dostupnih virtualiziranih sredstava (poput sklopovlja, platformi i usluga). Sredstva koja sudjeluju u oblaku moguće je dinamički prilagoditi promjeni razmjera korištenja, što omogućava dobru optimizaciju korištenja sredstava. Skup sredstava uobičajeno je dostupan preko plati-po-korištenju (engl. *pay per use*) modela naplate, gdje jamstva osigurava pružatelj infrastrukture (engl. *infrastructure provider*, *IP*) preko prilagođenih ugovora o kakvoći usluge (engl. *service level agreement*, *SLA*). Autori članka [5] također su postavili minimalnu definiciju računalnog oblaka kao raspodijeljenog sustava koji posjeduje tri osobine: razmjerni rast (engl. *scalability*), model uporabe plati-po-korištenju i virtualizacija sredstava (engl. *virtualization*).

Tri strane sudjeluju u paradigmi računalnog oblaka. Pružatelji infrastrukture osiguravaju infrastrukturu za izvršavanje programskih usluga, što čini osnovni skup usluga računalnog oblaka [5]. Pružatelji programskih usluga (engl. *service providers*, *SP*) koriste uslugu osiguravanja infrastrukture kako bi objavili i izvršavali svoje usluge [5]. Potrošači usluga (engl. *service users*) koriste usluge ponuđene u sklopu računalnog oblaka [5]. Na slici 2.4 prikazan je primjer korištenja računalnog oblaka. Pružatelji infrastrukture nude korištenje svoje infrastrukture kroz uslugu osiguravanja infrastrukture. Preko usluge osiguravanja infrastrukture, pružatelji i potrošači usluga mogu zatražiti određene sposobnosti infrastrukture koja se virtualizacijom odvaja od ostale infrastrukture. Pružatelj programskih usluga koristi infrastrukturu kako bi ponudio vlastitu uslugu, npr. mrežni tekst procesor, na korištenje potrošačima usluga. Potrošač usluga #1 koristi samo uslugu osiguravanja infrastrukture, npr. za pohranjivanje privatnih podataka na mrežno dostupnu lokaciju. Može se reći da potrošač usluga #1 pruža uslugu mrežnog objavljivanja podataka sam sebi. Potrošač usluga #2 koristi i uslugu osiguravanja infrastrukture, npr. za izvršavanje složenog algoritma, te uslugu mrežnog procesora teksta. Potrošač usluga #3 koristi samo uslugu mrežnog procesora teksta.



Slika 2.4 Primjer računalnog oblaka

Gledano sa strane uporabe, računalni oblak može pružati usluge u tri oblika: infrastruktura kao usluga, programsko okruženje kao usluga i programski sustav kao usluga [5].

Infrastruktura kao usluga (engl. *Infrastructure as a Service*, *IaaS*) predstavlja pružanje usluga računalnih sredstava poput procesorskog vremena i

memorijskih lokacija. Pomoću virtualizacije, sredstva se mogu aktivno mijenjati prema potrebama potrošača, proširujući ili smanjujući dodijeljene mogućnosti. Usluge računalnih sredstava pružaju pružatelji infrastrukture, a koriste pružatelji usluga [5].

Programsko ostvarenje kao usluga (engl. *Platform as a Service*, *PaaS*) je nadogradnja infrastrukture kao usluge, gdje se pruža programsko okruženje kao usluga, podižući razinu apstrakcije za jednu razinu. Virtualizirana infrastrukura se skalira prema potrebi i skriva od korisnika programskog okruženja [5]. Primjer programskog okruženja kao usluge je *Google Apps Engine*, kao i *Programmable Internet Enviroment* (*PIE*).

Programski sustav kao usluga (engl. *Software as a Service*, *SaaS*) predstavlja najvišu razinu apstrakcije. Programski sustavi su dostupni kao usluge na globalnoj računalnoj mreži Internet, zamjenjujući lokalno pokretane programske sustave. Programske sustave ponuđene kao usluge proizvođač može aktivno održavati za sve potrošače i naplaćivati njihovu upotrebu na jednostavan i balansiran način. Također, programski sustav ponuđen kao usluga je dostupan na svakom računalu bez potrebe za postavljanjem, što pruža višestruke prednosti za potrošače kao i za proizvođača. Primjer takvog programskog sustava je *Google Docs* [5, 6].

2.2.1. Obilježja računalnih oblaka

Računalni oblaci opisani su skupom od devet obilježja. Ključna obilježja računalnih oblaka su virtualizacija, razmjerni rast i ekonomski model plati-po-korištenju. Preostala obilježja su dijeljenje sredstava, heterogenost, sigurnost, vidljivost programskog okruženja, lakoća korištenja i standardizacija [5].

Obilježje virtualizacije (engl. *virtualization*) predstavlja funkcionalnost sustava da spoji ili podijeli fizička sredstva poput računalne snage i memorijskog prostora u virtualna računala s fizičkim značajkama koje točno odgovaraju potrebama usluga. Princip automatske prilagodbe sredstava omogućava automatsku provedbu razmjernog rasta (engl. *scalability*) za pojedine usluge, smanjujući sredstva dostupna usluzi s smanjenjem broja potrošača usluge i povećavajući dodijeljena sredstva s povećanjem broja potrošača. Automatska prilagodba sustava također čini model plati-po-

korištenju (engl. *pay-per-use*) prikladnim, omogućujući pružatelju usluga da plati točno za infrastrukturu koja je bila potrebna [5].

Obilježje dijeljenja sredstava (engl. *resource sharing*) odnosi se na razinu dijeljenja sredstava između različitih programskih usluga. Računalni oblaci osiguravaju sredstva na zahtjev pružatelja usluga, koristeći virtualizaciju da pruže dojam jedinstvenog sredstva. Izolacija sredstava nastala kao posljedica virtualizacije sprječava dijeljenje sredstava među različitim uslugama [5].

Heterogenost (engl. *heterogeneity*) predstavlja objedinjavanje heterogenih programskih i fizičkih sredstava, koje je implicitno podržano u računalnom oblaku [5].

Obilježje sigurnosti (engl. *security*) u računalnim oblacima riješeno je kao posljedica virtualizacije. Izolacija sredstava nastala virtualizacijom pruža sigurnost na razini korisnika pružajući svakom korisniku jedinstven pristup dodijeljenim virtualiziranim sredstvima [5].

Vidljivost programskog okruženja (engl. *platform awareness*) odnosi se na utjecaj programskog okruženja i računalne arhitekture na ostvarenje usluga. Računalni oblaci virtualizacijom inherentno skrivaju računalnu arhitekturu od usluga. Kao posljedica, usluge mogu biti razvijane neovisno od računalnog oblaka na kojemu će biti izvršavane [5].

Lakoća korištenja (engl. *usability*) odnosi se na obilježje računalnih oblaka da skrivaju detalje objavljivanja usluga od potrošača, olakšavajući postavljanje i korištenje usluga [5].

Obilježje standardizacije (engl. *standardization*) predstavlja razinu u kojoj su sučelja dijelova računalnih usluga standardizirana. Sučelja prema korisnicima koriste standardizacije razvijene za druge sustave, poput spleta računala, no standardizacija unutarnjih dijelova sustava je na niskoj razini [5].

2.2.2. Usporedbe računalnog oblaka s drugim računalnim modelima

Model računalnog oblaka uspoređuje se s uslužnim računalstvom, spletom računala, te u određenoj razini s starim modelom centraliziranog računala s radnim stanicama.

Računalni oblak u usporedbi s uslužnim računalstvom (engl. *utility computing*) definira se na sljedeći način: računalni oblak omogućava potrošačima i razvijateljima da koriste usluge bez znanja, stručnosti i kontrole nad infrastrukturom na kojoj se one izvršavaju. Uslužno računalstvo, s druge strane, osigurava infrastrukturu na zahtjev koju se može kontrolirati, prilagođavati i skalirati prema potrebi [7].

Splet računala (engl. *grid computing*) definira se kao raspodijeljena računalna okolina koja omogućava usklađeno dijeljenje heterogenih i geografski raspršenih sredstava kao što su računalna snaga, spremnički prostor, mrežna propusnost, aktuatori te osjetila [3]. Ciljevi spleta računala i računalnih oblaka su jednaki: smanjiti troškove korištenja računala i povećanje prilagodljivosti i pouzdanosti korištenjem sredstava ponuđenih od treće strane [5].

Spletovi računala i računalni oblaci jednaki su po pitanju heterogenosti, no razlikuju se u pristupu i ostvarenju obilježja virtualizacije, razmjernog rasta, dijeljenja sredstava, sigurnosti, vidljivosti programskog okruženja, lakoće korištenja, standardizacije, ekonomskog modela, usluga visoke razine, toka programskih sustava i ugovora o kakvoći usluga [5].

Spletovi računala omogućavaju virtualizaciju skupa sredstava u jedno sredstvo, pomoću sučelja koja skrivaju heterogenost sredstava iza tih sučelja. Računalni oblaci proširuju mogućnosti virtualizacije na skup fizičkih sredstava, omogućavajući stvaranje virtualnih fizičkih sredstava [5].

Razmjeran rast spleta računala ostvaruje se povećanjem broja elemenata u spletu, dok računalni oblaci koriste aktivnu promjenu dodijeljenih virtualnih fizičkih sredstava. Oba modela skrivaju probleme razmjernog rasta od proizvođača usluga [5].

Model računalnog oblaka i model spleta računala imaju suprotan pristup obilježju dijeljenja sredstava. Računalni oblak inherentno sprječava dijeljenje sredstva među uslugama, dok splet računala koristi dijeljenje sredstva među organizacijama za povećanje učinkovitosti [5].

Sigurnost u spletu računala ovisi o ostvarenju i nije riješena na razini modela, dok računalni oblaci inherentno ostvaruju obilježje sigurnosti kao posljedicu virtualizacije [5].

Model spleta računala zahtijeva razvoj programskih sustava namijenjenih izvođenju na spletu računala, postavljajući vidljivost programskog okruženja i računalne arhitekture na vrlo visoku razinu. Računalni oblaci, zahvaljujući virtualizaciji, imaju nisku razinu vidljivosti programskog okruženja i računalne arhitekture [5].

Model računalnog oblaka predviđa dobro definirana sučelja i skrivanje unutarnjih dijelova sustava, što olakšava korištenje usluga od strane potrošača. Model spleta računala je složen i zahtijeva visoku razinu upravljanja od strane potrošača, što otežava korištenje usluga [5].

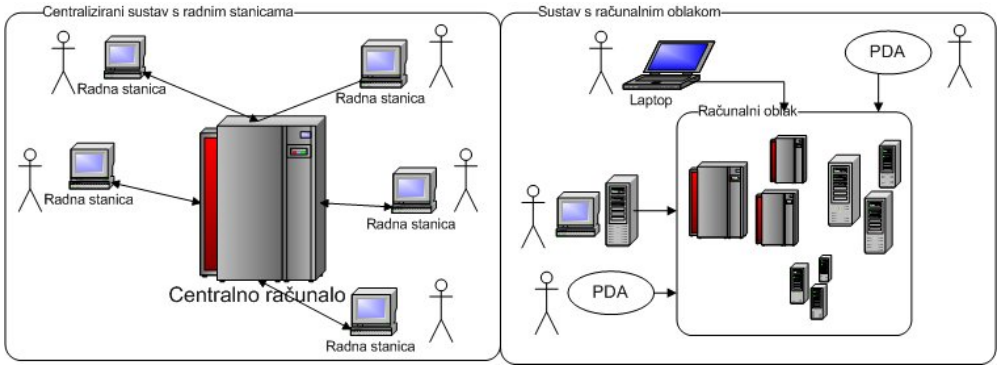
Model spleta računala je dobro standardiziran u sučeljima. Model računalnog oblaka koristi standardizaciju namijenjenu spletovima računala za vanjska sučelja, no unutarnja sučelja nisu standardizirana [5].

Većina financiranja spletova računala dolazila je iz javnih izvora, dok su računalni oblaci prvenstveno komercijalno financirani. Spletovi računala često ostvaruju plaćanje fiksnom ratom po usluzi ili dijeljenju sredstava među organizacijama. Računalni oblaci su orijentirani na model plati-po-korištenju [5].

Usluge visoke razine (engl. *high level services*) predstavljaju implicitni skup usluga ponuđenih u sklopu računalnih oblaka. Model spleta računala već ostvaruje neke usluge više razine poput pretrage metapodataka. Model računalnog oblaka nema definirane usluge viših razina [5].

Tok programskih sustava (engl. *software workflow*) odnosi se na potrebu usklađivanja različitih usluga na razini sustava. Model računalnog oblaka usmjeren je prema postavljanju usluga na zahtjev i stoga nema zahtjeva za koordinacijom različitih usluga. Model spleta računala, usmjeren prema uslugama i poslovima, zahtijeva usklađivanje lokacija i toka programskih sustava [5].

Ugovori o kakvoći usluge (engl. *quality of service*) predstavljaju razinu u kojoj sustav dopušta osiguravanje kakvoće pruženih usluga. Računalni oblaci često imaju inherentno ugrađene mehanizme za osiguranje kakvoće usluga. Spletovi računala u općenitom pogledu nisu usmjereni na osiguravanje kakvoće usluga zbog orijentiranosti na dijeljenje sredstava između organizacija [5].



Slika 2.5 Usporedba računalnog oblaka i sustava s radnim stanicama

Usporedba računalnog oblaka sa sustavom s centraliziranim, snažnim računalom kojemu se pristupa preko radnih stanica zasnovana je na paradigmi korištenja. Sustav s radnim stanicama posjedovao je jedno, snažno računalo koje je imalo veliki broj radnih stanica za unos i prikaz podataka. Centralno računalo izvršava sve programe i služi za pohranu podataka, dok radne stanice služe za komunikaciju i upravljanje centralnim računalom. Podacima spremljenim u centralno računalo može se pristupiti s bilo koje radne stanice, uz pravu autorizaciju.

Model korištenja računalnog oblaka sastoji se od potrošača usluga koji koriste usluge za obavljanje zadataka i spremanje podataka. Podaci spremljeni na oblak dostupni su sa bilo kojeg uređaja sposobnog za spajanje u računalnu mrežu. Računalni oblak stoga može služiti za usklađivanje podataka na različitim uređajima, no i korištenju slabih uređaja za pokretanja složenih aplikacija i rad s podacima bez trajnog spremanja na pristupni uređaj. Pri pogledu s strane korištenja, računalni oblak je povratak centraliziranom računalnom sustavu sa radnim stanicama globalnog dometa i većim sredstvima na raspolaganju [8].

2.3. *Java Servlet*

Java Servlet tehnologija služi za modularno ostvarenje funkcionalnosti poslužitelja. *Servlet* je razred definiran u programskom jeziku *Java* koji služi za proširenje funkcionalnosti poslužitelja koji objavljuje programska ostvarenja kojima se pristupa koristeći komunikacijski model zahtjev-odgovor [9]. Alternativna definicija *Servlet* objekta glasi: *Servlet* je mali programski sustav

koji se izvršava u sklopu mrežnog poslužitelja [10]. *Java Servlet* tehnologija posjeduje mogućnost odgovaranja na bilo koji oblik zahtjeva, no najčešće su korišteni *Servlet* objekti koji obrađuju *HTTP* zahtjeve [9]. *Servlet* objekti izvršavaju se u sklopu udomitelja (engl. *container*) koji je dio mrežnog poslužitelja. *Apache Tomcat* je primjer poslužitelja otvorenog kôda koji ostvaruje *Java Servlet* i *Java Server Pages* tehnologije [11].

Prednosti *Java Servlet* tehnologije su učinkovitost, prenosivost, niska cijena i ugrađena funkcionalnost u odnosu na *CGI* (engl. *Common Gateway Interface*) tehnologiju [12]. Poslužitelj pri pokretanju stvara objekt na temelju *Java Servlet* razreda kao jednu dretvu, što je učinkovitije nego stvaranje novog procesa. Postojeća ostvarenja poslužitelja i korištenje programskog jezika *Java* omogućuju prenosivost na različite računalne arhitekture i operacijske sustave, te pružaju veliki skup ugrađene funkcionalnosti [12].

Java Servlet razredi moraju ostvarivati sučelje *Servlet*. Apstraktni razred *GenericServlet* ostvaruje sučelje *Servlet* i pruža osnovnu funkcionalnost *Java Servlet* objektima. Apstraktni razred *HttpServlet* nadjačava razred *GenericServlet* i proširuje osnovnu funkcionalnost specijaliziranim funkcijama za obradu *HTTP* zahtjeva [9].

U poglavlju 2.3.1 je opisano sučelje *Servlet*. Apstraktni razred *GenericServlet* opisan je u poglavlju 2.3.2, a apstraktni razred *HttpServlet* u poglavlju 2.3.3.

2.3.1. Sučelje *Servlet*

Sučelje *Servlet* definira funkcije životnog ciklusa *Servlet* objekta, te pomoćne funkcije *getServletConfig* i *getServletInfo*. Izvođenje *Java Servlet* objekta na poslužitelju uvjetovano je ostvarivanjem sučelja *Servlet*. Općenito, objekt nadjačava apstraktne razrede *GenericServlet* ili *HttpServlet*, i time neizravno ostvaruje sučelje *Servlet* [10].

Slika 2.6 Životni ciklus *Java Servlet* objekta

Životni ciklus *Servlet* objekta prikazan je na slici 2.6. Životni ciklus *Servlet* objekta počinje pozivom konstruktora, čime se stvara objekt, ali se ne aktivira. Objekt nije mrežno dostupan dok poslužitelj ne pozove funkciju *init* [10].

Poslužitelj poziva funkciju *init* da obavijesti objekt kako je stavljen u službu. Funkcija prima *ServletConfig* objekt s početnim podacima. Funkcija *init* služi za postavljanje početnih vrijednosti *Servlet* objekta i pripremanje objekta za primanje zahtjeva [9]. Ako funkcija *init* ne završi uspješno, poslužitelj smatra objekt nespremnim za stavljanje u službu i objekt ne postaje mrežno dostupan [10].

Funkciju *service* poslužitelj poziva za svaki primljeni zahtjev i čini centralni dio životnog ciklusa *Servlet* objekta. Funkcija prima *ServletRequest* i *ServletResponse* objekte. *ServletRequest* objekt sadrži primljeni zahtjev, koji se obrađuje i priprema odgovor koji se šalje koristeći primljeni *ServletResponse* objekt. Za svaki primljeni zahtjev poslužitelj poziva funkciju *service* i premješta izvođenje funkcije u novu dretvu. Moguće je istovremeno izvođenje više funkcija *service* istog *Servlet* objekta [10].

Poslužitelj poziva funkciju *destroy* kao obavijest *Servlet* objektu da je povučen iz službe. Funkcija *destroy* pruža priliku objektu da zapiše podatke koji trebaju ostati sačuvani, oslobodi rezervirane memorijske lokacije i završi svoj rad. Poslužitelj zove funkciju *destroy* točno jednom nakon što se završe sve pokrenute funkcije *service*, i poslije poziva funkcije *destroy* više ne poziva funkciju *service* tog *Servlet* objekta [10].

Pomoćna funkcija *getServletConfig* služi za dohvaćanje objekta *ServletConfig* primljenog u funkciji *init*. Objekt *ServletConfig* sadrži početne parametre *Servlet* objekta. Programsko ostvarenje sučelja mora se pobrinuti za pravilno spremanje *ServletConfig* objekta radi uspješnog izvršenja funkcije *getServletConfig* [10].

Pomoćna funkcija *getServletInfo* služi za dohvaćanje podataka o *Servlet* objektu, poput autora, verzije i autorskih prava, i vraća niz znakova. Preporučeni zapis podataka je u obliku običnog teksta [10].

2.3.2. Apstraktni razred *GenericServlet*

Apstraktni razred *GenericServlet* ostvaruje funkcionalnost općenitog *Servlet* objekta, neovisnog od komunikacijskih protokola. Ostvaruje sučelja *Servlet* i *ServletConfig*. Razred *GenericServlet* služi kao nacrt za izradu *Servlet* objekta, pružajući osnovna, jednostavna ostvarenja funkcija *init*, *destroy* i funkcija definiranih *ServletConfig* sučeljem. Objekt koji nadjačava razred *GenericServlet* mora nadjačati samo funkciju *service* da bi bio spreman za postavljanje na poslužitelj [13].

Konstruktor razreda *GenericServlet* ne obavlja specifične akcije, pored stvaranja novog objekta. Funkcija *init* postavlja početne parametre i obavlja normalne zadatke konstruktora [13].

Funkcija *destroy* služi za obavještavanje *Servlet* objekta da ga poslužitelj povlači iz upotrebe, kao što je definirano u sučelju *Servlet* [13].

Objekt *ServletConfig* sadrži inicijalizacijske parametre *Servlet* objekta. Pristup tim parametrima se pojednostavljuje funkcijama *getInitParameter* i *getInitParameterNames*, zahtijevanim sučeljem *ServletConfig*. Funkcija *getInitParameter* prima niz znakova s imenom parametra i vraća niz znakova s vrijednosti parametra, odnosno vrijednost *null* ako parametar ne postoji. Funkcija *getInitParameterNames* vraća listu s imenima svih parametara prisutnih u *ServletConfig* objektu [13].

Funkcija *getServletConfig* vraća *ServletConfig* objekt primljen u funkciji *init*, kao što je definirano u sučelju *Servlet* [13].

Funkcija *getServletContext* vraća pokazivač na *ServletContext* objekt. Funkcija je poziv funkcije *getServletContext ServletConfig* objekta i postavljena je radi jednostavnosti pristupa. Funkcija *getServletContext* zahtijevana je sučeljem *ServletConfig* [13].

Funkcija *getServletInfo* vraća podatke o *Servlet* objektu, kao što je definirano u sučelju *Servlet*. Ukoliko nije nadjačana, funkcija vraća prazan niz znakova [13].

Funkcija *getServletName* vraća ime *Servlet* objekta. Ime *Servlet* objekta definirano je od administracije poslužitelja ili među podacima o objavljivanju *Servlet* objekta. Ukoliko ime nije definirano, koristi se naziv *Servlet* razreda Funkcija *getServletName* zahtijevana je sučeljem *ServletConfig* [13].

Apstraktni razred *GenericServlet* sadrži dva oblika funkcije *init*. Prvi oblik ostvaruje funkciju *init* zadanu sučeljem *Servlet* i prima *ServletConfig* objekt, sprema ga u člansku varijablu i poziva drugi oblik funkcije *init*. Drugi oblik funkcije *init* nema ulaznih parametara. Preporučeno je nadjačavanje drugog oblika funkcije *init*, čime se omogućava dodavanje funkcionalnosti u funkciju *init* bez gubljenja ili ponovnog ostvarivanja osnovne funkcionalnosti poput spremanja objekta *ServletConfig* [13].

Funkcija *log* ostvarena je u dva oblika. Oba oblika služe za zapis poruke u datoteku definiranu udomiteljem *Servlet* objekta. Prvi oblik prima niz znakova sa porukom, te zapisuje u datoteku ime *Servlet* objekta i poruku. Drugi oblik prima niz znakova sa porukom i objekt tipa *Throwable*, te zapisuje u datoteku ime *Servlet* objekta, poruku i trag stoga sadržan u objektu tipa *Throwable* [13].

Apstraktna funkcija *service* služi za obradu zahtjeva kao što je izneseno u sučelju *Servlet*. Razredi koji nadjačavaju razred *GenericServlet* moraju nadjačati funkciju *service* [13].

2.3.3. Apstraktni razred *HttpServlet*

Apstraktni razred *HttpServlet* služi kao temelj za izradu *Servlet* objekta koji komunicira protokolom *HTTP*. Razred *HttpServlet* nasljeđuje razred *GenericServlet*, i neizravno sučelje *Servlet*. Razred koji nadjačava *HttpServlet* mora nadjačati barem jednu funkciju [14].

Komunikacija *HTTP* zahtjevima ostvarena je definicijom funkcija *doDelete*, *doGet*, *doHead*, *doOptions*, *doPost*, *doPut* i *doTrace*. Svaka funkcija zadužena je za pojedini *HTTP* zahtjev. Predviđeno je da se barem neke od ovih funkcija nadjačaju u sklopu ostvarenja funkcionalnosti obrade zahtjeva. Svaka od navedenih funkcija prima kao ulazne parametre objekte tipa *HttpServletRequest* i *HttpServletResponse*. Objekt tipa *HttpServletRequest* sadrži primljeni zahtjev. Objekt tipa *HttpServletResponse* služi za slanje odgovora na zahtjev [14].

Funkciju *doDelete* poslužitelj poziva ukoliko primi *HTTP DELETE* zahtjev. *HTTP DELETE* zahtjev po definiciji služi da korisnik odredi brisanje dokumenta s poslužitelja. *HTTP DELETE* zahtjev traži izvršavanje nesigurnih i neponovljivih operacija, stoga funkcija *doDelete* može imati nenamjerne posljedice (engl. *side effects*) za koje su odgovorni korisnici [14].

Funkcija *doGet* služi za obradu *HTTP GET* zahtjeva. Nadjačavanje ove funkcije automatski podržava i obradu *HTTP HEAD* zahtjeva, koji traži odgovor u obliku zaglavlja, ali bez tijela *HTTP GET* zahtjeva. Popunjavanje zaglavlja odgovora je preporučeno. Funkcija *doGet* treba biti sigurna i idempotentna, i ne bi trebala imati posljedice za koje korisnik odgovara [14].

Funkcija *doHead* služi za obradu *HTTP HEAD* zahtjeva. Odgovor na *HTTP HEAD* zahtjev bi trebao sadržavati zaglavlje odgovora na jednak *HTTP GET* zahtjev. Funkcija *doHead* treba biti sigurna i idempotentna [14].

Funkcija *doOptions* služi za obradu *HTTP OPTIONS* zahtjeva. Odgovor sadrži popis *HTTP* zahtjeva koje poslužitelj može obraditi. Funkcija *doOptions* je ostvarena u potpunosti i nema potrebe za nadjačavanjem [14].

Funkcija *doPost* služi za obradu *HTTP POST* zahtjeva. *HTTP POST* zahtjev služi za slanje neograničene količine podataka na poslužitelj. Funkcija *doPost* ne mora biti sigurna i idempotentna, te može imati nenamjerne posljedice za koje je odgovoran korisnik [14].

Funkcija *doPut* služi za obradu *HTTP PUT* zahtjeva. *HTTP PUT* zahtjev služi za slanje dokumenta na poslužitelj. Funkcija *doPut* ne mora biti sigurna i idempotentna, te može imati nenamjerne posljedice za koje je odgovoran korisnik [14].

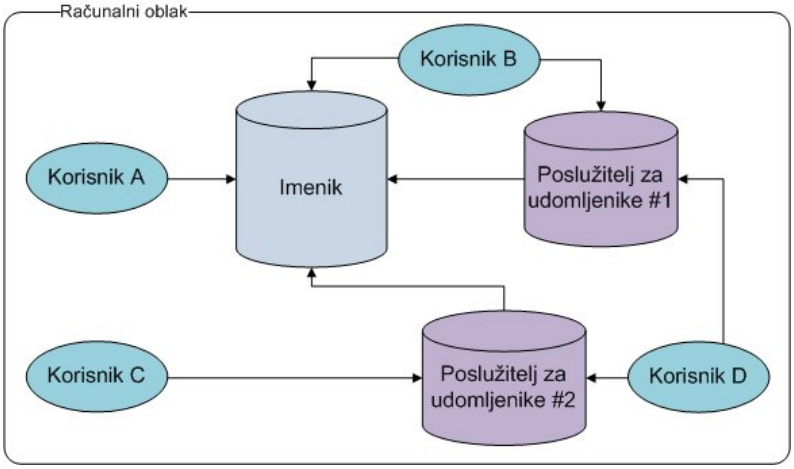
Funkcija *doTrace* služi za obradu *HTTP TRACE* zahtjeva. *HTTP TRACE* zahtjev koristi se za provjeru mrežne povezanosti. Odgovor na zahtjev sadrži zaglavlja prisutna u zahtjevu. Funkcija *doTrace* ostvaruje potpunu traženu funkcionalnost, stoga nema potrebe za nadjačavanjem [14].

Funkcija *getLastModifed* služi za određivanje vremena od zadnje promjene objekta tipa *HttpServletRequest*. Funkcija vraća broj milisekundi između zadnje promjene objekta i ponoći 1. siječnja 1970., GMT. Ukoliko vrijeme nije moguće odrediti, funkcija vraća negativan broj [14].

Apstraktni razred *HttpServlet* ostvaruje dva oblika funkcije *service*. Oblik funkcije *service* koji prima objekte *HttpServletRequest* i *HttpServletResponse* određuje tip *HTTP* zahtjeva i prosljeđuje zahtjev u prikladnu *doX* funkciju. Funkcije *doX* sadrže funkcionalnost za obradu zahtjeva. Oblik funkcije *service* koji prima objekte *ServletRequest* i *ServletResponse* ostvaruje funkciju traženu apstraktnim razredom *GenericServlet*, i prosljeđuje zahtjev prvom obliku funkcije *service*. Oba oblika funkcije *service* sadržavaju potpunu traženu funkcionalnost i nema potrebe za nadjačavanjem [14].

3. Arhitektura *sustava za prikupljanje i objedinjavanje podataka*

Sustav prikupljanja i objedinjavanja podataka u računalnom oblaku je raspodijeljeni sustav sa centraliziranom bazom podataka. Sustav nadograđuje sudionike računalnog oblaka za potrošačko programiranje sa tri moguće uloge: uloga centralizirane baze podataka, uloga poslužitelja udomljenika, te uloga korisnika. Jedan element sustava može imati više različitih uloga. Sudionik s ulogom centralizirane baze podataka naziva se imenik, i u računalnom oblaku postoji točno jedan imenik. Broj poslužitelja za udomljenike i korisnika je proizvoljan i promjenjiv. Komunikacija u sustavu zasniva se na razmjeni poruka prema protokolu *RPPU*, opisanim u poglavlju 4.



Slika 3.1 Arhitektura raspodijeljenog sustava

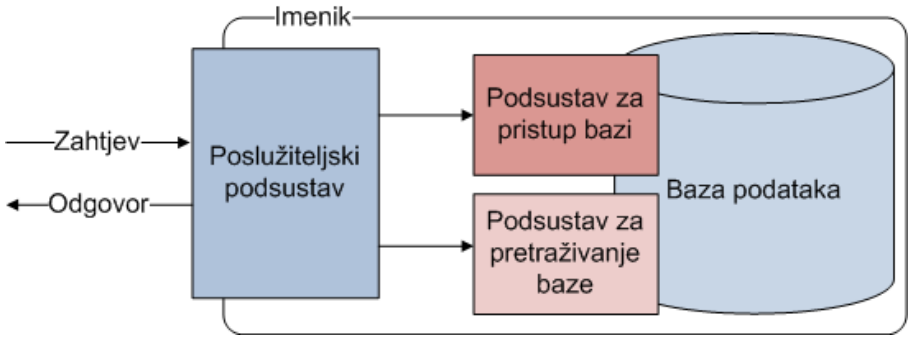
Na slici 3.1 prikazana je arhitektura sustava. Imenik je mrežno dostupna baza opisa udomljenika. Poslužitelji za udomljenike pružaju uslugu mrežnog objavljivanja udomljenika i javljaju promjene u opisima udomljenika imeniku. Korisnici su računalni programi koji mogu upravljati udomljenicima na poslužiteljima za udomljenike ili pretraživati podatke o udomljenicima prijavljenim na imeniku. Korisnici B, C i D stvaraju nove te upravljaju i koriste postojeće udomljenike. Promjene u opisima udomljenika nastale kao rezultati akcija korisnika B, C i D poslužitelji za udomljenike javljaju imeniku. Korisnici A i B šalju zahtjeve za pretraživanje udomljenika koji postoje u sustavu.

U poglavlju 3.1 opisana je arhitekturu imenika. U poglavljima 3.2 i 3.3 opisani su zahtjevi na arhitekture sudionika računalnog oblaka koji preuzimaju

uloge poslužitelja za udomljenike, odnosno korisnika. U poglavlju 3.4 opisani su obrasci rada na razini dijelova sustava.

3.1. Arhitektura imenika

Imenik je središnji dio *sustava za prikupljanje i objedinjavanje podataka u računalnom oblaku*. Zadaća imenika je održavati bazu podataka s opisima udomljenika koji su postavljeni na poslužitelje za udomljenike i obavljati akcije zatražene porukama. Akcija izvršava sve promjene zatražene zahtjevom ili niti jedna promjena, tj. akcije se izvršavaju po načelu atomarnosti.



Slika 3.2 Arhitektura imenika

Na slici 3.2 prikazana je građa imenika. Imenik se sastoji od poslužiteljskog podsustava, podsustava za pristup bazi podataka, podsustava za pretraživanje i baze podataka. Podsustavi za pristup bazi podataka i pretraživanje mogu biti ostvareni kao dva odvojena podsustava, ili kao jedan modul sa funkcionalnosti oba podsustava.

Poslužiteljski podsustav je mrežno sučelje imenika. Prima poruke prenesene mrežom, obavlja potrebna dekodiranja, poziva podsustave za pristup i pretraživanje baze podataka te kodira i odašilje odgovore.

Podsustav za pristup bazi podataka služi za izvršavanje zahtjeva za dodavanje, promjenu i brisanje podataka spremljenih u bazi podataka. Baza podataka čuva i održava podatke o poslužiteljima za udomljenike i udomljenicima.

Podsustav za pretraživanje je sučelje za pretraživanje baze podataka prema zadanim uvjetima. Zahtjev za pretraživanjem definira uvjete pretrage.

Svaki udomljenik se ocjenjuje prema razini kojom udovoljava zadanim uvjetima. Kao odgovor na zahtjev na pretragu se šalju udomljenici koji zadovoljavaju barem jedan od uvjeta pretrage. Podsustav za pretraživanje može biti ostvaren kao dio podsustava za pristup bazi podataka ili kao odvojeni podsustav koji podatke dohvaća izravno iz baze.

3.2. Zahtjevi na arhitekturu poslužitelja za udomljenike

Ulogu poslužitelja za udomljenike preuzimaju sudionici računalnog oblaka koji posjeduju mogućnost mrežnog objavljivanja udomljenika. Sudionik koji ostvaruje ulogu poslužitelja za udomljenike preuzima odgovornost održavanja podataka spremljenih u imeniku. Poslužitelj za udomljenike koristi protokol *RPPU* za komunikaciju s imenikom.

Poslužitelj za udomljenike prijavljuje se u imenik i dobiva identifikacijski broj. Podaci spremljeni u imenik čine osnovne podatke o poslužitelju (na primjer, mrežna adresa poslužitelja), te podatke koji opisuju udomljenike postavljene na poslužitelj. Poslužitelj za udomljenike obavještava imenik o stvaranju novih i promjeni podataka postojećih udomljenika. Također može poslati zahtjev za brisanjem podataka o udomljeniku ako udomljenik prestane biti mrežno dostupan.

3.3. Zahtjevi na arhitekturu korisnika

Korisnik u *sustavu za prikupljanje i objedinjavanje podataka u računalnom oblaku* može komunicirati sa imenikom i sa poslužiteljima za udomljenike. Sve akcije u *sustavu za prikupljanje i objedinjavanje podataka* izravno ili neizravno pokreće korisnik. Izravno, korisnik šalje zahtjev za pretraživanjem imenika ili upravlja udomljenicima na poslužitelju za udomljenike. Neizravno, upravljanje udomljenicima rezultira promjenom podataka o udomljenicima i uzrokuje komunikaciju između poslužitelja za udomljenike i imenika.

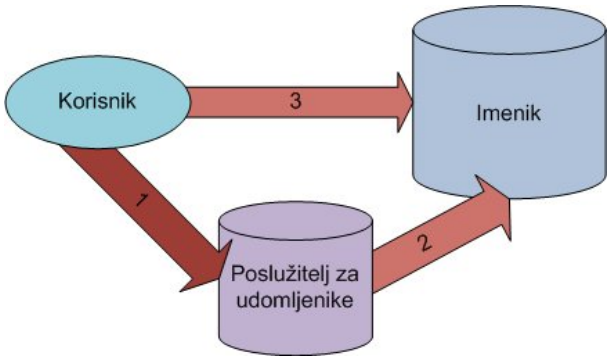
Komunikacijom sa poslužiteljem za udomljenike korisnik može upravljati udomljenicima objavljenim na tom poslužitelju. Upravljanje udomljenicima

obuhvaća stvaranje novih udomljenika te korištenje, mijenjanje i brisanje postojećih udomljenika.

Komunikaciju između korisnika i imenika čine zahtjevi za pretraživanjem podataka spremljenih u imenik. Korisnik postavlja uvjete pretraživanja, a imenik odgovara sa podacima o udomljenicima koji zadovoljavaju uvjete.

3.4. Rad *sustava za prikupljanje i objedinjavanje podataka*

Rad *sustava za prikupljanje i objedinjavanje podataka* pokreću korisnici akcijama upravljanja udomljenicima na poslužiteljima za udomljenike i zahtjevima za pretraživanje imenika. Komunikacija unutar sustava odvija se prema modelu korisnik-poslužitelj (engl. *client-server*) [3].



Slika 3.3 Smjerovi komunikacije u sustavu

Sustav u svom radu jedinstveno identificira poslužitelja za udomljenike prema njegovom identifikacijskom broju. Imenik dodjeljuje identifikacijski broj poslužitelja pri prvoj prijavi poslužitelja. Udomljenici se jedinstveno identificiraju kombinacijom identifikacijskog broja udomljenika i identifikacijskog broja poslužitelja za udomljenike na kojemu su objavljeni. Poslužitelj za udomljenike dodjeljuje identifikacijski broj udomljeniku pri nastanku udomljenika.

Na slici 3.3 prikazani su mogući parovi komunikacije strelicama 1, 2 i 3. U sklopu komunikacije, poruka sa zahtjevom šalje se u smjeru strelice, a potom se poruka odgovora šalje u suprotnom smjeru. Imenik nikad ne šalje zahtjeve. Komunikacija između korisnika i poslužitelja za udomljenike, prikazana strelicom 2, nije definirana u sklopu *sustava za prikupljanje i objedinjavanje podataka*.

Pretraživanja podataka spremljenih u imenik se odvija na sljedeći način. Korisnik šalje zahtjev za pretraživanjem imenika (strelica 3 na slici 3.3). Imenik pretražuje bazu podataka i priprema listu udomljenika koji odgovaraju uvjetima. Imenik šalje odgovor koji sadrži podatke o pronađenim udomljenicima.

Komunikacija između poslužitelja za udomljenike i imenika, prikazana strelicom 2 na slici 3.3, služi za održavanje baze podataka u imeniku. Poslužitelj za udomljenike šalje zahtjev za upravljanjem podacima o udomljenicima spremljenim u imenik. Imenik prima zahtjev i provodi traženu akciju, te odgovora sa prikladnom porukom.

4. Protokol *RPPU*

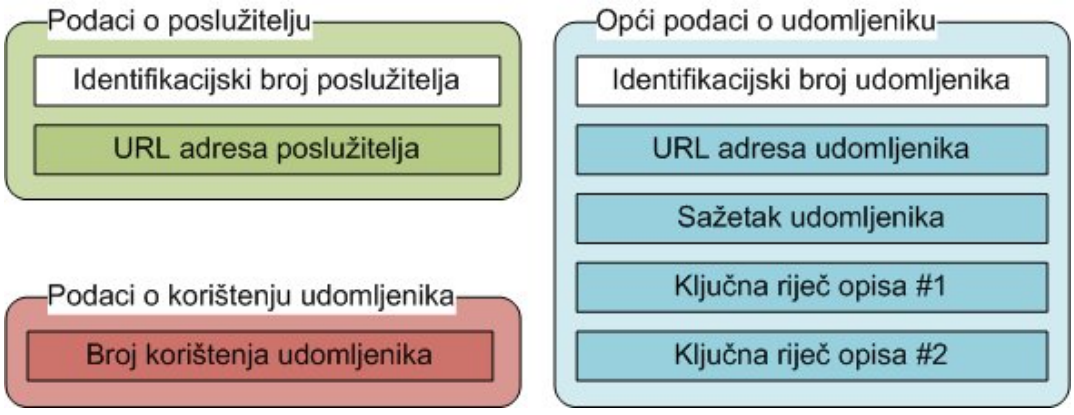
Protokol *raspodijeljenog prijavljivanja i pretraživanja udomljenika* (*RPPU*, engl. *Distributed report and discovery of widgets*) je protokol posredničkog sloja koji definira poruke koje se koriste u *sustavu za prikupljanje i objedinjavanje podataka*.

Protokol *RPPU* je definiran i ostvaren za potrebe programskog ostvarenja *sustava za prikupljanje i objedinjavanje podataka*. Poruke se zapisuju jezikom *XML*, i prenose se *HTTP* protokolom. Izgled poruka definiran je u *XML Schema* jeziku.

U poglavlju 4.1 opisani su zahtjevi koje protokol rješava. Poglavlja 4.2 do 4.9 sadrže opise poruka definiranih protokolom *RPPU*. Primjeri u poglavljima 4.2 do 4.9 zasnovani su na podacima prikazanim na slici 4.1.

4.1. Zahtjevi na protokol

Komunikacija unutar *sustava za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje* služi za prijenos zahtjeva između elemenata sustava. Zahtjevi definiraju akciju koja treba biti izvršena nad bazom podataka u imeniku. Poruka mora definirati zahtjev i prenijeti podatke potrebne za pravilno izvršavanje zahtjeva.



Slika 4.1 Primjer podataka koji se koriste u sustavu

Poruke prenose podatke koji opisuju udomljenike i poslužitelje za udomljenike. Podaci koji se koriste unutar sustava dijele se u tri skupine: *Podaci o poslužitelju*, *Opći podaci o udomljeniku* i *Podaci o korištenju udomljenika*.

Svaka poruka ima definirane specifične podatke ili skupinu podataka koju prenosi. Na slici 4.1 prikazan je primjer podataka za svaku od navedenih skupina.

Skupina *Podaci o poslužitelju* sastoji se od podataka koji opisuju poslužitelja za udomljenike i ne ovise o udomljenicima prijavljenim na poslužitelja. Identifikacijski broj i mrežna adresa poslužitelja primjer su podataka iz skupine *Podaci o poslužitelju*, kao što je prikazano na slici 4.1.

Skupina *Opći podaci o udomljeniku* sadržava podatke koji opisuju i definiraju udomljenika. Na slici 4.1 prikazan je primjer podataka koji spadaju u skupinu *Opći podaci o udomljeniku*. Podatak o ključnoj riječi opisa može se ponoviti više puta, dok podaci o identifikacijskom broju, mrežnoj adresi i sažetku programskog kôda su jedinstveni za svaki udomljenik.

Skupina *Podaci o korištenju udomljenika* sastoji se od podataka koji opisuju korištenje udomljenika. Primjer prikazan na slici 4.1 sadrži podatak o broju korištenja udomljenika. Pretpostavka je da će podaci iz skupine *Podaci o korištenju udomljenika* biti mijenjani mnogo češće od podataka iz druge dvije skupine.

Identifikacijski broj poslužitelja na kojem je udomljenik objavljen i identifikacijski broj udomljenika čine ključ za jedinstvenu identifikaciju udomljenika. Identifikacijski broj poslužitelja pripada u skupinu *Podaci o poslužitelju* i služi za jednostavnu identifikaciju poslužitelja. Identifikacijski broj udomljenika, koji pripada u skupinu *Opći podaci o udomljeniku*, dodjeljuje poslužitelj pri stvaranju udomljenika.

U sklopu *sustava za prikupljanje i objedinjavanje podataka* prenose se zahtjevi za prijavom poslužitelja u imenik, zapisivanjem podataka o novom udomljeniku, promjeni i brisanju podataka o udomljeniku te pretraživanju podataka o udomljenicima prijavljenim u imenik. Radi prijenosa navedenih zahtjeva u okviru protokola *RPPU* definirane su poruke *Prijavi*, *Stvori*, *Promijeni*, *Ažuriraj, Obriši*, *Pretraži* i *Rezultati pretrage*. Poruka *Prijavi* služi za upravljanje podacima o poslužitelju. Poruke *Stvori*, *Promijeni*, *Ažuriraj* i *Obriši* služe za upravljanje podacima o udomljeniku. Poruke *Pretraži* i *Rezultati pretraživanja*

služe za pretragu podataka o udomljenicima. Poruka *Greška* služi za dojavu o neuspjehu provedbe akcije.

Odgovor ovisi o uspjehu provedbe akcije. Ako je akcija bila uspješno provedena, odgovor može nositi podatke koji su rezultat akcije. U slučaju da primatelj nije uspio provesti akciju, odgovor čini poruka *Greška* sa opisom greške. Poruke *Prijavi*, *Stvori*, *Promijeni*, *Ažuriraj*, *Obriši* i *Pretraži* su poruke zahtjeva, a poruke *Rezultati pretraživanja* i *Greška* su poruke odgovora na zahtjev.

4.2. Poruka *Prijavi*

Poruka *Prijavi* služi za prijavu i promjenu podataka koje čuva imenik o poslužitelju za udomljenike. Poruka prenosi djelomičnu ili cijelu skupinu *Podaci o poslužitelju*. Pošiljatelj poruke je poslužitelj za udomljenike, a primatelj imenik. U sklopu programskog ostvarenja poruka *Prijavi* je definirana kao *XML* element *mesServerStartup*, a odgovori su definirani kao *XML* elementi *mesServerInitSuccess* i *mesServerChangeSuccess*.

A
```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesServerStartup
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
  <tns:serverURL>www.server1.hr</tns:serverURL>
</tns:mesServerStartup>
```

B
```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesServerStartup
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
  <tns:serverID>1</tns:serverID>
  <tns:serverURL>www.server1.com</tns:serverURL>
</tns:mesServerStartup>
```

Slika 4.2 Primjeri dva oblika poruke *Prijavi*

Korištenje poruke *Prijavi* za prvu prijavu poslužitelja za udomljenike zahtijeva slanje svih podataka iz skupine *Podaci o poslužitelju* osim identifikacijskog broja poslužitelja. Primjer takve poruke je dan na slici 4.2 s porukom A, koja prijavljuje novi poslužitelj za udomljenike u imenik. Poruka prenosi samo oznaku *serverURL*, čija vrijednost je mrežna adresa novog poslužitelja. Imenik dodjeljuje identifikacijski broj i šalje ga poslužitelju za udomljenike u sklopu odgovora na poruku. Primjer odgovora na poruku A sa

slike 4.2 je dan pod A na slici 4.3. U primjeru je imenik dodijelio identifikacijski broj *„1“* poslužitelju s mrežnom adresom *„www.server1.hr“*. Poslužitelj za udomljenike mora čuvati dodijeljeni identifikacijski broj radi prijavljivanja i promjene svojih udomljenika u imenik.

A
```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesServerInitSuccess
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
1</tns:mesServerInitSuccess>
```

B
```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesServerChangeSuccess
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
1</tns:mesServerChangeSuccess>
```

Slika 4.3 Primjeri odgovora na poruku Prijavi

Pri korištenju poruke *Prijavi* za promjenu podataka o poslužitelju za udomljenike šalje se identifikacijski broj i podaci koji se mijenjaju. Poruka B na slici 4.2 pokazuje primjer korištenja za promjenu URL adrese poslužitelja udomljenike. Poruka javlja imeniku da poslužitelj sa identifikacijskim brojem *„1“* se sada nalazi na mrežnoj adresi *„www.server1.com“*. Imenik će javiti grešku ako poslužitelj s identifikacijski brojem *„1“* nije prijavljen u imenik. Ako se nije dogodila greška, imenik odgovara s porukom koja sadrži identifikacijski broj poslužitelja. Primjer poruke odgovora je označen sa B na slici 4.3.

4.3. Poruka *Stvori*

Poruka *Stvori* je zahtjev za dodavanje podataka o novom udomljeniku u imenik. Spada u grupu poruka za upravljanje podacima o udomljeniku. Poruka prenosi skupinu *Opći podaci o udomljeniku* i identifikacijski broj poslužitelja. Identifikacijski broj poslužitelja i identifikacijski broj udomljenika zajedno jedinstveno određuju udomljenika u imeniku.

Primjer poruke *Stvori* dan je na slici 4.4. Poruka sadrži oznaku *serverID* sa identifikacijskim brojem poslužitelja koji mrežno objavljuje udomljenika, u ovom slučaju *„21“*. Oznaka *generalData* sadrži skupinu oznaka koje pripadaju u grupu *Opći podaci o udomljeniku*. Vrijednosti oznaka *serverID* i *gadgetID* zajedno određuju udomljenika, u ovom slučaju par *(21, 42)*. Oznaka *gadgetURL*

predstavlja mrežnu adresu udomljenika, a oznaka *hash* sažetak programskog kôda udomljenika. Oznaka *keyword* predstavlja ključnu riječ ili izraz koji opisuje udomljenika. Svaki udomljenik je opisan jednom ili više ključnih riječi/izraza. U primjeru na slici 4.4 udomljenik je opisan sa ključnim riječima *„Primjer“* i *„protokol RPPU“*

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesCreateGadget
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
  <tns:serverID>21</tns:serverID>
  <tns:generalData>
    <tns:gadgetID>42</tns:gadgetID>
    <tns:gadgetURL>www.gadget-primjer.gov</tns:gadgetURL>
    <tns:hash>A3D2FFFB449</tns:hash>
    <tns:keyword>Primjer</tns:keyword>
    <tns:keyword>protokol RPPU</tns:keyword>
  </tns:generalData>
</tns:mesCreateGadget>
```

Slika 4.4 Primjer poruke *Stvori*

Greška nastaje ako u imenik nije prijavljen poslužitelj s identifikacijskim brojem *„21“*, te ako u imeniku je već postoje podaci o udomljeniku definiranom parom *(21, 42)*, odnosno da za definirani poslužitelj već postoje podaci o udomljeniku s identifikacijskim brojem *„42“*. U oba slučaja odgovor je poruka *Greška* sa opisom greške.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesCreateSuccess
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
42</tns:mesCreateSuccess>
```

Slika 4.5 Primjer odgovora na poruku *Stvori*

Ukoliko ne dođe do greške, imenik zapisuje podatke o udomljeniku i odgovara porukom koja sadrži identifikacijski broj udomljenika koji je upravo dodan. Na slici 4.5 prikazan je primjer poruke odgovora nakon uspješnog dodavanja udomljenika s identifikacijskim brojem *„42“*.

4.4. Poruka *Promijeni*

Poruka *Promijeni* služi za promjenu podataka iz skupine *Opći podaci o udomljeniku*. Dio je grupe poruka za upravljanje podacima o udomljeniku. Poruka prenosi identifikacijski broj poslužitelja i identifikacijski broj udomljenika

s kojima se specificira udomljenik, te nove vrijednosti podataka iz skupine *Opći podaci o udomljeniku*.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesChangeGadget
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
  <tns:serverID>21</tns:serverID>
  <tns:changeGadget>
    <tns:gadgetID>42</tns:gadgetID>
    <tns:gadgetURL>www.server21.com/gadget-primjer</tns:gadgetURL>
    <tns:keyword>raspodijeljeni sustav</tns:keyword>
    <tns:keyword delete="false">sustav za prikupljanje i objedinjavanje podataka</tns:keyword>
    <tns:keyword delete="true">Primjer</tns:keyword>
  </tns:changeGadget>
</tns:mesChangeGadget>
```

Slika 4.6 Primjer poruke *Promjeni*

Na slici 4.6 prikazan je primjer poruke *Promjeni*. Poruka sadrži identifikacijski broj poslužitelja za udomljenike pod oznakom *serverID* i podatke iz skupine *Opće podaci o udomljeniku* pod oznakom *changeGadget*. Oznaka *gadgetID* sadrži identifikacijski broj udomljenika. Vrijednosti spremljene pod oznakama *serverID* i *gadgetID* definiraju udomljenika čiji podaci se mijenjaju. Podatak pod oznakom *gadgetURL* predstavlja novu vrijednost podatka o mrežnoj adresi udomljenika. Primjer ne sadrži oznaku *hash*, iako ona spada u skupinu *Opće podaci o udomljeniku*, što znači da se vrijednost oznake *hash* zapisane u imeniku ne mijenja. Oznaka *keyword* je navedena u primjeru u tri oblika. Prva dva oblika oznake su istovjetna, i označavaju da se vrijednost oznake dodaje u opis udomljenika u imeniku. Treći oblik oznake *keyword* ima vrijednost atributa *delete* postavljenu na *true*, što označava da se treba obrisati vrijednost oznake *keyword* iz opisa udomljenika u imeniku. Ukoliko opis udomljenika u imeniku ne sadrži ključnu riječ navedenu u vrijednosti oznake *keyword*, nema promjene u opisu udomljeniku. Prema poruci sa slike 4.6, udomljeniku identificiranom parom *(21, 42)* bi se promijenila mrežna adresa u *„www.server21.com/gadget-primjer“*, te u opis dodali ključni izrazi *„raspodijeljeni sustav“* i *„sustav za prikupljanje i objedinjavanje podataka“*, a izbrisao ključni izraz *„Primjer“*.

Greška nastaje ako u imenik nije prijavljen poslužitelj s poslanim identifikacijskim brojem, ako za definirani poslužitelj nije prijavljen udomljenik s poslanim identifikacijskim brojem ili ako nakon provedbe uputa navedenih u poruci opis udomljenika ne bi sadržavao barem jednu ključnu riječ.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesChangeSuccess
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
42</tns:mesChangeSuccess>
```

Slika 4.7 Primjer odgovora na poruku *Promjeni*

Na slici 4.7 prikazan je primjer odgovora na poruku. Odgovor sadrži identifikacijski broj udomljenika koji je bio mijenjan.

4.5. Poruka *Ažuriraj*

Poruka *Ažuriraj* služi za promjenu podataka iz skupine *Podaci o korištenju udomljenika*. Pripada grupi poruka za upravljanje podacima o udomljeniku. Poruka prenosi identifikacijski broj poslužitelja, identifikacijski broj udomljenika i podatke iz skupine *Podaci o korištenju udomljeniku*.

Udomljenik kojemu se mijenjaju podaci o korištenju identificira se koristeći ključ koji se sastoji od identifikacijskog broja poslužitelja i identifikacijskog broja udomljenika. Podaci iz skupine *Podaci o korištenju udomljenika* se postavljaju na primljene vrijednosti, ili ne mijenjaju ako nisu bili sadržani u poruci.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesUpdateGadget
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
  <tns:serverID>21</tns:serverID>
  <tns:gadgetID>42</tns:gadgetID>
  <tns:usageData>
    <tns:numTimesUsed>13896</tns:numTimesUsed>
  </tns:usageData>
</tns:mesUpdateGadget>
```

Slika 4.8 Primjer poruke *Ažuriraj*

Na slici 4.8 prikazan je primjer poruke *Ažuriraj*. Oznaka *usageData* sadržava podatke koji spadaju u skupinu mijenjaju *Podaci o korištenju udomljenika*, u ovom slučaju podatak označen sa *numTimesUsed*. Poruka u primjeru traži postavljanje broja korištenja udomljenika s identifikacijskim brojem *„42“* objavljenom na poslužitelju s identifikacijskim brojem *„21“* na vrijednost *„13896“*.

Greška nastaje ako u imeniku nije prijavljen poslužitelj s primljenim identifikacijskim brojem, ili ako udomljenik s primljenim identifikacijskim brojem nije prijavljen za definirani poslužitelj.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesUpdateSuccess
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
42</tns:mesUpdateSuccess>
```

Slika 4.9 Primjer odgovora na poruku *Ažuriraj*

Odgovor na poruku *Ažuriraj* je prikazan na slici 4.9. Odgovor sadrži identifikacijski broj udomljenika čiji podaci su bili mijenjani.

4.6. Poruka *Obriši*

Poruka *Obriši* služi za brisanje podataka o udomljeniku sa imenika. Spada u grupu poruka za upravljanje podacima o udomljeniku. Poruka sadrži dva podatka: identifikacijski broj poslužitelja i identifikacijski broj udomljenika, koji čine ključ za određivanje udomljenika. Podaci o tom udomljeniku se brišu iz imenika.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesDeleteGadget
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
  <tns:serverID>21</tns:serverID>
  <tns:gadgetID>42</tns:gadgetID>
</tns:mesDeleteGadget>
```

Slika 4.10 Primjer poruke *Obriši*

Na slici 4.10 prikazan je primjer poruke *Obriši* kojom se traži brisanje podataka o udomljeniku s identifikacijskim brojem *„42“* objavljenog na poslužitelju za udomljenike identifikacijskog broja *„21“*. U slučaju uspješnog brisanja, odgovor čini poruka s identifikacijskim brojem udomljenika. Primjer odgovora je dan na slici 4.11.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesDeleteSuccess
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
42</tns:mesDeleteSuccess>
```

Slika 4.11 Primjer odgovora na poruku *Obriši*

Greška nastaje ako u imeniku nije prijavljen poslužitelj s primljenim identifikacijskim brojem, ili ako udomljenik s primljenim identifikacijskim brojem nije prijavljen za definirani poslužitelj.

4.7. Poruka *Pretraži*

Poruka *Pretraži* je zahtjev za pretragu podataka o udomljenicima. Odgovor na poruku u slučaju uspješno obavljenog pretraživanja čini poruka *Rezultati pretrage* opisana u poglavlju 4.8.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesSearch
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
  <tns:numSearchResults>100</tns:numSearchResults>
  <tns:gadgetURL>.com</tns:gadgetURL>
  <tns:keyword>protokol</tns:keyword>
  <tns:keyword>raspodijeljen</tns:keyword>
</tns:mesSearch>
```

Slika 4.12 Primjer poruke *Pretraži*

Na slici 4.12 prikazan je primjer poruke *Pretraži*. U primjeru je postavljen maksimalni broj udomljenika poslanih kao odgovor i dani uvjeti rangiranja udomljenika. Tri uvjeta rangiranja su postavljena. Uvjet pod oznakom *gadgetURL* prihvaća udomljenike čija mrežna adresa sadrži niz znakova *„.com“*. Uvjeti pod oznakom *keyword* prihvaćaju udomljenike sa ključnim riječima ili izrazima koje sadrže nizove znakova *„protokol“* ili *„raspodijeljen“*. Udomljenik koji zadovoljava više uvjeta se postavlja na viši rang na ljestvici. Podatak spremljen pod oznakom *numSearchResults* određuje podaci od koliko najbolje rangiranih udomljenika trebaju biti poslani kao odgovor.

4.8. Poruka *Rezultati pretraživanja*

Poruka *Rezultati pretraživanja* služi kao odgovor na poruku *Pretraži*. Poruka prenosi podatke o udomljenicima koji ispunjavaju uvjete zadane u poruci *Pretraži*. Za svaki udomljenik čiji podaci ispunjavaju uvjete pretrage se prenosi identifikacijski broj poslužitelja na kojemu je prijavljen, te svi podaci iz skupina *Opći podaci o udomljeniku* i *Podaci o korištenju udomljenika*. Redoslijed udomljenika u poruci nije definiran protokolom.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesSearchReply
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
  <tns:searchReplyGadget>
    <tns:serverID>143</tns:serverID>
    <tns:generalData>
      <tns:gadgetID>12</tns:gadgetID>
      <tns:gadgetURL>www.policija.hr/transferUdomljenik</tns:gadgetURL>
      <tns:hash>FFF376CD</tns:hash>
      <tns:keyword>policijski protokoli</tns:keyword>
      <tns:keyword>preraspodijeljen</tns:keyword>
    </tns:generalData>
    <tns:usageData>
      <tns:numTimesUsed>964</tns:numTimesUsed>
    </tns:usageData>
  </tns:searchReplyGadget>
  <tns:searchReplyGadget>
    <tns:serverID>21</tns:serverID>
    <tns:generalData>
      <tns:gadgetID>42</tns:gadgetID>
      <tns:gadgetURL>www.server21.com/gadget-primjer</tns:gadgetURL>
      <tns:hash>A3D2FFFB449</tns:hash>
      <tns:keyword>protokol RPPU</tns:keyword>
      <tns:keyword>raspodijeljeni sustav</tns:keyword>
      <tns:keyword>sustav za prikupljanje i objedinjavanje podataka</tns:keyword>
    </tns:generalData>
    <tns:usageData>
      <tns:numTimesUsed>13896</tns:numTimesUsed>
    </tns:usageData>
  </tns:searchReplyGadget>
</tns:mesSearchReply>
```

Slika 4.13 Primjer poruke *Rezultati pretraživanja*

Maksimalni broj udomljenika čiji podaci se šalju u odgovoru zadaje se u poruci *Pretraži*. Ako poruka ne sadrži maksimalni traženi broj udomljenika, imenik određuje broj udomljenika koje šalje kao odgovor. Broj udomljenika koji se prenosi može biti manji od definiranog.

Na slici 4.13 prikazan je primjer poruke *Rezultati pretraživanja*, kao primjer odgovora na poruku *Pretraži* prikazanu na slici 4.12. Odgovor čine dva udomljenika, svaki smješten unutar oznake *searchReplyGadget*. Za svaki udomljenik se prenosi identifikacijski broj poslužitelja na kojemu je objavljen pod oznakom *serverID*, podaci iz skupine *Opći podaci o udomljeniku* smješteni pod oznakom *generalData* i podaci iz skupine *Podaci o korištenju udomljenika* pod oznakom *usageData*. U primjeru drugi udomljenik zadovoljava sva tri uvjeta iznesena na poruci prikazanoj na slici 4.12, dok prvi udomljenik zadovoljava uvjet na ključnim riječima/izrazima, ali ne i uvjet na mrežnu adresu.

4.9. Poruka *Greška*

Poruka *Greška* koristi se kao odgovor na poruku ako nije moguće provesti akciju zahtijevanu porukama *Prijava*, *Stvori*, *Promijeni*, *Ažuriraj*, *Obriši* i *Pretraži*. Poruka u sebi sadrži opis greške koja se dogodila. prikazuje primjer poruke *Greška* u slučaju pokušaja promjene podataka poslužitelja čiji identifikacijski broj ne postoji.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:error
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
Server with ID 4 does not exist!</tns:error>
```

Slika 4.14 Primjer poruke *Greška*

5. Programsko ostvarenje imenika

Imenik je ostvaren kao *Java Servlet* u programskom jeziku *Java*, korištenjem razvojnog okruženja *Eclipse Java EE* i testiran pomoću *Apache Tomcat* poslužitelja [9, 12]. Komunikacija s imenikom obavlja se pomoću poruka zapisanih jezikom *XML* prema protokolu *RPPU* opisanom u poglavlju 4.

Programsko ostvarenje imenika rađeno je modularno kako bi se omogućilo lakše održavanje i proširivanje. U sklopu modularne izrade, apstraktni razred *Repository* opisuje sučelje baze podataka, a apstraktni razred *Search* opisuje sučelje podsustava za pretraživanje. Razred *RepositoryConstants* sadrži konstante koje se koriste u drugim razredima radi lakšeg usklađivanja i promjene podataka. Konstante sadržane u razredu *RepositoryConstants* su nazivi oznaka, početne vrijednosti podataka iz skupine *Podaci o korištenju udomljenika* i kosturi poruka odgovora, koje se popuni vrijednostima tijekom obrade zahtjeva.

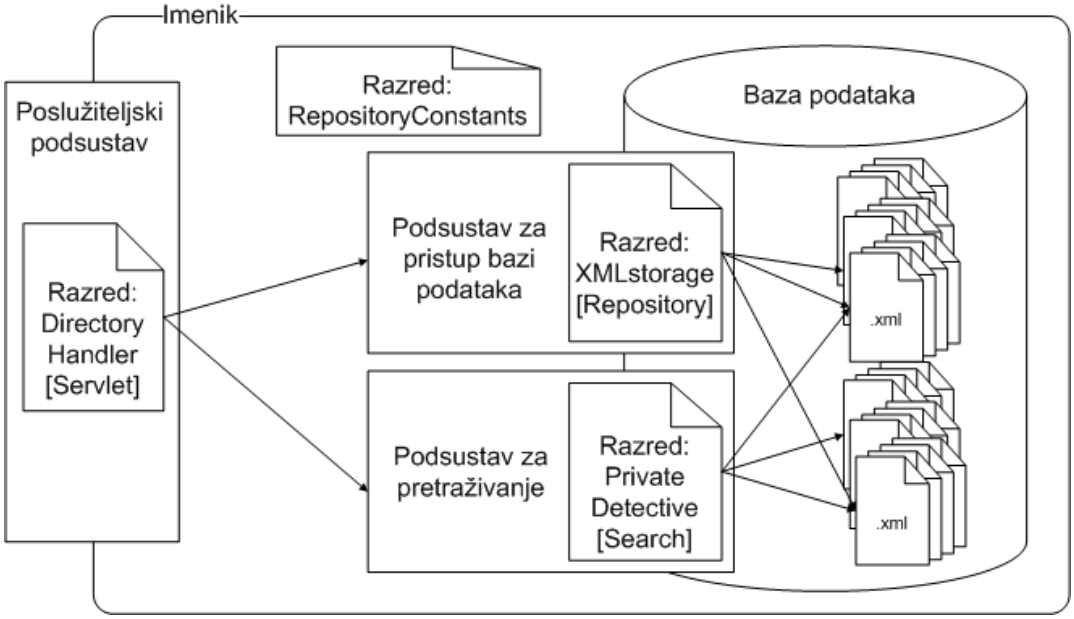
Sučelje podsustava za pristup bazi podataka opisano je u apstraktnom razredu *Repository* i sastoji se od funkcija *start*, *create*, *change*, *update* i *delete*. Svaka funkcija prima objekt tipa *org.w3c.dom.Document* koji sadrži poruku zahtjeva. Funkcije vraćaju znakovni niz koji predstavlja poruku odgovora prema protokolu *RPPU*.

Sučelje podsustava za pretraživanje, opisano u razredu *Search*, sastoji se od funkcije *search* koja prima poruku u obliku *org.w3c.dom.Document* objekta i vraća znakovni niz koji sadrži poruku *Rezultati pretraživanja* definiranu protokolom *RPPU*. Podsustav za pretraživanje pretražuje bazu podataka prema uputama iz poruke.

Programsko ostvarenje imenika pokreće se postavljanjem na poslužitelj koji podržava *Java Servlet* tehnologiju. Radi pravilnog rada poslužiteljskog podsustava, poslužitelj na kojemu je postavljen imenik mora proslijediti imeniku *ServletConfig* objekt.

Na slici 5.1 prikazana je građa imenika na razini programskog ostvarenja. Programsko ostvarenje imenika zasniva se na poslužiteljskom podsustavu ostvarenom razredom *DirectoryHandler*. Bazu podataka čine datoteke s podacima zapisanim u jeziku *XML* i podsustav za pristup bazi. Podsustav za

pristup bazi podataka ostvaren je razredom *XMLstorage*. Podsustav za pretraživanje ostvaren je razredom *PrivateDetective*.



Slika 5.1 Programsko ostvarenje imenika

Programsko ostvarenje poslužiteljskog podsustava opisano je u poglavlju 5.1. Oblik *XML* datoteka opisan je u poglavlju 5.2. Programsko ostvarenje podsustava za pristup bazi podataka opisano je u poglavlju 5.3. Poglavlje 5.4 opisuje programsko ostvarenje podsustava za pretraživanje.

5.1. Ostvarenje poslužiteljskog podsustava

Poslužiteljski podsustav je programski ostvaren koristeći *Java Servlet* tehnologiju u obliku razreda *DirectoryHandler*. Razred nadjačava generički razred *HttpServlet* koja pruža osnovnu funkcionalnost za primanje *HTTP* zahtjeva u sklopu programskog jezika *Java* [14].

Poslužiteljski podsustav je zadužen za primanje *HTTP* zahtjeva. Na osnovi primljenog zahtjeva poslužiteljski podsustav poziva bazu podataka ili podsustav za pretraživanje i predaje im podatke prenesene porukom. Nakon što baza podataka i podsustav za pretraživanje obave akciju, predaju odgovor poslužiteljskom podsustavu koji ga šalje pošiljatelju zahtjeva. Radi ostvarenja ovih mogućnosti razred *DirectoryHandler* nadjačava tri funkcije razreda *HttpServlet*: *init*, *doPost* i *doGet*.

Programsko ostvarenje poslužiteljskog sustava koristi članske varijable za zapis lokacije direktorija sa bazom podataka i čuvanje objekata koji predstavljaju podsustave za pristup i pretraživanje baze podataka. Tipovi varijabli podsustava i pozivi funkcionalnosti definirani su sučeljima *Repository* i *Search*.

U sklopu sustava se poruke prenose pomoću *HTTP GET* i *HTTP POST* zahtjeva. Poruke *Prijavi*, *Stvori*, *Promijeni*, *Ažuriraj* i *Obriši* se prenose koristeći *HTTP POST*, a poruka *Pretraži* se prenosi koristeći *HTTP GET* zahtjev. Poslužiteljski podsustav prihvaća poruke samo ako su poslane pravim tipom *HTTP* zahtjeva.

5.1.1. Funkcija *init*

Funkcija *init* poziva se pri pokretanju poslužitelja na kojeg je imenik postavljen i služi za postavljanje početnih vrijednosti članskih varijabli. U sklopu funkcije *init* određuje se lokacija direktorija sa datotekama baze podataka, te stvaraju objekti koji predstavljaju bazu podataka i podsustav za pretraživanje.

Lokacija direktorija sa datotekama baze podataka određuje se pomoću parametra koji se prenosi u objektu *ServletConfig*. *ServletConfig* objekt ovisi o programskog ostvarenju poslužitelja na koji je postavljen imenik. Lokacija direktorija treba biti zapisana kao parametar s imenom *directory* i sadržavati apsolutni put do direktorija. Put mora završavati sa znakom „\“.

Pri postavljanju vrijednosti članskih varijabli, funkcija *init* sprema objekt tipa *XMLstorage* u varijablu koja predstavlja podsustav za pristup bazi podataka i objekt tipa *PrivateDetective* u varijablu koja predstavlja podsustav za pretraživanje.

5.1.2. Funkcija *doGet*

Funkcija *doGet* poziva se pri dolasku *HTTP GET* zahtjeva u imenik. Poruka se prenosi kao upit dio *URI*-a imenika. Funkcija prima poruku i pretvara je u objekt tipa *org.w3c.dom.Document*. Ako se radi o poruci *Pretraži*, poziva se funkcija *search* definirana u apstraktnom razredu *Search* i ostvarena u razredu *PrivateDetective*, kojoj se predaje objekt s porukom. Funkcija *search* po

završetku obrade vraća znakovni niz, koji funkcija *doGet* šalje kao odgovor na zahtjev.

Ako poruka nije bila tipa *Pretraži*, funkcija *doGet* ne obrađuje zahtjev nego odgovara s porukom *Greška*.

5.1.3. Funkcija *doPost*

Funkcija *doPost* poziva se pri dolasku *HTTP POST* zahtjeva u imenik. Poruka dolazi kao znakovni niz u tijelu zahtjeva, te potom pretvara u objekt tipa *org.w3c.dom.Document*. Funkcija *doPost* potom određuje o kojoj poruci je riječ, te poziva prikladnu funkciju definiranu apstraktnim razredom *Repository* i ostvarenoj u razredu *XMLstorage*. Odabranoj funkciji se predaje objekt koji sadrži poruku te potom čeka njen rezultat. Rezultat funkcije za obradu poruke je znakovni niz koji sadrži poruku odgovora. Poruku odgovora potom funkcija *doPost* šalje pošiljatelju zahtjeva.

Ako poruka nije bila niti jednog od navedenih tipova, ili se dogodila druga greška, funkcija *doPost* odgovara s porukom *Greška*.

5.2. Datoteke s podacima

Programsko ostvarenje baze podataka čini skupina datoteka sa tekstom zapisanim jezikom *XML*. Svaka datoteka predstavlja jedan poslužitelj za udomljenike prijavljen u imenik i sadrži podatke o poslužitelju, te podatke o udomljenicima koje dani poslužitelj objavljuje. Ime datoteke određuje se kao *„server“*, potom identifikacijski broj poslužitelja za udomljenike i ekstenzija *„.xml“*.

Na slici 5.2 prikazan je primjer jedne datoteke s podacima o poslužitelju za udomljenike i udomljenicima spremljenim na poslužitelja. Glavna oznaka koja obuhvaća dokument je *repositoryData*, i sadržava dva tipa oznaka: *serverData* i *gadget*. Oznaka *repositoryData* sadržava točno jednu oznaku *serverData* te neograničen broj oznaka *gadget*.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns1:repositoryData
    xmlns:tns1="http://www.example.org/repositoryDefinition"
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/repositoryDefinition repositoryDefinition.xsd ">
  <tns:serverData>
    <tns:serverID>42</tns:serverID>
    <tns:serverURL>www.serveri.com/server42</tns:serverURL>
  </tns:serverData>
  <tns:gadget>
    <tns:generalData>
      <tns:gadgetID>1</tns:gadgetID>
      <tns:gadgetURL>www.serveri.com/server42/gadget1</tns:gadgetURL>
      <tns:hash>1234567890</tns:hash>
      <tns:keyword>probni udomljenik</tns:keyword>
      <tns:keyword>primjer</tns:keyword>
      <tns:keyword>pour encourager les autres</tns:keyword>
    </tns:generalData>
    <tns:usageData>
      <tns:numTimesUsed>876</tns:numTimesUsed>
    </tns:usageData>
  </tns:gadget>
  <tns:gadget>
    <tns:generalData>
      <tns:gadgetID>22</tns:gadgetID>
      <tns:gadgetURL>www.serveri.com/server42/gadget12</tns:gadgetURL>
      <tns:hash>FFCD4231538F</tns:hash>
      <tns:keyword>test</tns:keyword>
      <tns:keyword>primjer</tns:keyword>
      <tns:keyword>poslužitelj</tns:keyword>
    </tns:generalData>
    <tns:usageData>
      <tns:numTimesUsed>1674</tns:numTimesUsed>
    </tns:usageData>
  </tns:gadget>
</tns1:repositoryData>
```

Slika 5.2 Primjer datoteke s podacima o poslužitelju

Oznaka *serverData* sadržava podatke o poslužitelju za udomljenike. U primjeru na slici 5.2 podaci o poslužitelju su identifikacijski broj poslužitelja kao vrijednost oznake *serverID* i mrežna adresa poslužitelja kao vrijednost oznake *serverURL*.

Svaki udomljenik objavljen na poslužitelju za udomljenike je predstavljen jednom oznakom *gadget*, koja sadrži podatke o tom udomljeniku. Na slici 5.2 prikazan je primjer sa podacima o dva udomljenika. Za svakog udomljenika su zapisani opći podaci sadržani u oznaci *generalData* i podaci o korištenju sadržani u oznaci *usageData*. Oznaka *generalData* sadrži identifikacijski broj udomljenika kao vrijednost oznake *gadgetID*, mrežnu adresu kao vrijednost oznake *gadgetURL*, sažetak programskog kôda udomljenika kao vrijednost oznake *hash* i ključne riječi kao vrijednosti oznaka *keyword*. Oznaka *usageData* sadrži broj korištenja udomljenika kao vrijednost oznake *numTimesUsed*.

5.3. Ostvarenje pristupa bazi podataka

Podsustav za pristup bazi podataka služi za promjenu podataka spremljenih u bazi. Baza podataka je ostvarena kao skupina datoteka zapisanih jezikom *XML* prema pravilima objašnjenima u poglavlju 5.2. Podsustav za pristup bazi podataka ostvaren je u razredu *XMLstorage*, koja ostvaruje funkcije zadane sučeljem opisanim u apstraktnom razredu *Repository*.

Apstraktni razred *Repository* opisuje sučelje pristupa bazi podataka sa strane poslužiteljskog podsustava. Poslužiteljski podsustav ovisno o pristigloj poruci poziva jednu od funkcija. Poruka *Prijavi* obrađuje se funkcijom *start*, poruka *Stvori* funkcijom *create*, poruka *Promijeni* funkcijom *change*, poruka *Ažuriraj* funkcijom *update* i poruka *Obriši* funkcijom *delete*. Svaka funkcija prima poruku spremljenu u objekt tipa *org.w3c.dom.Document*, te vraća poruku odgovora kao znakovni niz s tekstom zapisanim u jeziku *XML*.

Razred *XMLstorage* ostvaruje funkcije specificirane u apstraktnom razredu *Repository*, te konstruktor i privatne funkcije. Javna članska varijabla je lokacija direktorija s bazom podataka. Privatne članske varijable sadrže objekte potrebne za pretvorbu niza znakova sa tekstom zapisanim u jeziku *XML* u objekt tipa *org.w3c.dom.Document*, te pretvorbu objekta tog tipa u znakovni niz s tekstom zapisanim u jeziku *XML*.

Konstruktor razreda *XMLstorage* postavlja vrijednosti članskih varijabli. Ulazna varijabla *newDirectory* sadrži lokaciju direktorija sa datotekama baze podataka koja se sprema u javnu člansku varijablu *directory*. Privatne članske varijable popunjavaju se novostvorenim objektima.

5.3.1. Funkcija *start*

Funkcija *start* služi za obradu obje verzije poruke *Prijavi*. Prva verzija poruke *Prijavi* služi za prvu prijavu poslužitelja za udomljenike. Druga verzija poruke služi za promjenu podataka o poslužitelju za udomljenike. Funkcija prima poruku u obliku objekta tipa *org.w3c.dom.Document*. Dijagram toka funkcije *start* prikazan je na slici 5.4.

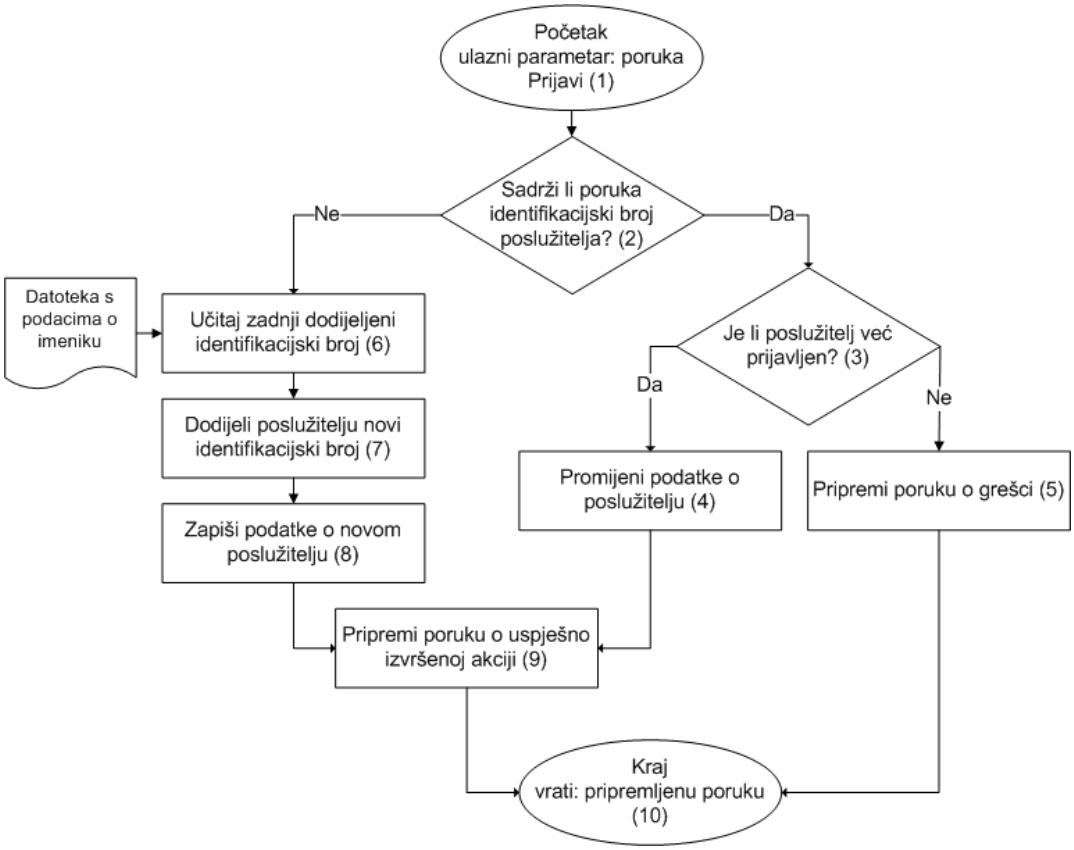
Obrada poruke *Prijavi* započinje provjerom prisutnosti podatka o identifikacijskom broju poslužitelja za udomljenike (2). Prisutnost

identifikacijskog broja poslužitelja za udomljenike označava da se poruka *Prijavi* koristi za promjenu podataka o poslužitelju. Ako identifikacijski broj nije poslan porukom *Prijavi*, učitava se datoteka sa podacima o imeniku (6).

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<tns:repositoryConfig
    xmlns:tns="http://www.example.org/repositoryConfigDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/repositoryConfigDefinition repositoryConfigDefinition.xsd ">
  <tns:lastServerID>57</tns:lastServerID>
</tns:repositoryConfig>
```

Slika 5.3 Primjer datoteke s podacima o imeniku

Datoteka s podacima o imeniku služi za pohranu zadnjeg dodijeljenog identifikacijskog broja. Na slici 5.3 prikazan je primjer datoteke sa podacima o imeniku. Oznaka *lastServerID* sadrži zadnji dodijeljeni identifikacijski broj poslužitelja za udomljenike. Datoteka se nalazi u istom direktoriju kao i datoteke baze podataka. Zadnji dodijeljeni identifikacijski broj uvećava se za jedan i dodjeljuje kao identifikacijski broj poslužitelja za udomljenike koji je poslao poruku *Prijavi* (7), te zapisuje u datoteku s podacima o imeniku. Stvara se datoteka baze podataka za poslužitelj s novim identifikacijskim brojem (8).



Slika 5.4 Dijagram toka funkcije *start*

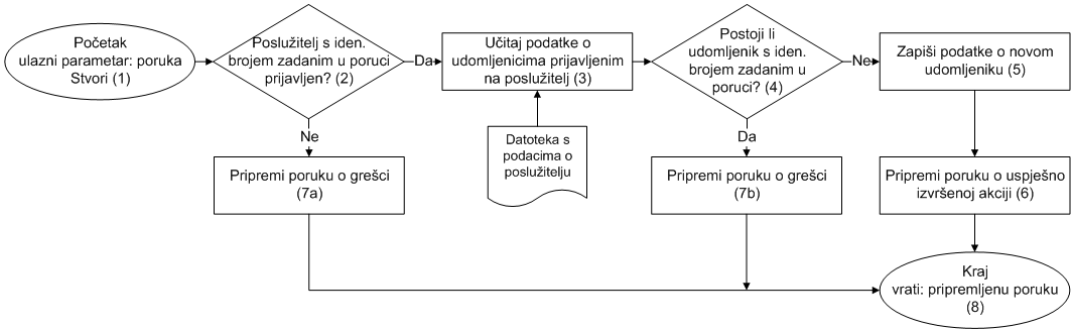
Ako je poruka *Prijavi* sadržavala identifikacijski broj, provjerava se postojanje datoteke za poslužitelj s primljenim identifikacijskim brojem (3). U slučaju da datoteka za poslužitelj s tim identifikacijskim brojem ne postoji, izvođenje funkcije *start* se prekida i odgovara se s opisom greške upisanim u poruku *Greška* (5).

Funkcija *Prijavi* postavlja vrijednosti podataka o poslužitelju spremljenih u datoteci na vrijednosti prenesene porukom (4). Potom se podaci zapisuju u datoteku i priprema odgovor o uspješnoj provedbi prijave (9), odnosno promjene podataka poslužitelja za udomljenike. Odgovor se vraća pozivatelju u obliku niza znakova (10).

5.3.2. Funkcija *create*

Funkcija *create* služi za obradu poruke *Stvori*, koja prenosi podatke o novom udomljeniku objavljenom na poslužitelju za udomljenike. Prenose se podaci iz skupine *Opći podaci o udomljeniku*, a podaci iz skupine *Podaci o korištenju udomljenika* se postavljaju na početne vrijednosti. Na slici 5.5 prikazan je dijagram toka funkcije *create*.

Funkcija prima poruku *Stvori* u objektu tipa *org.w3c.dom.Document* (1) i provjerava postoji li datoteka o poslužitelju s navedenim identifikacijskim brojem (2). Ako datoteka ne postoji prekida se izvođenje funkcije i odgovara se s opisom greške upisanim u poruku *Greška* (7a).



Slika 5.5 Dijagram toka funkcije *create*

Ako datoteka postoji, učitaju se podaci iz datoteke (3) i provjerava se postoji li udomljenik s identifikacijskim brojem navedenim u poruci (4). Ako udomljenik s navedenim identifikacijskim brojem postoji, prekida se izvođenje funkcije i odgovara se s opisom greške upisanim u poruku *Greška* (7b).

Funkcija *create* stvara novi objekt koji predstavlja udomljenik. Podaci iz skupine *Opće podaci o udomljeniku* postavljaju se na vrijednosti navedene u poruci *Stvori*. Podaci iz skupine *Podaci o korištenju udomljenika* postavljaju se na početne vrijednosti zadane u razredu *RepositoryConstants*.

Promjene se zapisuju u datoteku poslužitelja za udomljenike (5). Priprema se odgovor o uspješno provedenoj akciji (6) i vraća pozivatelju funkcije u obliku niza znakova (8).

5.3.3. Funkcija *change*

Funkcija *change* služi za obradu poruke *Promijeni*. Poruka *Promijeni* služi za promjenu podataka iz skupine *Opći podaci o udomljeniku* za udomljenik koji je upisan u bazu podataka.

Poruka *Promijeni* prosljeđuje se funkciji *change* kao objekt tipa *org.w3c.dom.Document*. Iz poruke se preuzima identifikacijski broj poslužitelja za udomljenike i provjerava postoji li datoteka o poslužitelju s navedenim identifikacijskim brojem. Ako datoteka ne postoji, odgovara se s opisom greške upisanim u poruku *Greška*.

Funkcija učitava datoteku o poslužitelju u radnu memoriju, i provjerava postoji li udomljenik s identifikacijskim brojem navedenim u poruci. Ako ne postoji udomljenik s tim identifikacijskim brojem, odgovara se s opisom greške upisanim u poruku *Greška*.

Podataci iz skupine *Opći podaci o udomljeniku* postavlja se na vrijednost navedenu u poruci *Promijeni*. Ako podatak nije bio naveden u poruci *Promjeni*, vrijednost se ne mijenja. Vrijednosti sadržane u oznaci *keyword* se ne mijenjaju nego dodaju u opis udomljenika ako atribut *delete* nije postavljen, ili je postavljen na *false*. Ako je atribut *delete* oznake *keyword* postavljen na *true*, traži se oznaka *keyword* u postojećem opisu udomljenika sa vrijednošću jednakom primljenoj. Ako takva oznaka postoji, briše se iz opisa udomljenika. Ako takva oznaka ne postoji, ignorira se oznaka navedena u poruci *Promijeni*. Nakon što su sve promjene izvršene, ako ne postoji više niti jedna oznaka *keyword* za udomljenik, sve promjene se odbacuju i odgovara se s opisom greške upisanim u poruku *Greška*. Ukoliko postoji barem jedna oznaka *keyword*, promjene se zapisuju u datoteku o poslužitelju.

Promjene se zapisuju u datoteku poslužitelja za udomljenike. Priprema se odgovor o uspješno provedenoj akciji i vraća pozivatelju funkcije u obliku niza znakova.

5.3.4. Funkcija *update*

Funkcija *update* služi za obradu poruke *Ažuriraj*. Poruka *Ažuriraj* služi za promjenu podataka iz skupine *Podaci o korištenju udomljenika*.

Poruka *Ažuriraj* prosljeđuje se funkciji *update* kao objekt tipa *org.w3c.dom.Document*. Funkcija *update* provjerava postoji li datoteka o poslužitelju s identifikacijskim brojem navedenim u primljenoj poruci. Ako datoteka ne postoji, odgovara se s opisom greške upisanim u poruku *Greška*.

Funkcija *update* potom učitava datoteku o poslužitelju u radnu memoriju i provjerava je li udomljenik s identifikacijskim brojem navedenim u poruci prijavljen za odabrani poslužitelj. Ako udomljenik s tim identifikacijskim brojem nije prijavljen, odgovara se s opisom greške upisanim u poruku *Greška*.

Svaki podatak iz skupine *Podaci o korištenju udomljenika* postavlja se na vrijednost navedenu u primljenoj poruci. Promjene se potom zapisuju u datoteku o poslužitelju.

Promjene se zapisuju u datoteku poslužitelja za udomljenike. Priprema se odgovor o uspješno provedenoj akciji i vraća pozivatelju funkcije u obliku niza znakova.

5.3.5. Funkcija *delete*

Funkcija *delete* služi za obradu poruke *Obriši*, koja prenosi zahtjev za brisanjem podataka o udomljeniku. Poruka *Obriši* prenosi identifikacijski broj poslužitelja za udomljenike i identifikacijski broj udomljenika.

Funkcija prima poruku *Ažuriraj* kao objekt tipa *org.w3c.dom.Document*. Provjerava se postoji li datoteka o poslužitelju s identifikacijskim brojem navedenim u primljenoj poruci. Ako datoteka ne postoji, odgovara se s opisom greške upisanim u poruku *Greška*.

Datoteka o poslužitelju učita se u radnu memoriju i obavlja provjera postoje li podaci o udomljeniku s identifikacijskim brojem navedenim u poruci. Ako

udomljenik s tim identifikacijskim brojem nije prijavljen, odgovara se s opisom greške upisanim u poruku *Greška*.

Podaci o udomljeniku se brišu, te se promjene spremaju u datoteku o poslužitelju za udomljenike. Priprema se odgovor o uspješno provedenoj akciji i vraća pozivatelju funkcije u obliku niza znakova.

5.4. Ostvarenje podsustava za pretraživanje

Podsustav za pretraživanje služi za pretragu udomljenika prijavljenih u imenik. Sučelje podsustava za pretraživanje prema poslužiteljskom podsustavu definirano je apstraktnim razredom *Search*. Razred *PrivateDetective* ostvaruje funkcije definirane sučeljem.

Apstraktni razred *Search* definira funkciju *search*, koja prima poruku *Pretraži* kao objekt tipa tipa *org.w3c.dom.Document*. Funkcija *search* kao odgovor vraća znakovni niz koji sadrži poruku zapisanu jezikom *XML*.

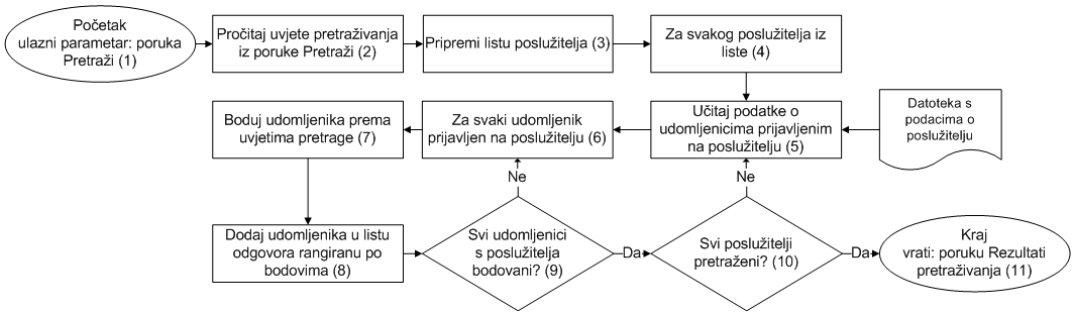
Razred *PrivateDetective* ostvaruje funkcionalnost pretraživanja baze podataka za udomljenicima koji odgovaraju uvjetima navedenim u poruci *Pretraži*. Javne funkcije razreda su konstruktor i ostvarenje funkcije *search* specificirane u apstraktnom razredu *Search*. Lokacija direktorija s datotekama baze podataka je javna članska varijabla. Privatna članska varijabla se koristi za spremanje objekta koji služi za pretvaranje niza znakova sa tekstom u jeziku *XML* u objekt tipa *org.w3c.dom.Document*.

Konstruktor razreda *PrivateDetecive* popunjava vrijednosti članskih varijabli. Ulazna varijabla *newDirectory* sadrži lokaciju direktorija sa datotekama baze podataka koja se sprema u javnu člansku varijablu *directory*. Privatna članska varijabla popunjavaja se s novostvorenim objektom.

5.4.1. Funkcija *search*

Funkcija *search* služi za obradu poruke *Pretraži*. Poruka *Pretraži* postavlja uvjete pretraživanja podataka o udomljenicima prijavljenim u imenik. Odgovor funkcije čini poruka *Rezultati pretraživanja* ili poruka *Greška*. Dijagram toka funkcije *search* prikazan je na slici 5.6.

Ulazni parametar funkcije čini poruka *Pretraži* kao objekt tipa *org.w3c.dom.Document* (1). Funkcija prvo određuje broj opisa udomljenika koje treba vratiti kao rezultat (2). Ako broj opisa nije definiran u poruci *Pretraži*, koristi se vrijednost definirana u razredu *RepositoryConstants*.



Slika 5.6 Dijagram toka funkcije *search*

Funkcija *search* priprema listu poslužitelja koje treba pretražiti. Ako u poruci *Pretraži* nije naveden identifikacijski broj poslužitelja, pretraga se obavlja nad svim prijavljenim poslužiteljima (3). Ako poruka *Pretraži* sadrži identifikacijski broj poslužitelja, funkcija *search* u listu poslužitelja dodaje samo navedeni poslužitelj. Ako poslužitelj s navedenim identifikacijskim brojem nije prijavljen na imenik, odgovara se s opisom greške upisanim u poruku *Greška*.

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:mesSearch
    xmlns:tns="http://www.example.org/messageDefinition"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/messageDefinition messageDefinition.xsd ">
  <tns:gadgetURL>.hr</tns:gadgetURL>
  <tns:keyword>primjer</tns:keyword>
  <tns:keyword>raspodijeljen</tns:keyword>
</tns:mesSearch>
```

Slika 5.7 Poruka *Pretraži* s uvjetima pretraživanja

Za svaki poslužitelj u listi poslužitelja, funkcija *search* učitava datoteku s podacima o poslužitelju za udomljenike (5). Svaki udomljenik prijavljen na poslužitelja se boduje prema uvjetima pretrage navedenim u poruci *Pretraži* (7). Pri usporedbi identifikacijskog broja udomljenika prihvaćaju se samo brojevi jednaki zadanom u poruci. Podaci o mrežnoj adresi, sažetku programskog kôda i ključnim riječima provjeravaju se sadrže li znakovni niz definiran pod istom oznakom u poruci *Pretraži*.

Udomljenk dobiva bodove za svaki uvjet koji zadovoljava, te se rangira u listi po broju bodova (8). Svaki udomljenik sa barem jednim bodom ulazi u listu

udomljenika koji se šalju kao odgovor. Ako lista udomljenika dosegne maksimalni broj rezultata, odbacuju se udomljenici s najmanjim brojem bodova.

A
```
<tns:gadget>
    <tns:generalData>
      <tns:gadgetID>435</tns:gadgetID>
      <tns:gadgetURL>www.udomljenici.hr/udomljenik-primjer</tns:gadgetURL>
      <tns:hash>0ffffedc423a</tns:hash>
      <tns:keyword>primjer udomljenika</tns:keyword>
      <tns:keyword>primjer u sustavu za prikupljanje i objedinjavanje podataka</tns:keyword>
    </tns:generalData>
    <tns:usageData>
      <tns:numTimesUsed>1469</tns:numTimesUsed>
    </tns:usageData>
</tns:gadget>
```

B
```
<tns:gadget>
    <tns:generalData>
      <tns:gadgetID>347</tns:gadgetID>
      <tns:gadgetURL>www.udomljenici.com/udomljenik-primjer</tns:gadgetURL>
      <tns:hash>0ffffedc423a</tns:hash>
      <tns:keyword>primjer</tns:keyword>
      <tns:keyword>udomljenik</tns:keyword>
      <tns:keyword>sustav za prikupljanje i objedinjavanje podataka</tns:keyword>
    </tns:generalData>
    <tns:usageData>
      <tns:numTimesUsed>3475</tns:numTimesUsed>
    </tns:usageData>
</tns:gadget>
```

Slika 5.8 Primjer podataka o 2 udomljenika

Na slici 5.7 prikazan je primjer poruke *Pretraži* s uvjetima pretraživanja. Traženi su udomljenici čija mrežna adresa sadrži niz *„.hr“*, te ključni izrazi sadrže nizove *„primjer“* i *„raspodijeljen“*. Na slici 5.8 prikazani su opisi dva udomljenika. Sažetak programskog kôda upućuje na činjenicu da je programski kôd jednak za oba, no udomljenici su objavljeni na dvije različite domene i opisani različitim ključnim izrazima. Mrežna adresa udomljenika A, sadržana pod oznakom *gadgetURL,* sadrži niz *„.hr“* zadan u poruci *Pretraži* zbog čega je udomljeniku dodijeljen jedan bod. Dva ključna izraza udomljenika A, sadržana pod oznakama *keyword*, sadržavaju niz *„primjer“*, zbog čega je udomljeniku A dodijeljeno još dva boda. Ključni izrazi udomljenika A nisu sadržavali niz *„raspodijeljen“*, stoga udomljenik A ima ukupno tri boda. Mrežna adresa udomljenika B ne sadrži niz *„.hr“*, stoga udomljenik ne dobiva bodove za mrežnu adresu. Jedan ključni izraz udomljenika B sadrži niz *„primjer“*, stoga udomljenik B dobiva jedan bod. Niti jedan ključni izraz udomljenika ne sadrži niz *„raspodijeljen“*, stoga udomljenik B ima ukupno jedan bod. U krajnjem poretku udomljenika, udomljenik A bi bio na višoj poziciji od udomljenika B.

Nakon što su pregledane sve datoteke o poslužiteljima i stvorena lista udomljenika koji najbolje zadovoljavaju uvjete (10), priprema se odgovor kao

objekt tipa *org.w3c.dom.Document*, koji predstavlja poruku *Rezultati pretraživanja*. U poruci se zapisuju svi udomljenici iz liste. Za svaki udomljenik se zapisuje identifikacijski broj poslužitelja za udomljenike na kojem je prijavljen, te svi podaci o udomljeniku zapisani u imeniku. Funkcija *search* pretvara poruku *Rezultati pretraživanja* u znakovni niz i vraća ga pozivatelju (11).

6. Zaključak

Računalni oblak za potrošačko programiranje je raspodijeljeni sustav koji omogućava potrošačima izradu primjenskih sustava bez potrebe za poznavanjem osnova programiranja. Primjenski sustavi sastavljeni su od komponenata potrošačkog programiranja poput *Yahoo! Widgets* ili *Google Gadgets* udomljenika. Potrošači stvaraju, mijenjaju i koriste udomljenike objavljene na specijaliziranim poslužiteljima za udomljenike.

U ovom diplomskom radu osmišljen je *sustav za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje* koji omogućava pretraživanje udomljenika objavljenih na razini računalnog oblaka. Definirana je arhitektura *sustava za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje* i protokol *raspodijeljenog prijavljivanja i pretraživanja udomljenika* (*RPPU*), te programski ostvaren centralni dio sustava.

Sustav za prikupljanje i objedinjavanje podataka nadograđuje sudionike računalnog oblaka sa ulogama imenika, poslužitelja za udomljenike i korisnika. Sustav predviđa postojanje točno jednog imenika, te proizvoljnog broja poslužitelja za udomljenike i korisnika. Ulogu poslužitelja za udomljenike preuzimaju poslužitelji koji mrežno objavljuju udomljenike, a ulogu korisnika može preuzeti bilo koji sudionik računalnog oblaka. Komunikacija između elemenata u sklopu *sustava za prikupljanje i objedinjavanje podataka* odvija se protokolom *RPPU*.

Imenik je mrežno dostupna centralna baza podataka koja sadržava podatke o udomljenicima i poslužiteljima za udomljenike. Imenik održava bazu podataka prema porukama dobivenim od poslužitelja za udomljenike, te obavlja pretraživanje baze podataka na zahtjev korisnika. Uvjete pretraživanja postavljaju korisnici u zahtjevu za pretraživanjem. Poslužitelji za udomljenike obavještavaju imenik o stvaranju novih te promjenama, korištenju i brisanju postojećih udomljenika.

U sklopu diplomskog rada imenik je ostvaren u programskom jeziku *Java* koristeći tehnologiju *Java Servlet*, te je postavljen i ispitan na poslužitelju *Apache Tomcat* koristeći programsko okruženje *Eclipse*. Imenik je podijeljen u

podsustave s definiranim sučeljima. Komunikacija s imenikom ostvaruje porukama opisanim u sklopu protokola *RPPU*, koje se zapisuju jezikom *XML* i prenose zahtjevima *POST* i *GET* protokola *HTTP*.

Baza podataka u sklopu imenika ostvarena je u obliku datoteka s podacima zapisanim u jeziku *XML*. Odabir podataka o poslužiteljima za udomljenike i udomljenicima koji su spremljeni na imenik potrebno je prilagoditi računalnom oblaku u kojemu se primjenjuje *sustav za prikupljanje i objedinjavanje podataka*.

Mogućnost za daljnji razvoj *sustav za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje* leži u proširenju sustava s dodatnim imenicima kako bi se osiguralo svojstvo razmjernog rasta sustava. U tu svrhu potrebno je razviti mehanizam i skup poruka za usklađivanje podataka spremljenih u imenicima. Unaprjeđenje programskog ostvarenja imenika zasniva se na nadogradnji postojeće baze podataka ili zamjeni s alternativnom bazom podataka. Jedna od mogućih nadogradnji postojeće baze podataka je programsko ostvarenje mehanizama za siguran paralelni pristup bazi podataka.

7. Literatura

[1] Tanenbaum, A., van Steen, M., Distributed Systems: Principles and Paradigms, USA, Prentice-Hall, Inc., 2002.

[2] Basic Information, *Geppeto Consumer Programming Tool*, http://www.ris.fer.hr/, svibanj 2010.

[3] Lovrek, I., Raspodijeljeni sustavi, materijali za predavanja, 2008., svibanj 2010.

[4] Coulouris, G., Dollimore, J., Kindberg, T., Distributed Systems: Concepts and Design, 4. izdanje, Pearson Education Ltd., 2005.

[5] Vaquero, L. M., Rodero-Merino, L., Caceres, J., Lindner, M., A Break in the Clouds: Towards a Cloud Definition, ACM SIGCOMM Computer Communication Review, Svezak 39., 1 (2009.), str. 50-55

[6] Srbljić, S., Žužak, I., Računarstvo zasnovano na uslugama, materijali za predavanja, 2008., svibanj 2010.

[7] Danielson, K., Distinguishing Cloud Computing from Utility Computing, 26.3.2008., *SaaS Week*, http://www.ebizq.net/blogs/saasweek/2008/03/distinguishing_cloud_computing/, svibanje 2010.

[8] Baritchi, A., Cloud Computing: The Mainframe Reincarnated, 16.12.2008., http://de.sys-con.com/node/723583, svibanj 2010.

[9] Bodoff, S., Java Servlet Technology, http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html, svibanj 2010.

[10] Interface Servlet, http://download.oracle.com/docs/cd/E17477_01/javaee/1.3/api/javax/servlet/Servlet.html, svibanj 2010.

[11] Apache Tomcat, http://tomcat.apache.org/, svibanj 2010.

[12] Servlets & JSP: Overview and Setup , *Beginning and Intermediate-Level Servlet, JSP, and JDBC Tutorials*, 2010., http://courses.coreservlets.com/Course-Materials/csajsp2.html, svibanj 2010.

[13] Class GenericServlet, http://download.oracle.com/docs/cd/E17477_01/javaee/1.3/api/javax/servlet/GenericServlet.html, svibanj 2010.

[14] Class HttpServlet http://download.oracle.com/docs/cd/E17477_01/javaee/1.3/api/javax/servlet/http/HttpServlet.html, svibanj 2010.

8. Sažetak

Sustav prikupljanja i objedinjavanja podataka u računalnom oblaku za potrošačko programiranje

U ovom diplomskom radu osmišljen je *sustav za prikupljanje i objedinjavanje podataka u računalnom oblaku za potrošačko programiranje*, te protokol *raspodijeljenog prijavljivanja i pretraživanja udomljenika* (*RPPU*). *Sustav za prikupljanje i objedinjavanje podataka* omogućuje jedinstveno pretraživanje komponenata potrošačkog programiranja objavljenih na poslužiteljima u sklopu računalnog oblaka. Podacima o komponentama pristupa se pomoću centralizirane baze podataka o komponentama.

Ključne riječi: prikupljanje podataka, objedinjavanje podataka, raspodijeljeni sustav, računalni oblak, potrošačko programiranje

9. Summary

Data Collection and Aggregation System for Consumer Programming Cloud

This thesis contains description of architecture for *data collection and aggregation system for consumer programming cloud*, as well as *distributed report and discovery of widgets* protocol. *Data collection and aggregation system* is used to simplify discovery of consumer programming components published on servers that are a part of consumer programming cloud. Centralized database is used for access to component descriptions.

Keywords: data collection, data aggregation, distributed system, cloud computing, consumer programming

10. Dodatak A: Definicija poruka korištenih u programskom ostvarenju

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
   xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.example.org/messageDefinition"
   xmlns:tns="http://www.example.org/messageDefinition"
   elementFormDefault="qualified">

   <element name="gadgetID" type="positiveInteger"></element>
   <element name="serverID" type="positiveInteger"></element>
   <element name="gadgetURL" type="string"></element>
   <element name="serverURL" type="string"></element>
   <element name="hash" type="string"></element>
   <element name="keyword">
      <complexType>
       <simpleContent>
        <extension base="string">
         <attribute name="delete" type="boolean" default="false"></attribute>
        </extension>
       </simpleContent>
      </complexType>
   </element>
   <element name="numSearchResults" type="positiveInteger"></element>
   <element name="numTimesUsed" type="positiveInteger"></element>

   <complexType name="generalDataType">
      <sequence>
         <element ref="tns:gadgetID" maxOccurs="1" minOccurs="1"></element>
         <element ref="tns:gadgetURL" maxOccurs="1" minOccurs="1"></element>
         <element ref="tns:hash" maxOccurs="1" minOccurs="1"></element>
         <element ref="tns:keyword" maxOccurs="unbounded" minOccurs="1"></element>
      </sequence>
   </complexType>

   <complexType name="usageDataType">
      <sequence>
         <element ref="tns:numTimesUsed"></element>
      </sequence>
   </complexType>

   <complexType name="mesCreateGadgetType">
      <sequence>
         <element ref="tns:serverID"></element>
         <element ref="tns:generalData"></element>
      </sequence>
   </complexType>

   <complexType name="mesDeleteGadgetType">
      <sequence>
         <element ref="tns:serverID"></element>
         <element ref="tns:gadgetID"></element>
      </sequence>
   </complexType>

   <complexType name="mesChangeGadgetType">
      <sequence>
         <element ref="tns:serverID"></element>
```

```
      <element ref="tns:changeGadget"></element>
   </sequence>
</complexType>

<complexType name="mesUpdateGadgetType">
   <sequence>
      <element ref="tns:serverID"></element>
      <element ref="tns:gadgetID"></element>
      <element ref="tns:usageData"></element>
   </sequence>
</complexType>

<complexType name="changeGeneralDataType">
   <sequence>
      <element ref="tns:gadgetID" maxOccurs="1" minOccurs="1"></element>
      <element ref="tns:gadgetURL" maxOccurs="1" minOccurs="0"></element>
      <element ref="tns:hash" maxOccurs="1" minOccurs="0"></element>
      <element ref="tns:keyword" maxOccurs="unbounded" minOccurs="0"></element>
   </sequence>
</complexType>

<complexType name="mesSearchType">
   <sequence>
      <element ref="tns:numSearchResults" maxOccurs="1" minOccurs="0"></element>
      <element ref="tns:serverID" maxOccurs="1" minOccurs="0"></element>
      <element ref="tns:gadgetID" maxOccurs="1" minOccurs="0"></element>
      <element ref="tns:gadgetURL" maxOccurs="1" minOccurs="0"></element>
      <element ref="tns:hash" maxOccurs="1" minOccurs="0"></element>
      <element ref="tns:keyword" maxOccurs="unbounded" minOccurs="0"></element>
   </sequence>
</complexType>

<complexType name="mesServerStartupType">
   <sequence>
      <element ref="tns:serverID" maxOccurs="1" minOccurs="0"></element>
      <element ref="tns:serverURL"></element>
   </sequence>
</complexType>

<complexType name="gadgetDataType">
   <sequence>
      <element ref="tns:generalData"></element>
      <element ref="tns:usageData"></element>
   </sequence>
</complexType>

<complexType name="serverDataType">
   <sequence>
      <element ref="tns:serverID"></element>
      <element ref="tns:serverURL"></element>
   </sequence>
</complexType>

<element name="generalData" type="tns:generalDataType"></element>
<element name="usageData" type="tns:usageDataType"></element>
<element name="changeGadget" type="tns:changeGeneralDataType"></element>
<element name="serverData" type="tns:serverDataType"></element>
<element name="gadget" type="tns:gadgetDataType"></element>
<element name="mesChangeGadget" type="tns:mesChangeGadgetType">
   <annotation>
```

```
            <documentation>changes general gadget data;
success reported with gadget ID</documentation>
        </annotation>
    </element>
    <element name="mesCreateGadget" type="tns:mesCreateGadgetType">
        <annotation>
            <documentation>creates new gadget info on server;
usage info is set to defaults;
success reported with gadget ID</documentation>
        </annotation>
    </element>

    <element name="mesDeleteGadget" type="tns:mesDeleteGadgetType">
        <annotation>
            <documentation>deletes gadget info from server;
success reported with gadget ID</documentation>
        </annotation>
    </element>

    <element name="mesSearch" type="tns:mesSearchType">
        <annotation>
            <documentation>returns search reply message;
gadgets enter as possible reponses if they match at least one given criteria;
those that match more will be higher on list;
if numSearchResults is not given, all gadgets that match at least one given criteria will be
returned</documentation>
        </annotation>
    </element>

    <element name="mesServerStartup" type="tns:mesServerStartupType">
        <annotation>
            <documentation>if serverID is present, will try to find existing server ID file (error if it
cannot be found) - serverURL will be updated in that case - success reported with message
serverID;
if only serverURL is sent, repository will try to create new server file - success reported with
server ID</documentation>
        </annotation>
    </element>

    <element name="mesUpdateGadget" type="tns:mesUpdateGadgetType">
        <annotation>
            <documentation>changes usage info of given gadget;
success reported with gadget ID</documentation>
        </annotation>
    </element>

    <element name="searchReplyGadget" type="tns:searchReplyGadgetType"></element>

    <complexType name="mesSearchReplyType">
        <sequence>
            <element ref="tns:searchReplyGadget" maxOccurs="unbounded"
              minOccurs="0">
            </element>
        </sequence>
    </complexType>

    <element name="mesSearchReply" type="tns:mesSearchReplyType">
        <annotation>
            <documentation>response to Search message; contains a list of gadgets matching
criteria</documentation>
```

```
      </annotation>
   </element>

   <complexType name="searchReplyGadgetType">
      <sequence>
         <element ref="tns:serverID"></element>
         <element ref="tns:generalData"></element>
         <element ref="tns:usageData"></element>
      </sequence>
   </complexType>

   <simpleType name="mesChangeSuccessType">
      <annotation>
         <documentation>response to successful mesChangeGadget message</documentation>
      </annotation>
      <restriction base="string"></restriction>
   </simpleType>

   <simpleType name="mesCreateSuccessType">
      <annotation>
         <documentation>response to successful mesCreateGadget message</documentation>
      </annotation>
      <restriction base="string"></restriction>
   </simpleType>

   <simpleType name="mesDeleteSuccessType">
      <annotation>
         <documentation>response to successful mesDeleteGadget message</documentation>
      </annotation>
      <restriction base="string"></restriction>
   </simpleType>

   <simpleType name="mesServerChangeSuccessType">
      <annotation>
         <documentation>response to successful mesServerStartup message if it was used for
changing server data (i.e. it contained valid serverID)</documentation>
      </annotation>
      <restriction base="string"></restriction>
   </simpleType>

   <simpleType name="mesServerInitSuccessType">
      <annotation>
         <documentation>response to successful mesServerStartup if it was used as server
initialization message;
it contains newly assigned serverID</documentation>
      </annotation>
      <restriction base="string"></restriction>
   </simpleType>

   <simpleType name="mesUpdateSuccessType">
      <annotation>
         <documentation>response to successful mesUpdateGadget message</documentation>
      </annotation>
      <restriction base="string"></restriction>
   </simpleType>

   <simpleType name="mesErrorType">
      <annotation>
         <documentation>response if error has occured (to any message);
contains description of error</documentation>
```

```
      </annotation>
      <restriction base="string"></restriction>
   </simpleType>

   <element name="error" type="tns:mesErrorType"></element>
   <element name="mesChangeSuccess" type="tns:mesChangeSuccessType"></element>
   <element name="mesCreateSuccess" type="tns:mesCreateSuccessType"></element>
   <element name="mesDeleteSuccess" type="tns:mesDeleteSuccessType"></element>
   <element name="mesServerInitSuccess" type="tns:mesServerInitSuccessType"></element>
   <element name="mesServerChangeSuccess"
type="tns:mesServerChangeSuccessType"></element>
   <element name="mesUpdateSuccess" type="tns:mesUpdateSuccessType"></element>
</schema>
```