SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 176

Osmišljavanje računalnog oblaka

Neven Ćubić

Zagreb, lipanj 2011.

Sadržaj

1. Uvod

Kroz povijest razvoj računala najviše se temeljio na povećavanju računalne moći. Redovito su izlazile nove verzije procesora različitih namjena koji bi dodatno podigli granicu u odnosu na prijašnju generaciju. Pojavom višejezgrenih procesora pojačao se i razvoj metoda paralelizacije, pa osim brzine proizvođači su pojačali i razvoj programske potpore za učinkovito iskorištavanje novih arhitektura, a programi su se počeli prilagođavati za njih. No za krajnjeg korisnika osim kupovanja novih računala danas postoji još jedna vrlo popularna alternativa. Razvojem Interneta, njegovim širenjem i povećavanjem brzina pristupa omogućeno je da se određeni zadaci obavljaju udaljeno. Ideja je da korisnik umjesto ulaganja u nova računala i opremu može s postojećom ili slabijom opremom dobiti uslugu za koju je potrebna naprednija arhitektura. Sa tim uređajima korisnik pristupa preko Interneta sa zahtjevom na koji dobiva natrag rezultat. To se naravno naplaćuje no za neke korisnike je isplativije od nabavljanja nove opreme zbog posla koji im nije temelj poslovanja. Taj se oblik pružanja usluga popularno naziva računarstvo u oblacima (engl. cloud computing) i temelji se na iznajmljivanju vlastite arhitekture za obavljanje raznih zadataka, od običnih usluga poput pisanja dokumenata do izvođenja složenijih virtualnih računala. Sva komunikacija između klijenta i iznajmljenih računala odvija se preko Interneta.

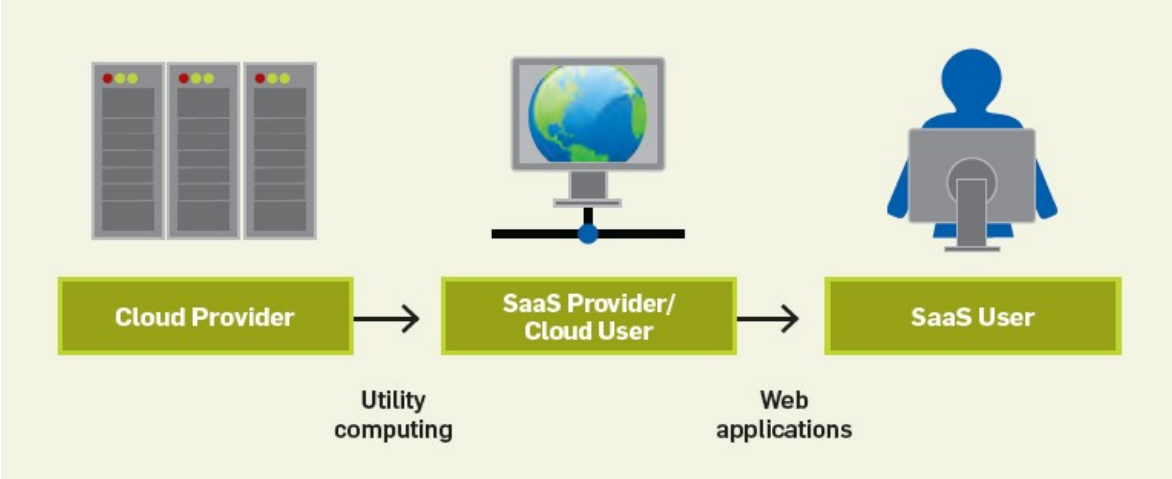
Cilj je ovog rada upoznati se s računalnim oblacima pregledom popularnih rješenja na tom području i razvojem vlastitog protokola i arhitekture za pružanje sličnih usluga. Razvijena arhitektura pruža obradu relativno jednostavnih zadataka koji se temelje na komunikaciji preko protokola TCP i izmjenjivanja zahtjeva i odgovora u formatu temeljenom na XML-u. U prvom poglavlju govori se općenito o računalnim oblacima i nekim popularnijim rješenjima i konceptima, drugo poglavlje opisuje protokol razvijen u sklopu ovog rada, a treće poglavlje opisuje napravljenu arhitekturu i programe. Oni su priloženi uz ovaj rad i predstavljaju samo jedan od načina na koji se izmišljeni protokol može ostvariti. Sam je protokol dosta prilagodljiv i ostavlja dosta slobode pri izradi implementacije i konfiguraciji mreže.

2. Računalni oblaci

Računarstvo u oblaku postao je popularan termin koji se često koristi u reklamiranju usluga no nije uvijek jasno što točno predstavlja. Zablude oko samog termina idu toliko daleko da je CEO Oraclea, Larry Ellison, jednom izjavio kako je računarstvo u oblacima danas definirano tako da obuhvaća sve što već radimo [2]. Najjednostavnija definicija kaže da je računarstvo u oblacima model računarstva u kojem se usluge postavljaju na Internet i korisnici im pristupaju prema određenim uvjetima [3]. Malo složenija definicija kaže da je oblak skup računala i programa na koje se postavlja usluga koja se pruža preko Interneta, a računarstvo u oblacima obuhvaća oblak i usluge koje se postavljaju na oblak [1]. Nije ni dogovorena podjela između tipova usluga koje se mogu postaviti na oblak, iako često se spominju tri tipa usluga. Prva je program kao usluga (engl. *Software as a service, SaS*), druga platforma kao usluga (engl. *Platform as a service, PaS*), a treća infrastuktura kao usluga (engl. *Infrastructure as a service, IaS*).

Program kao usluga relativno je stari koncept koji se odnosi na pristup aplikaciji preko Interneta, najčešće preko internetskog preglednika, uz plaćanje određene naknade. Umjesto kupovanja programa i instalacije na lokalno računalo, SaS definira da korisnik samo plaća pristup i korištenje programa koji se izvodi na udaljenom računalu. Ipak nije usklađeno koji oblici SaS-a jesu oblak, a koji nisu. Na primjer pristup Googleovom servisu Gmail radi pisanja i primanja elektroničke pošte obično se ne smatra oblikom oblaka, no ako netko iznajmi domenu i poslužitelja za elektroničku poštu od Googlea za vlastite zaposlenike tada se govori o računarstvu u oblacima. To je više definicija poslovnog modela nego tehnička definicija. Pretpostavlja se da je to oblik poslovanja u kojem pružatelj određene usluge iznajmljuje tuđu arhitekturu na koju postavlja vlastitu uslugu. U gore navedenom primjeru vlasnik tvrtke iznajmljuje poslužitelj elektroničke pošte koji Google postavlja na vlastitu infrastrukturu. Vlasnik pritom plaća Googleu korištenje njihove infrastrukture i time izbjegava troškove kupovanja vlastite opreme za posluživanje elektroničke pošte. Doduše danas se često svi tipovi usluga koji za korisnika nešto spremaju preko Interneta reklamiraju kao računarstvo u oblacima. Ostali oblici, PaS i IaS, zapravo su slični modeli i razlika između njih nije točno definirana. Za razliku od SaS-a gdje korisnici iznajmljuju program ili skup programa koji se izvode udaljeno, kod modela PaS i IaS korisnici iznajmljuju samu infrastrukturu i sami određuju kako će ju iskoristiti. Generalno model platforme kao usluge definira da korisnik sam razvija i postavlja vlastite programe u oblak, dok kod infrastrukture kao usluge korisnik iznajmljuje nekakav oblik virtualnog računala koji se izvodi u oblaku.

Računarstvo u oblacima dodatno se može podijeliti i na privatne i na javne oblake. Privatni oblak je infrastruktura koju tvrtka pruža vlastitim zaposlenicima. Prednost je optimizacija korištenja tvrtkine opreme, svi korisnici se spajaju na istu infrastrukturu s vlastitim zahtjevima i upravljački programi osiguravaju optimalnu dodjelu i iskorištenost resursa. Obično kad se govori o računarstvu u oblacima misli se na javne oblake. To je vrsta računarstva u oblacima gdje svatko može pristupiti infrastrukturi uz naknadu. Može se opisati kao kombinacija modela SaS i modela uslužnog računarstva (engl. *utility computing*)[3]. Taj opis kaže da su korisnici uslužnog računarstva zapravo pružatelji usluga koji svoje usluge postavljaju u oblak. Oni plaćaju naknadu vlasniku oblaka, a njihovi korisnici plaćaju naknadu njima za korištenje usluga.



Slika 2.1: Primjer javnog oblaka[1].

2.1. Prednosti i nedostaci računalnog oblaka

Najveća prednost računalnog oblaka jest korištenje po potrebi. Korisnici ne moraju kupovati vlastitu opremu već plaćaju korištenje oblaka. Pritom plaćaju koliko im treba umjesto iznajmljivanja uz pretplatu ili prethodnog rezerviranja resursa na duže vrijeme. Korisnici mogu platiti korištenje sustava na nekoliko minuta ili sati, pritom odrediti koliki im resursi trebaju i otpustiti te resurse čim su gotovi s njima. Takav oblik poslovanja posebno je prigodan za manje tvrtke koje tek počinju s poslovanjem i mogu izbjeći nabavu skupe opreme za izvršavanje određenih poslova. Takav način korištenja omogućuje vlasnicima oblaka da ih učinkovito iznajmljuju i po potrebi dodjeljuju resurse pojedinim korisnicima na temelju potreba i prioriteta. Već je napomenuto da prilikom definiranja oblaka postoje mnoge nejasnoće. Najveća je što se smatra oblakom i generalno se to određuje na temelju veličine infrastrukture koja ostvaruje oblak. Kod velikih infrastruktura iskorištenost i učinkovita dodjela resursa je vrlo bitna za dobro poslovanje. Iz perspektive korisnika stvara se dojam neograničenih računalnih resursa koje koriste po potrebi. To se obično postiže metodama virtualizacije i pametnim dodjeljivanjem resursa.

Najvažniji nedostaci su dostupnost i sigurnost. Od korisnika se traži da im poslovanje ovisi o uslugama smještenim na tuđoj infrastrukturi. Ako dođe do problema i infrastruktura postane nedostupna to im može izazvati velike poslovne gubitke. Osim toga same usluge mogu sadržavati tajne podatke koji onda na oblaku moraju biti prikladno osigurani. Pružatelj oblaka zato mora uspostaviti odnos s klijentima koji se temelji na povjerenju i osigurava privatnost i zaštitu podataka. Ako dođe do narušavanja sigurnosti i gubljenja podataka klijenata to može uništiti pružatelja. Isto se može dogoditi i ako je usluga nepouzdana i previše vremena je izvan pogona. Korisnici neće ići pružatelju koji im ne može garantirati veliku pouzdanost oblaka.

2.2. Primjeri komercijalnih računalnih oblaka

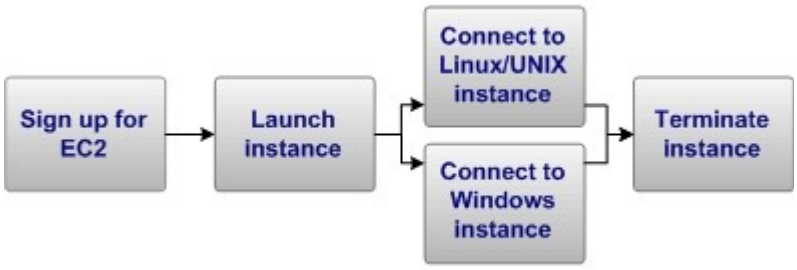
Danas većina velikih IT korporacija pruža neki oblik računarstva u oblacima. Najpoznatiji na tom području su Google i Amazon. Njihove usluge su često podijeljene na besplatni i komercijalni dio. Besplatni dio ili sadržava jednostavne usluge koje se ni ne mogu smatrati modelom računarstva u oblacima ili dopušta korištenje oblaka ali uz granice na resurse dok se ne plaća. Na primjer manje memorije za aplikaciju korisnika i određen broj dopuštenih zahtjeva po satu. U sljedećem tekstu ukratko su opisani neki od poznatijih komercijalnih oblaka.

2.2.1. Google App Engine

Googleova je usluga odličan primjer računarstva u oblacima. Korisnici sami razvijaju svoje web-aplikacije koje postavljaju na Googleovu infrastrukturu. Aplikacije mogu biti pisane za izvođenje u radnom okruženju jezika Java, Go ili Python. Korisnik napiše vlastitu aplikaciju, napravi račun na App Engine servisu i šalje ju na Googleove poslužitelje. Usluga je besplatna za određenu količinu iskorištenih resursa, a ako korisnici žele više resursa onda ih plaćaju po korištenju. Domenu aplikacija mogu sami odrediti ili se prijaviti za Googleovu domenu. Nakon toga Google preuzima svu brigu o radu aplikacije. Njihova arhitektura dodjeljuje aplikacijama resurse, brine se za prosljeđivanje zahtjeva prema njima i daje im određeno vrijeme za sastavljanje odgovora. Pritom Google nudi dodatne usluge poput memorijskog prostora koje korisničke aplikacije mogu koristiti pri radu.

2.2.2. Amazon Elastic Compute Cloud (EC2)

EC2 je primjer oblaka koji pruža infrastrukturu kao uslugu. Dok u App Engineu korisnici postavljaju vlastite web-aplikacije u oblak, u EC2 korisnici plaćaju pristup virtualnim računalima koja se izvode u oblaku. Na primjer korisnik može iznajmiti virtualno računalo s operacijskim sustavom Windows i udaljeno se spajati na njega pomoću protokola poput SSH. Korisnik pritom ima opciju odrediti početne resurse za virtualno računalo te ih može mijenjati po potrebi. Osim toga EC2 može raditi s Amazonovim web-uslugama koje pružaju razne servise poput dodatnog memorijskog prostora i relacijskih baza podataka. Same instance virtualnih strojeva pokreću se preko API-a koji omogućuje i potpunu kontrolu nad pokrenutim instancama. Jedan način kontroliranja vlastitih resursa u Amazonovom oblaku je slanjem SOAP poruka putem HTTP protokola.



Slika 2.2: Tok korištenja EC2 [4]

Protokol SOAP definira format poruka koje se smještaju u zahtjeve HTTP-a ili RPC-a i omogućuje pozivanje metoda, prijenos argumenata i vraćanje rezultata. Temelji se na XML-u i zamišljen je za korištenje pri radu s web-uslugama. Kako je pristupna aplikacija za oblak često napravljena kao web-usluga, SOAP se često koristiti za komunikaciju s oblakom. Protokol je razmatran pri razvoju protokola opisanog u trećem poglavlju. Primjer poruka u formatu SOAP (preuzeto iz [5]):

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

2.2.3. Microsoft Windows Azure

Microsoft Windows Azure je skup komercijalnih usluga u oblaku koje Microsoft nudi korisnicima. Najpoznatija je po usluzi postavljanja korisničkih aplikacija u oblak. Taj dio isto spada pod model platforme kao usluge i funkcionira slično kao i Googleov App Engine. Korisnici pišu vlastite aplikacije u jednom od dostupnih jezika i postave ih u oblak. Aplikacije mogu biti pisane za ASP.NET, WFC, ali i u Javi, PHP-u i u Rubyu. Osim toga dostupna je usluga pohrane podataka u oblak, usluga koja pruža relacijsku bazu podataka temeljenu na Microsoftovom SQL Serveru i usluga koja omogućuje pokretanje virtualnih strojeva u oblaku. Time Azure podržava i model infrastrukture kao usluge, odnosno korisnici mogu u oblaku dobiti vlastiti virtualni stroj, na njemu pokrenuti određeni operacijski sustav i udaljeno mu pristupati.

3. Osmišljeni komunikacijski protokol

Glavni je cilj rada bio razviti protokol koji omogućuje postavljanje računala u oblak i posluživanje drugih računala preko Interneta. Najvažnije je bilo definirati dvije komponente protokola. Prva je format poruka koje se razmjenjuju između sudionika, a druga je način komunikacije. Zato je ovo poglavlje podijeljeno na dva dijela. Prvi dio opisuje sve poruke koje su definirane unutar protokola, a drugi opisuje način razmjena poruka između tri osnovna čvora u mreži. To su klijent koji pokušava dobiti obradu zahtjeva, posrednik koji prosljeđuje poruke između klijenata i poslužitelja i poslužitelj koji prima zahtjev i obrađuje ga. Najvažniji član je posrednik jer on kontrolira svu komunikaciju između klijenata i poslužitelja. U oblaku se nalaze posrednici i poslužitelji, a klijenti im pristupaju preko Interneta. Još jedan od ciljeva protokola bio je omogućiti lagano dodavanje novih čvorova i pametan način prosljeđivanja koji pokušava dobro iskoristiti postojeće resurse.

3.1. Poruke protokola

Za ovaj protokol definirane su tri osnovne skupine poruka, one za registraciju poslužitelja, za traženje usluge i za provjeru opterećenja. Svaka skupina definira bar dvije poruke, zahtjev i odgovor na njega. Sve se poruke šalju preko TCP-a (engl. transmission control protocol), protokola prijenosnog sloja u TCP/IP modelu. Poruke su većinom građene tako da je na početku poruke oznaka tipa, a iza toga slijedi XML koji sadrži sve potrebne podatke za njezinu obradu. Ako su potrebni podaci vrlo jednostavni (npr. potvrda primitka) onda se ona samo sastoji od tipa i takvog podatka, bez XML-a. To omogućuje brzu provjeru tipa poruke i dohvat podataka, a ako je potrebno prenijeti složenije podatke poput argumenata zahtjeva koristi se XML. On je prikladan zbog svoje fleksibilne strukture koja omogućuje lagano definiranje parametara za traženu uslugu i rezultata izvršavanja, ali i popisa usluga. Uz to je lagano dodati druge potrebne podatke poput oznaka i adresa koji se većinom stavljaju u poruku kao atributi XML-a. Iako protokol strogo definira format, dijelovi u poruci koji su vezani uz parametre i rezultate su slobodni i definiraju se po potrebi određenog zadatka, važno je samo održati zadanu strukturu oko tih dijelova. Tako je definiranje zadataka koji će se izvršavati u oblaku potpuno slobodno i neograničeno protokolom. Ako je zadatak presložen i rezultati se ne mogu vratiti u samo jednoj poruci onda prva poruka može jednostavno vratiti podatke pomoću kojih se klijent spaja na poslužitelja i dohvaća ostale rezultate.

3.1.1. Poruke za registraciju

Registrirati se može poslužitelj posredniku ili drugi posrednik onom koji je viši u hijerarhiji. To je napravljeno radi skalabilnosti sustava i laganog dodavanja čvorova u sustav koji je organiziran kao hijerarhija posrednika i poslužitelja. Registracija uključuje slanje IP adrese i priključne točke na kojima se sluša za raznim zahtjevima poput provjere opterećenja te slanje popisa svih pružanih usluga i priključnih točaka na kojima se te usluge mogu zatražiti. Ako zahtjev za registracijom šalje posrednik onda u poruku stavlja sve usluge koje sam pruža ali i one koje su registrirane kod njega. Tako viši posrednik ne vidi sve čvorove koji su registrirani kod nižeg posrednika, no vidi sve usluge koje može dobiti preko njega. Na zahtjev za registracijom odgovara se porukom prihvaćanja ili odbijanja. Tip tih poruka je REG. Osim njih sustav podržava poruku za micanje registracije koja je tipa DREG i poruku za promjenu podataka koja je tipa CREG. DREG sadrži oznaku poslužitelja čiji se podaci miču, a CREG istu XML poruku registracije kao i u običnoj REG poruci. Na DREG se ne šalje poruka odgovora, već se uspješni prijenos podataka smatra uspješnim micanjem registracije. Za poruku CREG šalje se poruka odgovora koja može biti odbijanje ili prihvaćanje promjene.

Format poruke REG:

Zahtjev:

"REG <?xml version="1.0"?> <Reg oznaka="oznaka_poslužitelja" ip="ip_poslužitelja"

port="port_poslužitelja"><Usluge><Usluga ime="ime_pružane_usluge"

port="port_pružane usluge"/>.....</Usluge></Reg>"

Odgovor:
"REG OK" za prihvaćanje zahtjeva ili "REG ODB" za odbijanje.

Format poruke CREG:
Zahtjev:
"CREG ista_xml_poruka_kao_u_reg"
Odgovor:
"CREG OK" za prihvaćanje, "CREG ODB" za odbijanje.

Format poruke DREG:
Zahtjev:
"DREG oznaka_poslužitelja"
Odgovor:
"DREG OK" za prihvaćanje, "DREG ODB" za odbijanje.

3.1.2. Poruke za traženje usluga

Zahtjev za uslugom klijent šalje posredniku. Njegov je tip REQ i moguća su tri odgovora. Prvi je odgovor tipa RES i sadrži odgovor na zahtjev (rezultate). Drugi je tipa SRV i sadrži IP adresu i priključnu točku drugog posrednika ili poslužitelja kojeg klijent mora kontaktirati s tim zahtjevom. Više o tijeku poruka i načinu rada u poglavlju 3.2. Zadnji je mogući odgovor tipa ERR i sadrži razlog zašto posrednik nije mogao dovršiti obradu zahtjeva. Klijent šalje posredniku svoju oznaku, naziv usluge koju želi dobiti i parametre potrebne za izvršavanje. Koji će odgovor primiti ovisi o konfiguraciji posrednika, osim ako dođe do greške kada dobiva poruku ERR. Kad posrednik prosljeđuje poruku klijenta samo šalje dalje istu REQ poruku i kao i klijent može dobiti samo jedan od tri navedena odgovora.

Format poruke REQ:

"REQ <?xml version="1.0"?><Klijent oznaka="oznaka_klijenta"><Usluga oznaka="ime_tražene_usluge"><Parametri>.....</Parametri></Usluga><Klijent>"

Format poruke RES:

"RES <?xml version="1.0"?><Rezultat>.....</Rezultat>

Format poruke SRV:

"SRV novi_ip novi_port"

Format poruke ERR:

"ERR opis_greške"

3.1.3. Poruke za provjeru opterećenja

Pri radu sustava posrednik mora provjeriti stanje svojih poslužitelja radi efikasnog prosljeđivanja zahtjeva. Zato su definirane poruke koje mu omogućuju dobivanje podataka o opterećenju od vlastitih poslužitelja ili posrednika. Poruka je tipa STATUS i sadrži ime usluge čija se opterećenost ispituje. Opterećenje se ispituje na razini usluge jer posrednik u pravilu ne zna komunicira li s drugim posrednikom ili poslužiteljem. Posrednik može samo vratiti vlastitu opterećenost ako je tako napravljeno ili može odrediti koju će opterećenost vratiti na temelju poslužitelja koji pružaju tu uslugu. Moguće je koristiti i drukčiju heuristiku koja uzima u obzir opterećenost posrednika i opterećenost poslužitelja s uslugom.

Format poruke STATUS:

Zahtjev:

"STATUS ime_usluge"

Odgovor:

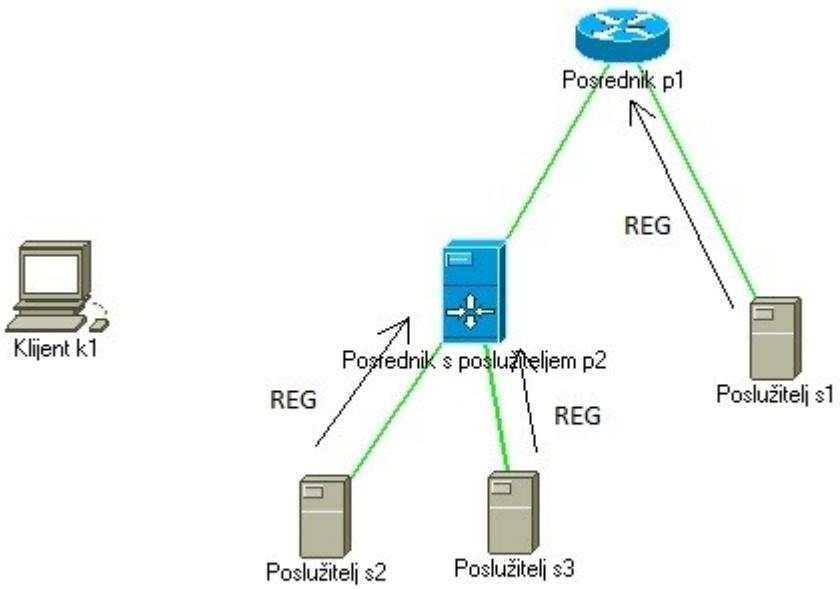
"STATUS razina_opterećenosti"

3.2. Rad protokola

Protokol se izvodi u dvije faze. Prva je faza registracije u kojoj se poslužitelji registriraju s nadležnim posrednicima, a posrednici s onim koji je viši u hijerarhiji. Druga je faza rada u kojoj posrednici prosljeđuju poruke zahtjeva od klijenata i po potrebi šalju poruke zahtjeva za statusom drugim čvorovima u mreži. U daljnjem tekstu navedeni su temeljni načini razmjene poruka u tim fazama. Sama implementacija ih se ne mora potpuno pridržavati već može koristiti i kombinacije sljedećih načina. Posrednici ne moraju biti podešeni na isti način, svaki posrednik se podešava zasebno.

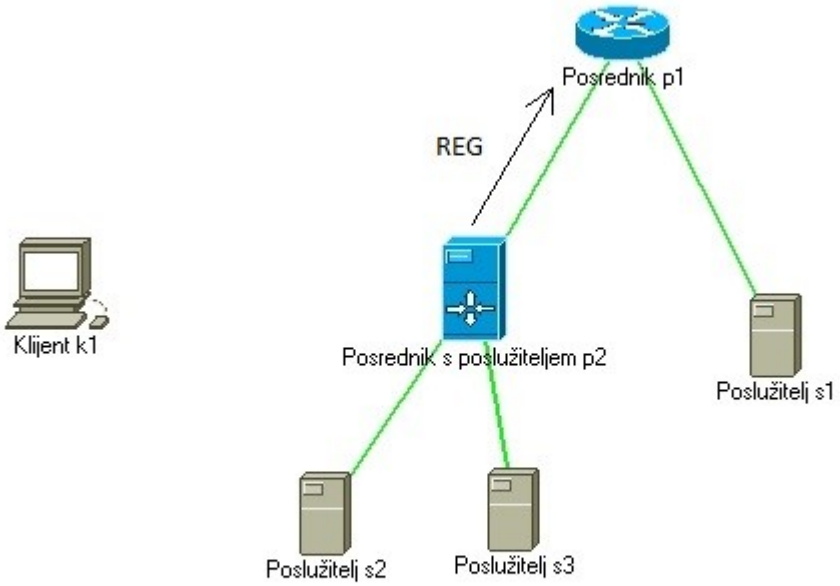
3.2.1. Faza registracije

Osim poslužitelja koji se naknadno jave posredniku, posrednici mogu i samo posluživati usluge. Njih popisuju pri pokretanju i uključuju ih u registracijsku poruku za višeg posrednika. Kad prime novu registracijsku poruku moraju ažurirati svoju listu i opet je poslati višem posredniku. To može biti poruka o registraciji, promjeni podataka ili micanju registracije. Protokol ne definira kad posrednik mora poslati nove podatke, odnosno hoće li odmah poslati ili počekati da se dogodi više promjena. U implementaciji pisanoj za ovaj rad napravljeno je da se novi popis šalje odmah. Poslužitelji odmah pri pokretanju šalju vlastite registracijske poruke i počinju slušanje za zahtjevima. Tijek poruka pri registraciji prikazuju slike 3.1. i 3.2.. Na prvoj se slici svi poslužitelji registriraju s nadležnim posrednicima. Nakon toga se posrednik p2, koji pruža i vlastitu uslugu, registrira s posrednikom p1. To je prikazano na slici 2. P2 će imati popisanu vlastitu uslugu i usluge poslužitelja s2 i s3. Tu listu će poslati p1 koji u listi ima samo usluge poslužitelja s1. Ako dođe do promjene u listi usluga kod s2 on će poslati novu listu p2, koji će zabilježiti promjene i proslijediti ih u novoj registracijskoj poruci do p1. P1 ne zna za poslužitelje s2 i s3, on samo vidi posrednika p2 i usluge koje mu je prijavio.



Slika 3.1: Poslužitelji šalju registracijske poruke nadležnim posrednicima.

Ako poslužitelj prekine s radom trebao bi poslati DREG poruku nadležnom posredniku. Ako to ne napravi neće doći do problema u radu, no zato će se nepotrebno gubiti vrijeme dok nadležni posrednik pomoću STATUS ili REQ poruka zaključi da je poslužitelj ispao iz mreže. Pri promjeni podataka poslužitelj mora poslati CREG poruku u kojoj je ista oznaka kao i ona koju je stavio u početnu REG poruku.



Slika 3.2: Posrednik p2 šalje p1 listu koja sadrži usluge p2, usluge s2 i usluge s3.

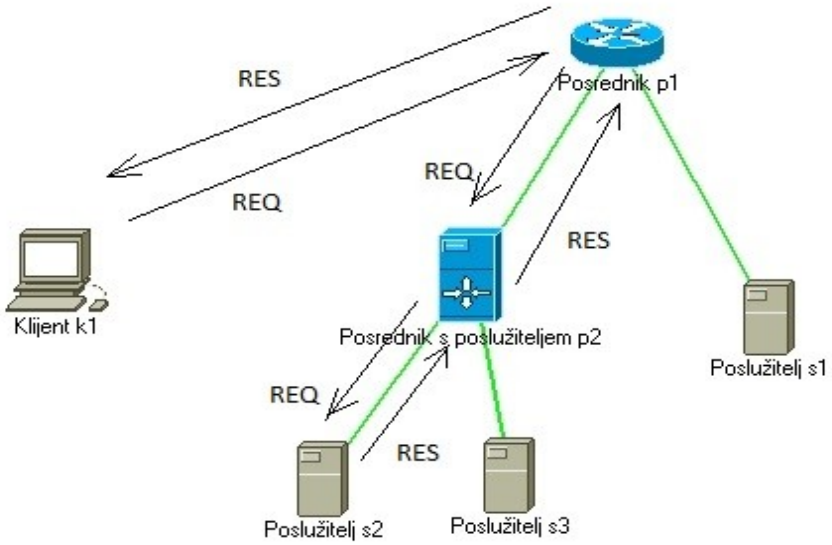
Posrednici bi trebali biti oblikovani tako da ako im nadležni čvor ispadne iz mreže oni sami mogu promijeniti postavke i nastaviti raditi kao vršni posrednici. Protokolom je definirano da posrednici moraju biti izgrađeni na način da rade ispravno neovisno o vlastitom položaju o hijerarhiji. Zato posrednik ne razlikuje druge posrednike koji su mu podređeni od poslužitelja i tako se lagano može dodati novi posrednik s novim poslužiteljima u hijerarhiju. Samo mu je potrebna adresa nadležnog. Posrednik u registracijskoj poruci stavlja sebe kao poslužitelja svake usluge koja je kod njega prijavljena. Tako nadležni posrednik ne zna broj čvorova u mreži ispod, on samo vidi jednog poslužitelja i usluge koje je naveo u registraciji. Time je poboljšana skalabilnosti sustava jer jedan posrednik koji je viši u hijerarhiji ne mora imati popis svih čvorova u mreži, samo mora znati kojom granom mreže proslijediti zahtjev za određenom uslugom.

Faza registracije je početna faza u radu protokola no ne završava prije početka faze rada. Zbog mogućih promjena u mreži i pružanim uslugama poruke registracije se razmjenjuju tijekom cijelog rada. Iznimka bi bila idealna mreža u kojoj su svi čvorovi stabilni i njihove usluge se ne mijenjaju pa nema potrebe za daljnjim registracijskim porukama. Protokol je sposoban prepoznati kad se dogodi promjena i tek se onda šalju novi registracijski podaci.

3.2.2. Faza rada

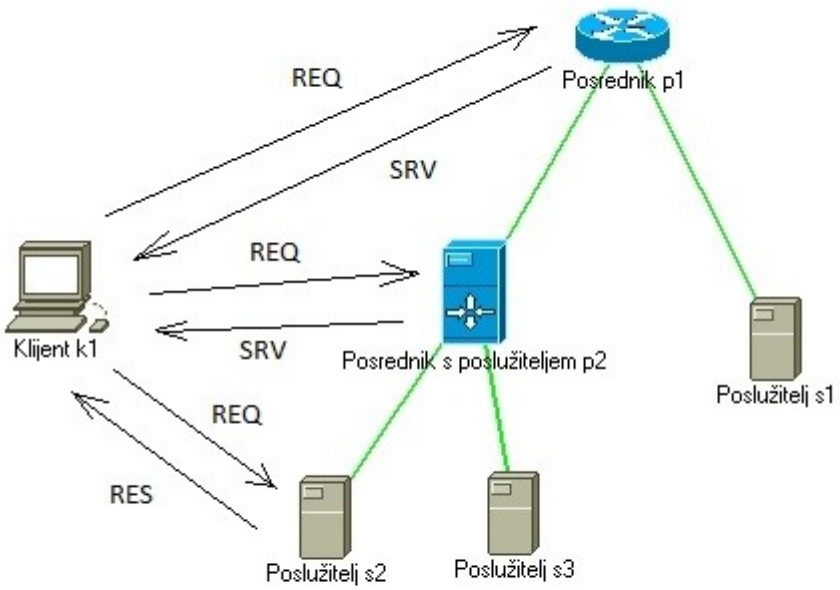
U ovom su protokolu najvažniji članovi posrednici. Oni tijekom rada primaju i stavljaju u mrežu nove poslužitelje i posrednike, ali su i odgovorni za prosljeđivanje zahtjeva klijenata. Iako to nije onemogućeno protokolom, preporuka je da sva komunikacija od klijenata ide prvo preko posrednika. Klijent može izravno kontaktirati poslužitelja s obzirom da je poruka REQ u svakom slučaju ista, ali to bi trebalo izbjegavati. Idealno bi bilo kad klijenti ne bi znali ništa o poslužiteljima. Razlog zašto protokol dopušta izravnu vezu između klijenata i poslužitelja jest zbog učinkovitosti. Ako je posrednik prenatrpan vezama ili na nedovoljno brzoj opremi, moguće ga je podesiti da umjesto posredovanja pri prijenosu zahtjeva samo obavještava klijenta na koju adresu mora poslati zahtjev. Sukladno s tim posrednik može raditi na dva načina koji su ilustrirani slikama 3.3. i 3.4..

Na slici 3.3. je prikazan način rada u kojem posrednik prosljeđuje klijentov zahtjev do odgovarajućeg poslužitelja i onda čeka odgovor koji šalje natrag klijentu. Ovdje klijent mora znati samo za tog posrednika i ne komunicira izravno ni s jednim drugim čvorom u mreži. Klijent k1 šalje poruku REQ posredniku p1. On u svojoj listi nalazi kome mora proslijediti tu poruku, odnosno tko sadrži zahtjevanu uslugu. U ovom je slučaju navedeno da ju sadrži posrednik p2 iako ju zapravo sadrži tek poslužitelj s2. P2 prosljeđuje dalje tu poruku do s2. On ju obrađuje i šalje rezultat u poruci RES do p2. P2 prosljeđuje poruku RES prema p1 koji ju onda šalje klijentu.



Slika 3.3: Posrednici prosljeđuju klijentovu poruku i čekaju natrag odgovor koji vraćaju višem čvoru, odnosno klijentu.

Na slici 3.4. posrednik ne prosljeđuje poruku već vraća klijentu adresu drugog posrednika ili poslužitelja kojeg mora kontaktirati. Dakle k1 šalje REQ p1. On mu vraća adresu p2 u poruci SRV. Postupak se ponavlja samo između k1 i p2. Sad k1 kontaktira s2 koji mu napokon obrađuje zahtjev i vraća rezultate u poruci RES. U ovom slučaju klijent izravno kontaktira sve čvorove u hijerarhiji do poslužitelja s uslugom, ali zato posrednici ne moraju čekati rezultate već samo vraćaju klijentu potrebnu adresu. Kako je moguća izravna veza između klijenta i poslužitelja u manje složenoj mreži moguće je potpuno izostaviti posrednike i pritom koristiti opisani protokol i programe pisane za njega.



Slika 3.4: Posrednici vraćaju klijentu adrese umjesto prosljeđivanja zahtjeva.

Protokolom je definirano kako posrednici mogu raditi, no odluka se ostavlja administratoru. Nije zamišljeno da se odluka radi za cijelu mrežu, već za svaki posrednik pojedinačno. Zato u mreži mogu biti posrednici koji prosljeđuju zahtjeve i oni koji samo vraćaju adrese. Ovisno o situaciji administrator može definirati da se samo vraćaju adrese. Time se smanjuje mogućnost da će oni postati uska grla i dobivaju se bolje performanse. No onda klijent koji je izvan mreže mora dobiti adrese svih unutarnjih čvorova na putu do usluge i povećava se broj poruka koje se šalju mrežom što može loše utjecati na skalabilnost mreže.

Zadnja vrsta komunikacije koja se odvija u mreži su provjere opterećenja. Protokol samo definira format poruka, no ne definira kad se točno moraju slati. Što se češće provjerava opterećenje posrednici će imati bolje informacije za prosljeđivanje zahtjeva, no

povećava se i broj poruka u mreži i vrijeme koje čvorovi troše na njihovu obradu. Najjednostavniji je način provjera svih prijavljenih poslužitelja kad posrednik primi zahtjev od klijenta. Problem je što se onda značajno produži vrijeme obrade zahtjeva i odjednom se šalje puno poruka u mrežu. Drugi način bi bio periodično provjeravati opterećenosti poslužitelja i bilježiti te podatke u bazu. Kad se primi zahtjev odabere se najprikladniji poslužitelj na temelju pohranjenih podataka. Treći način, koji je implementiran u ovom radu, jest provjeravati poslužitelje kad se primi poruka, no uz dodatne uvjete kojima se izbjegava slanje poruka svim poslužitelja. Detaljni je opis u trećem poglavlju, no svodi se na to da se traži prvi prihvatljivi poslužitelj prema definiranim parametrima, a ne onaj s najmanjim opterećenjem.

2.3. Karakteristike protokola

Najveća prednost prikazanog protokola jest njegova prilagodljivost. Korištenje XML-a za poruke omogućuje lagano definiranje zahtjeva i odgovora, a poruke su definirane tako da se mreža lagano može proširiti novim čvorovima. Poruke zahtjeva su iste neovisno o tome šalju li se posredniku ili poslužitelju, a posrednici se prijavljuju višim posrednicima istim porukama kao i drugi poslužitelji. Jedan posrednik tako samo zna za čvorove koji su izravno ispod njega, no ne zna jesu li poslužitelji ili drugi posrednici. Osim toga mora znati samo za jedan čvor iznad njega kojem će se prijaviti. To omogućuje lagano proširenje mreže u kojem novi čvor samo šalje registracijsku poruku preporučenom posredniku. Svi čvorovi iznad tog posrednika ne moraju znati za novi čvor, za njih je samo bitno da im on prijavi nove usluge ako su se pojavile. Klijenti mogu izravno kontaktirati poslužitelje ako imaju adresu, što omogućuje lagano definiranje načina rada u kojem posrednici samo vraćaju natrag potrebnu adresu. Protokol je vrlo fleksibilan i može se lagano prilagođavati ovisno o situaciji i potrebi. No nedostatak je što se ne može automatski prilagođavati prema potrebi, već se administratori moraju brinuti da su posrednici prikladno podešeni. Makar je moguće napraviti arhitekturu koja će koristiti opisani protokol i automatski namještati postavke ovisno o opterećenju i situaciji.

Protokol omogućuje skrivanje skoro svih čvorova od vanjskih uređaja. Za rad je potrebno znati adresu samo jednog posrednika iz mreže (vršnog) što je dobro za transparentnosti mreže. Struktura se može mijenjati po potrebi pri čemu klijent ne mora znati za promjene, ali ni za lokacije drugih čvorova. Iznimka je ako su posrednici podešeni da vraćaju adrese. No čak i tada promjene u mreži ne utječu na klijente koji i dalje moraju znati samo jednu adresu za početak rada.

Najveći je nedostatak protokola sigurnost, jer nije uopće definirana. Oslanja se na vanjske mehanizme koje će dizajner arhitekture implementirati da se ostvari odgovarajuća razina sigurnosti. Oni moraju osigurati metode autentifikacije i integritet i tajnost poruka. Autentifikacija se mora osigurati jer inače se bilo koji poslužitelj može prijaviti u mrežu ili preuzeti poruke za drugog poslužitelja ako se poruke ne označe tajnim ključem ili certifikatom. Važno je kontrolirati i identitete klijenata koji koriste usluge, zbog sigurnosti ali i zbog kontrole korištenja sustava. Integritet i tajnost se najbolje osiguravaju protokolom poput SSL-a. Sam protokol u porukama sadrži samo oznake čvorova koje je lagano mijenjati.

Osim sigurnosti protokol ne definira ni logiku iza provjere opterećenosti, već samo definira format poruke. Dizajner sam mora definirati heuristiku po kojoj određeni poslužitelj određuje broj koji predstavlja njegovu opterećenosti, i to mora uskladiti za cijelu mrežu. Ako nije dobro usklađeno sustav može lošije raditi jer opterećeniji poslužitelji javljaju manje brojeve od onih koji su manje opterećeni. Osim toga dizajner mora odlučiti i kada se šalju poruke opterećenosti, no to omogućuje bolju kontrolu nad performansama sustava.

4. Implementacija protokola

U ovom je poglavlju opisana implementacija osmišljenog protokola. Programi su pisani u jeziku C# u programskom okruženju Visual Studio 2010. Najvažniji je program posrednika koji sadržava osnovnu funkcionalnost protokola, a za demonstraciju rada napravljeni su i klijent, poslužitelj i program koji predstavlja uslugu posrednika. Poruke koje se razmjenjuju su u skladu s opisanim protokolom pa se neće dodatno objašnjavati, ovo poglavlje će obrađivati dijelove implementacije koju su neovisni o protokolu i mogu biti implementirani po potrebama dizajnera. Opisat će se i glavne metode i klase u implementaciji posrednika, poslužitelja i klijenta.

4.1. Posrednik

Posrednik ima pet osnovnih funkcija. Prva je konfiguracija u kojoj korisnik definira parametre koje pri pokretanju posrednik mora primijeniti i raditi prema njima. Druga je vođenje baze podataka o vlastitim uslugama i prijavljenim poslužiteljima. Treća je pokretanje i održavanje vlastitih usluga koje se pokreću kao vanjski procesi. Četvrta se odnosi na slušanje za zahtjevima poslužitelja i drugih posrednika, a peta na obradu zahtjeva klijenata. Obrada zahtjeva uključuje i provjeru opterećenosti odgovarajućih poslužitelja. Glavna klasa posrednika koja kontrolira sve ostale objekte je `PosaoPosrednika`. Ona sadrži sljedeće metode:

```
PosaoPosrednika{
      public PosaoPosrednika(TextBox txt); -konstruktor
      public void PrekiniRad();
      public void ZaustaviPosluzivanje();
      public void ZapocniRad();
      public void PrijavaPosredniku();
      private void primajPorukePosluzitelja();
      private void primajPorukeKlijenata();
      private void ucitajKonfiguraciju();
      private void pokreniSveUsluge();
      private void zaustaviSveUsluge();
      public void IzvrsiNaredbuList(string naredba);
      public void IzvrsiNaredbuProc(string naredba);
      public void AzurirajKonf(string parametar);
      public void IspisiKonf();
```

```
        private void ispisiSveAktivneUslugePosrednika();
        private void vratiStatusUsluge(string imeUsl);
        private void pokreniUslugu(string imeUsl);
        private void zaustaviUslugu(string imeUsl);
}
```

Metoda `ZapocniRad` prvo poziva `pokreniSveUsluge` koja pokreće sve procese s uslugama posrednika. Onda stvara dvije dretve, jedna izvodi metodu `primajPorukePosluzitelja`, a druga `primajPorukeKlijenata`. Obje slušaju za zahtjevima i stvaraju odgovarajuće objekte kad prime zahtjev. Metoda `ucitajKonfiguraciju` se poziva u konstruktoru klase i ona dohvaća objekt tipa `KonfObjekt` iz klase Konfiguracija. Metoda `zaustaviPosluzivanje` zaustavlja procese usluga pomoću `zaustaviSveUsluge` i postavlja varijablu `PrekiniPosluzivanje` pomoću koje signalizira dretvama da prekinu sa slušanjem. Metoda `PrekiniRad` poziva `zaustaviPosluzivanje` i onda prekida izvođenje posrednika nakon što joj dretve signaliziraju kraj rada. Ostale metode obrađuju korisnikove naredbe koje će biti detaljnije opisane kasnije. Npr. metoda `zaustaviUslugu` na korisnikov zahtjev prekida proces koji izvršava navedenu uslugu, a metoda `pokreniUslugu` ju pokreće. Sve se naredbe koje korisnik unese u grafičko sučelje samo djelomično obrađuju u klasi korisničkog sučelja, a onda se šalju u objekt klase `PosaoPosrednika` gdje se dovršava njihovo izvođenje.

4.1.1. Konfiguracija

Prije pokretanja posrednika, korisnik mora popuniti datoteku konf.txt s odgovarajućim parametrima i smjestiti je u isti direktorij s njim. Parametri se ne odnose samo na rad posrednika već i na način na koji će protokol raditi. Konfiguracijsku datoteku i konfiguraciju moguće je naknadno modificirati preko korisničkog sučelja posrednika s naredbom SET. Detaljnije o naredbama posrednika u poglavlju 4.1.5.

Parametri:

- naziv; ovaj parametar definira oznaku posrednika koju će koristiti u registraciji s nadležnim posrednikom.
- ip; internetska adresa koju će posrednik koristiti u svim vezama prijenosnog protokola ali i za vlastite usluge.
- port_klijent; priključna točka na kojoj će slušati za zahtjevima klijenata.
- port_posl; priključna točka na kojoj će slušati za porukama poslužitelja i ostalih posrednika.
- usl; ovim parametrom zadaje se ime izvršne datoteke koja pruža uslugu posrednika i priključna točka na kojoj će slušati za zahtjevima. U datoteci može biti više parametara usl ako je više izvršnih datoteka.

- time_delay; vrijeme nakon kojeg zapis o opterećenju nekog poslužitelja više ne vrijedi i potrebno je zatražiti novu provjeru.
- con_num; granica opterećenja ispod koje se poslužitelj odabire bez daljnjeg traženja onog s boljim opterećenjem.
- hijer; parametar označava je li posrednik na vrhu hijerarhije ili ima nadležnog posrednika kojemu se mora prijaviti.
- hijer_ip; internetska adresa nadležnog posrednika.
- hijer_port; priključna točka nadležnog posrednika.
- posred; parametar označava da li posrednik prosljeđuje zahtjeve klijenata ili im samo vraća odgovarajuće adrese.

U poglavlju 4.1.5 detaljno je opisano kako se gornji parametri definiraju i zadan je primjer konfiguracijske datoteke.

U programu je za konfiguraciju zadužena klasa Konfiguracija čiji se objekt inicijalizira pri pokretanju programa. Ona sadrži metode za čitanje i ažuriranje konfiguracijske datoteke. Osim toga one sastavljaju objekt tipa `KonfObjekt` koji sadrži sve konfiguracijske parametre i vraća se u glavnu metodu posrednika.

```
Konfiguracija{
      public Konfiguracija(TextBox txt);
      public void UcitajPodatke(out KonfObjekt konfObjekt);
      public KonfObjekt AzurirajParametar(string parametar);
      private void dodajUslugu(string vrijednost);
      private KonfObjekt sastaviKonfObjekt();
      private int formatirajPort(string vrijednost, string param);
      private IpAddress formatirajIP(string vrijednost);
      public void IspisiPodatke();
}
```

Iznad su navedene najvažnije metode klase Konfiguracija. `UcitajPodatke` čita konfiguracijski datoteku i stvara objekt tipa `KonfObjekt` koji vraća u pozivajuću metodu. `AzurirajParametar` prima ime i novu vrijednost parametra i to ažurira u konfiguracijskoj datoteci te sastavlja novi `KonfObjekt`. Metoda `IspisiPodatke` ispisuje u korisničko sučelje posrednika sve aktualne konfiguracijske podatke. Ostale se metode koriste za sastavljanje `KonfObjekta` iz datoteke i provjeru ispravnosti podataka. Klasa `KonfObjekt` nema metoda, već samo članove koji predstavljaju parametre konfiguracije.

4.1.2. Baza podataka i odabir poslužitelja

Posrednik ne koristi pravu bazu podataka već pamti poslužitelje i usluge u radnoj memoriji preko raznih lista i sličnih memorijskih struktura. One su sadržane u objektu klase `BazaPosrednika`.

```
BazaPosrednika{
članovi s bitnim podacima:
      List<Posluzitelj> prijavljeniPosluzitelji;
      List<Usluga> aktivneUslugePosrednika;
      Dictionary<string, List<int>> usluge;
      Dictionary<string, int> prijavljeniPoslIdMap;
      Dictionary<int, string> prijavljeniPoslImeMap;
važne metode:
      public bool DodajUsluguPosrednika(Usluga u);
      public void MakniUsluguPosrednika(string ime);
      public bool DodajPosluzitelja(Posluzitelj p);
      public void IzmjeniPosluzitelja(Posluzitelj p);
      public void MakniPosluzitelja(string oznakaPosl);
      public static bool ParsirajPodatkePosl(string xml, out
      Posluzitelj posluzitelj, out string error);
      public string SastaviRegPoruku();
      public bool OdaberiPosluzitelja(string imeUsl, out IPEndPoint
      veza, out string error, int idZahtjeva);
      private bool vratiOpterecenost(int idPosl, string imeUsl, out
      int opterec);
}
```

Najvažniji je član lista `prijavljeniPosluzitelji` koja sadrži objekte tipa `Posluzitelj`. Taj objekt sadrži sve potrebne podatke o poslužitelju poput njegove numeričke oznake, naziva, mrežnih adresa i popisa svih usluga koje pruža. Drugi je važan član rječnik usluga. U njemu su navedeni nazivi svih usluga koje su prijavljene ili sadržane na posredniku. Svakom nazivu je pridružena lista sa svim poslužiteljima koji ju pružaju. Ostali su članovi manje bitni i koriste se za druge metode poput one za ispis podataka o poslužiteljima ili dohvat aktivnih usluga posrednika. Za svaku uslugu na poslužitelju stvara se objekt tipa Usluga. On se može stvoriti za više istih usluga ako su na različitim poslužiteljima i sadrži njihovo ime, priključnu točku, podatak o opterećenosti i vrijeme kad je izvršena provjera opterećenosti. Ako se radi o usluzi posrednika onda sadrži i ime izvršne datoteke i objekt tipa Process koji se stvara prilikom pokretanja procesa s uslugom.

Većina navedenih metoda bavi se ažuriranjem baze, odnosno dodavanjem i micanjem usluga posrednika i poslužitelja. Metoda `OdaberiPosluzitelja` prima ime tražene usluge i iz baze odabire poslužitelja na sljedeći način. Ako je za traženu uslugu dostupan samo jedan onda odabire njega. Ako ih ima više onda nasumično izabere jedan od njih. Ako je izabranom vrijeme testiranja opterećenosti unutar dozvoljenog okvira ne traži novu provjeru već uzima već zapisanu. Tada gleda je li ona unutar prihvatljive granice opterećenosti i ako je onda odabire tog poslužitelja. Ako nije ide dalje na sljedećeg i ponavlja postupak. Ako ispadne da ni jedan nije unutar prihvatljive granice opterećenosti onda odabire onog koji ima najmanju opterećenost. Ovaj postupak pokušava ubrzati proces izbora tako da traži prvi prihvatljivi, a ne najbolji poslužitelj u mreži. Najbolji se traži samo ako nema prihvatljivih. Navedene se granice definiraju u konfiguraciji posrednika pomoću parametara con_num i time_delay koji su opisani u poglavlju 4.1.1. Za provjeru opterećenja zadužena je metoda `vratiOpterecenost` koja prima oznaku poslužitelja i ime tražene usluge. Na temelju tih podataka sastavlja poruku STATUS i šalje ju navedenom poslužitelju. Ako ne uspije ostvariti vezu i dobiti odgovor taj se poslužitelj briše iz svih lista. Mora se maknuti njegova oznaka iz lista poslužitelja za svaku uslugu, a onda se briše sam zapis o poslužitelju.

Metoda `ParsirajPodatkePosl` prima XML iz poruke registracije i sastavlja objekt klase `Posluzitelj`. U daljnjem tekstu slijedi opis klase `Posluzitelj` i popis njezinih članova kao i opis klase Usluga i popis njezinih članova.

```
Posluzitelj{
      static int brojPosluzitelja; broj koji se koristi za dodjelu
            oznake poslužitelju
      int idPosl; dodijeljena numerička oznaka
      string oznakaPosl;
      IPAddress ipPosl;
      int portPosl;
      List<Usluga> uslugePosl;
}
Usluga{
      string usluga;
      int port;
      int opterecenost;
      DateTime vrijemeProvjereOpt;
članovi rezervirani za usluge posrednika:
      string fileName;
      Process procesUsluge;
}
```

4.1.3. Usluge posrednika

U konfiguraciji se zadaju lokacije izvršnih datoteka koje pružaju uslugu posrednika. Za svaku se stvara objekt tipa Process pomoću kojih se pokreću. Korisnik može pomoću naredbe proc zaustavljati pojedine usluge i ponovno ih pokretati. Za ime usluge posrednika uzima se naziv izvršne datoteke bez nastavka .exe. Prekid rada procesa uvijek poziva delegat `prekidProcesa` koji se nalazi u klasi `PosaoPosrednika`. U delegatu se provjerava zašto je došlo do prekida i ako nije izazvan od strane korisnika ispisuje u grafičko sučelje razlog prekida. Osim toga miče i uslugu posrednika koju je izvršavao prekinuti proces iz liste aktivnih usluga.

Usluga posrednika napravljena je kao obična izvršna datoteka bez grafičkog sučelja koja prima dva parametra, internetsku adresu i priključnu točku na kojima sluša za zahtjevima. Posrednik pri pokretanju procesa predaje mu vlastitu internetsku adresu i priključnu točku koji su za tu uslugu definirani u konfiguraciji. Podaci o procesima pamte se u objektima klase Usluga koji sadrže referencu na objekt tipa Process. Kad je potrebno manipulirati određenom uslugom posrednik traži u listi odgovarajuću uslugu i njezinu referencu na proces, a pomoću te reference može ju zaustaviti, pokrenuti ili jednostavno provjeriti njezino stanje.

Primjer je jedne usluge posrednika obična usluga koja sortira brojeve po metodi *quicksort*. Klasa koja definira njezinu funkcionalnost jest `UslugaSortiranja` koja sadrži sljedeće metode:

```
UslugaSortiranja{
      public void PocniRad(IPAddress ip, int port);
      private void obradiZahtjev(objekt oKlijent);
      private bool parsirajZahtjev(string xml, out int[] brojevi,
      out string error);
      private string vratiImeProcesa();
      private int[] quicksort(int[] niz);
}
```

Metoda `PocniRad` prima internetsku adresu i vrata koji su zadani kao ulazni parametri procesa. Tada započinje slušanje za zahtjevima i svaki primljeni zahtjev dodjeljuje jednoj dretvi. Obrada zahtjeva izvršava se u metodi `obradiZahtjev` koja prima priključnicu (engl. *socket*) s klijentom preko koje prima poruku REQ. Metoda `parsirajZahtjev` provjerava ispravnost poruke i dohvaća brojeve koje treba sortirati. Brojevi se sortiraju u rekurzivnoj metodi quicksort i lista sortiranih brojeva se pakira u poruku odgovora i šalje natrag klijentu. Pomoću metode `vratiImeProcesa` dohvaća se ime usluge koje se koristi u provjeri ispravnosti poruke, odnosno je li zatražena odgovarajuća usluga. Time se omogućuje promjena imena usluge običnom promjenom imena procesa.

4.1.4. Obrada zahtjeva

Primanje zahtjeva započinje u klasi `PosaoPosrednika` u metodama `primajPorukeKlijenata` i `primajPorukePosluzitelja`. Te metode započinju slušanje na internetskoj adresi posrednika, no moraju imati različite priključne točke. Kad prime zahtjev svaka stvara odgovarajući objekt za obradu zahtjeva. Ako je zahtjev od klijenta stvara se objekt klase `ZahtjevKlijenta`, a za poslužitelja objekt klase `ZahtjevPosluzitelja`. Te klase imaju sljedeće članove i metode:

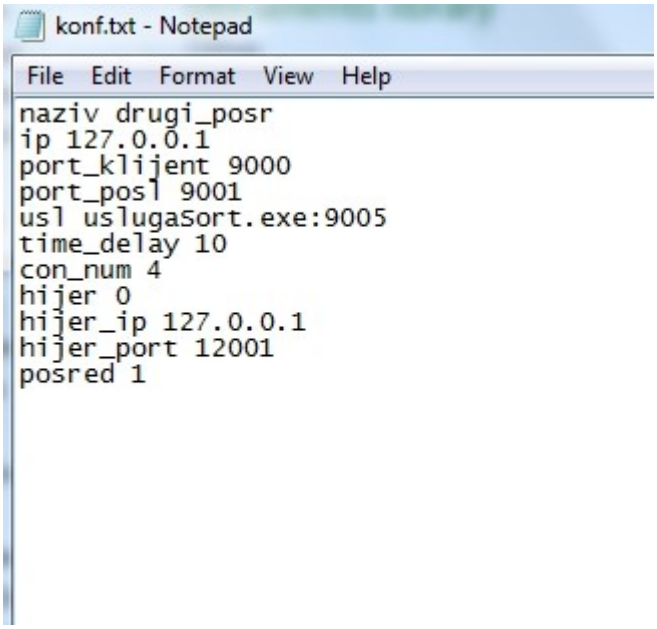
```
ZahtjevKlijenta{
članovi:
      public TcpClijent klijent;
      public BazaPosrednika baza;
      public KonfObjekt konfPosrednika;
      static int brojZahtjeva;
metode:
      public void ObradiZahtjev(object state);
      private IPEndPoint parsirajSrvPoruku(string odgovorPosl);
      private string sastaviSrvPoruku(IPEndPoint veza);
      private bool parsirajZahtjev(string dolPoruka, out string
      imeTrazeneUsluge,out string error);
}

ZahtjevPosluzitelja{
članovi:
      public TcpClijent klijent;
      public BazaPosrednika baza;
      public KonfObjekt konfPosrednika;
      public PosaoPosrednika posrednik;
      static int brojZahtjeva;
metode:
      public void ObradiZahtjev(object state);
      private bool izmjeniPosluzitelja(string dolXmlPoruka, out
      string error);
      private void odjaviPosluzitelja(string dolPoruka);
      private bool dodajPosluzitelja(string dolXmlPoruka, out
      string error);
}
```

Za oba zahtjeva pripadajućim se objektima mora predati priključnica s vezom, konfiguracijski objekt i objekt s bazom poslužitelja. Za zahtjev poslužitelja predaje se objektu i instanca klase `PosaoPosrednika`. Tako se može pozvati njezina metoda `PrijavaPosredniku` pomoću koje se nadležnom posredniku pošalju novi podaci o uslugama ako je došlo do promjena. To se radi samo ako nadležni posrednik postoji.

Metoda `ObradiZahtjev` u obje klase izvodi prikladnu logiku za obradu zahtjeva. Ako se radi o zahtjevu poslužitelja onda se ili registrira novi poslužitelj ili modificira ili briše postojeći. Za zahtjev klijenta traži se uvijek prikladni poslužitelj ako je zahtjev ispravan. Ovisno o konfiguraciji ili će se klijentu vratiti adresa pronađenog poslužitelja ili će ga posrednik sam kontaktirati i proslijediti mu zahtjev. Kad primi odgovor njega šalje natrag klijentu. Pritom mora biti sposoban obraditi SRV poruke i kontaktirati druge poslužitelje ako ne primi odmah odgovor za klijenta. Mora i sam moći sastaviti SRV poruku ako je konfiguriran da ne prosljeđuje zahtjeve.

4.1.5. Korištenje programa posrednika

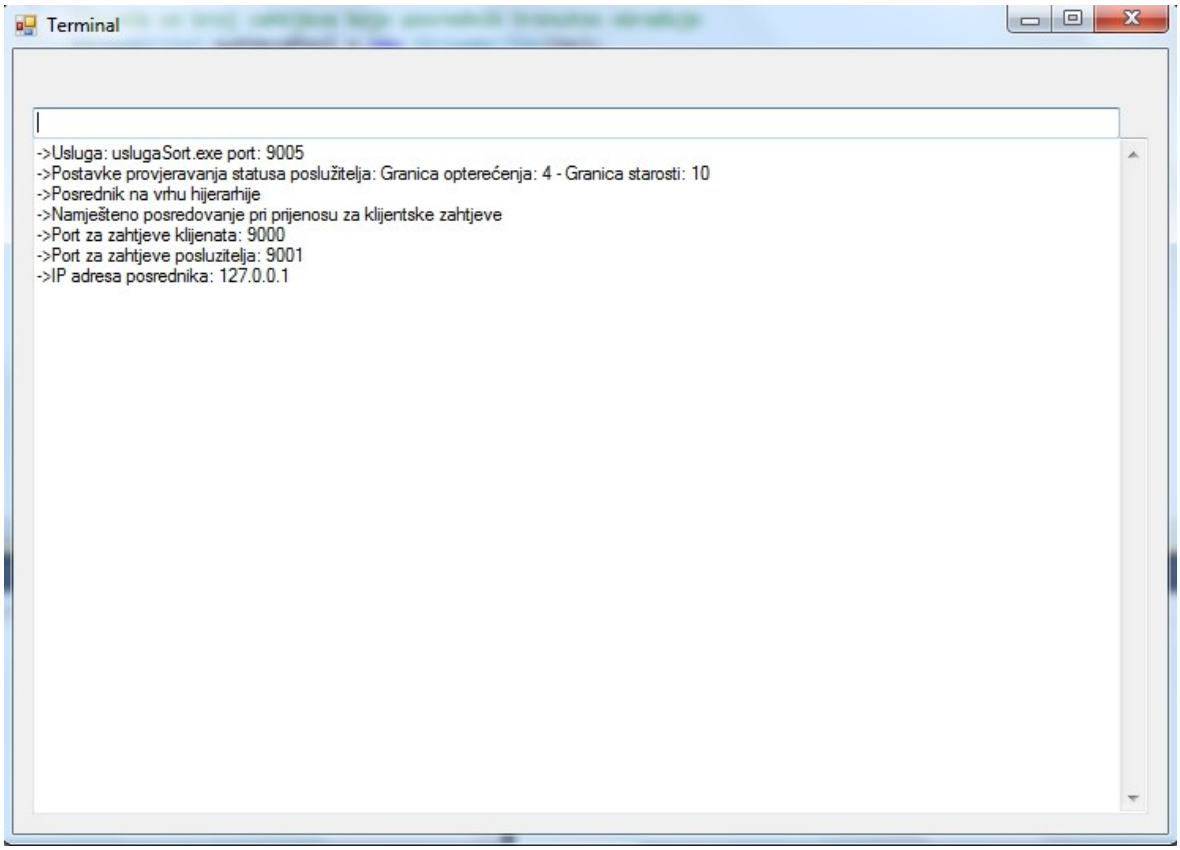
Prije pokretanja posrednika mora se definirati konfiguracijska datoteka konf.txt koja se smješta u isti direktorij s posrednikom. Parametri su objašnjeni u poglavlju 3.1.1., a ovdje je opisano korištenje datoteke. U svakom redu datoteke smije biti samo jedan parametar. Format retka je "ime_parametra vrijednost". Primjer ispravnog je retka "naziv prviPosrednik" kojim se definira da je oznaka posrednika "prviPosrednik". Primjer ispravne datoteke:



Slika 4.1: Primjer ispravne konfiguracijske datoteke.

Ispravan format parametra usl je "usl ime_izvršne_datoteke:priključna_točka". Kod parametra hijer 0 označava da nema nadležnog posrednika, a 1 da ima. Za parametar

posred 1 označava da posreduje pri obradi zahtjeva, a 0 da vraća adrese. Posrednik pri pokretanju provjerava ispravnost datoteke i jesu li svi potrebni podaci definirani. Ako je na vrhu hijerarhije onda nije potrebno definirati ip i vrata nadležnog posrednika. Nakon pokretanja sve podatke ispisuje u korisničko sučelje.



Slika 4.2: Izgled korisničkog sučelja posrednika.

Korisničko sučelje sastoji se od dva dijela, dio za unos naredbi na vrhu i dio za ispis poruka o izvođenju. Ispis je smješten odmah ispod reda u koji korisnici unose naredbe. Naredbe koje se mogu unositi su:

start – pokreće usluge posrednika i dretve koje slušaju za zahtjevima,

stop – zaustavlja usluge posrednika i slušanje za zahtjevima,

exit – isto zaustavlja sve usluge, a onda zatvara program,

list – ispisuje nazive i mrežne podatke svih prijavljenih poslužitelja,

list -i ime – ispisuje podatke o poslužitelju zadanog imena i popis usluga koje pruža,

list -u – ispisuje popis poslužitelja za svaku uslugu prijavljenu na posredniku,

proc -k – zaustavlja sve procese koji izvode usluge posrednika,

proc -k ime – zaustavlja proces koji izvodi zadanu uslugu,

proc -s – pokreće sve procese koji se ne izvode,

proc -s ime – pokreće proces navedene usluge,

proc -i ime – vraća stanje procesa navedene usluge,

set – ispisuje trenutnu konfiguraciju porednika,

set ime vrijednost – mijenja parametar sa zadanim imenom na zadanu vrijednost.

Mijenjanje parametara konfiguracije nije dopušteno dok posrednik sluša za zahtjevima, a mijenjanje stanja procesa s uslugama dozvoljeno je samo za vrijeme rada posrednika. Naredba list isto se može samo koristiti dok posrednik poslužuje zahtjeve.

U slučaju pojave greške pri pokretanju programa, greška će biti opisana u prozoru greške i izvođenje će biti prekinuto. To su obične greške vezane uz konfiguraciju. Sve se ostale greške za vrijeme rada ispisuju u korisničko sučelje i rad se ne prekida. To su obično greške pri zaprimanju i obradi zahtjeva.

4.2. Poslužitelj

Sve komponente poslužitelja ostvarene su u klasi Poslužitelj koja sadrži sljedeće članove i metode:

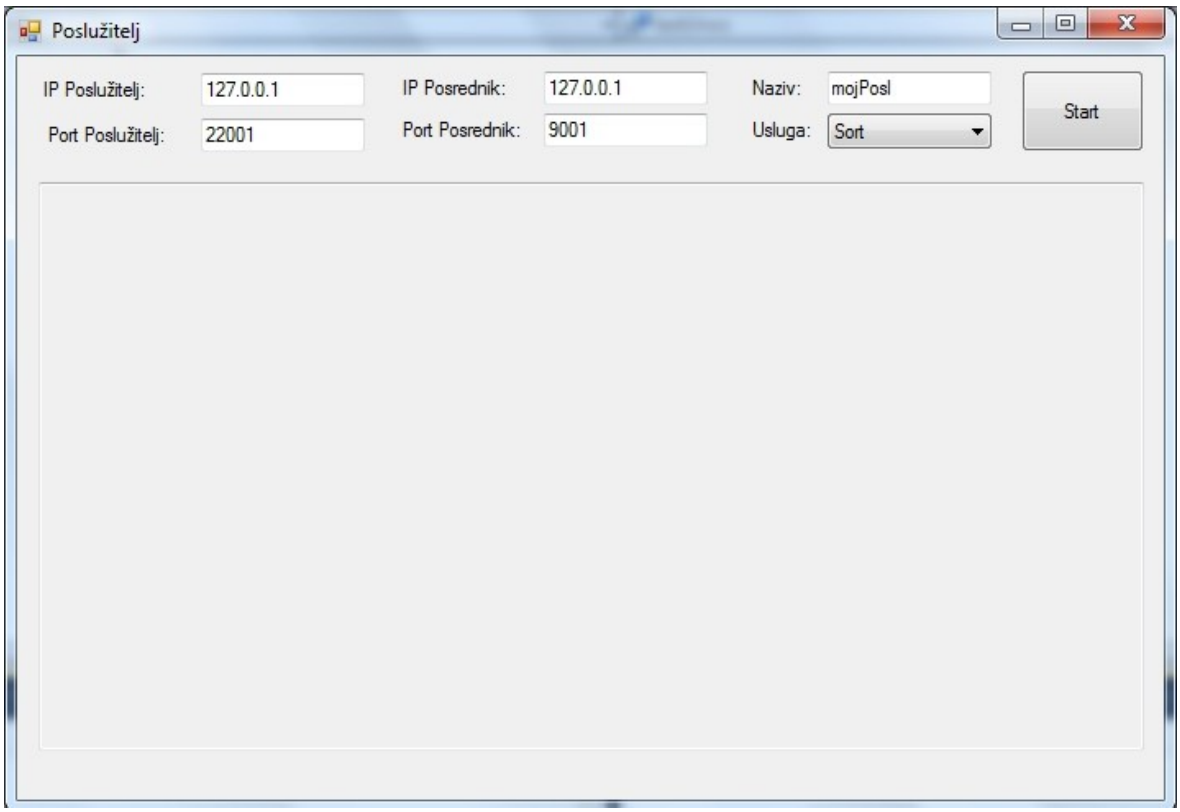
```
Posluzitelj{
članovi:
      IPAddress ipPosluzitelj;
      int portPosluzitelj;
      IPAddress ipPosrednik;
      int portPosrednik;
      string odabranaUsluga;
      string naziv;
metode:
      public void ZapocniPosluzivanje();
      public void PrekiniPosluzivanje();
      private void primajZahtjeve();
```

```
        private void obradiZahtjev(object oKlijent);
        private bool izvediUslugu(string xmlPoruka, out string
        odgovor, out string error, int brojZahtjev);
        private bool registrirajUsluge();
        private string napraviRegPoruku();
}
```

Član `odabranaUsluga` sadrži ime usluge koju je korisnik definirao pri pokretanju poslužitelja. Na temelju vrijednosti te varijable sastaviti će registracijsku poruku i odabrati koju metodu pozvati pri obradi zahtjeva. Metoda `ZapocniPosluzivanje` sastavlja i šalje zahtjev pomoću metode `registrirajUsluge` i pokreće dretvu koja izvodi metodu `primajZahtjeve`. Kad poslužitelj primi zahtjev on se dodjeljuje slobodnoj dretvi koja izvršava obradu. Prima poruku zahtjeva, predaje ju metodi `izvediUslugu` i odgovor šalje natrag klijentu. Ako je primljena poruka STATUS onda ne poziva tu metodu već samo vraća broj trenutno aktivnih posluživanja. Metoda `izvediUslugu` provjerava je li zahtjev ispravan i šalje podatke na obradu odgovarajućoj metodi. Npr. ako je zatražena usluga sortiranja poslati će brojeve koje treba sortirati metodi koja će ih sortirati i vratiti natrag.

4.2.1. Korištenje programa poslužitelja

Poslužitelj pri pokretanju prikazuje grafičko korisničko sučelje. U njemu korisnik mora ispuniti sva polja s podacima i pritisnuti tipku *Start* za početak rada. Osim toga može i prekinuti posluživanje i izmijeniti podatke. Izmjena podataka nije dopuštena za vrijeme posluživanja, već se poslužitelj prvo mora zaustaviti. To se radi istom tipkom kojom se i pokreće. Tada se može promjeniti i usluga koju će pružati.



Slika 4.3: Grafičko sučelje poslužitelja.

Sučelje se sastoji od dijela za ispis poruka sustava i dijela za konfiguraciju. Korisnik pomoću tipke *Start* pokreće posluživanje, no poslužitelj će se pokrenuti samo ako su uneseni svi potrebni podaci. Moraju biti unesene mrežna adresa posrednika s kojim će se registrirati, mrežna adresa na kojoj će slušati za zahtjevima, oznaka poslužitelja i iz padajućeg izbornika potrebno je odabrati uslugu koju će posluživati. Time je olakšana demonstracija rada jer se može pokrenuti jedan program u više instanci i svaka instanca može posluživati drugu uslugu. Poslužitelj se neće pokrenuti ako nisu svi podaci uneseni, već će javljati grešku. Ako se ne uspije registrirati s posrednikom javit će grešku i prekinuti s radom. Osim toga sve ostale greške ispisuju se u korisničko sučelje i izbjegava se prekid rada.

4.3. Klijent

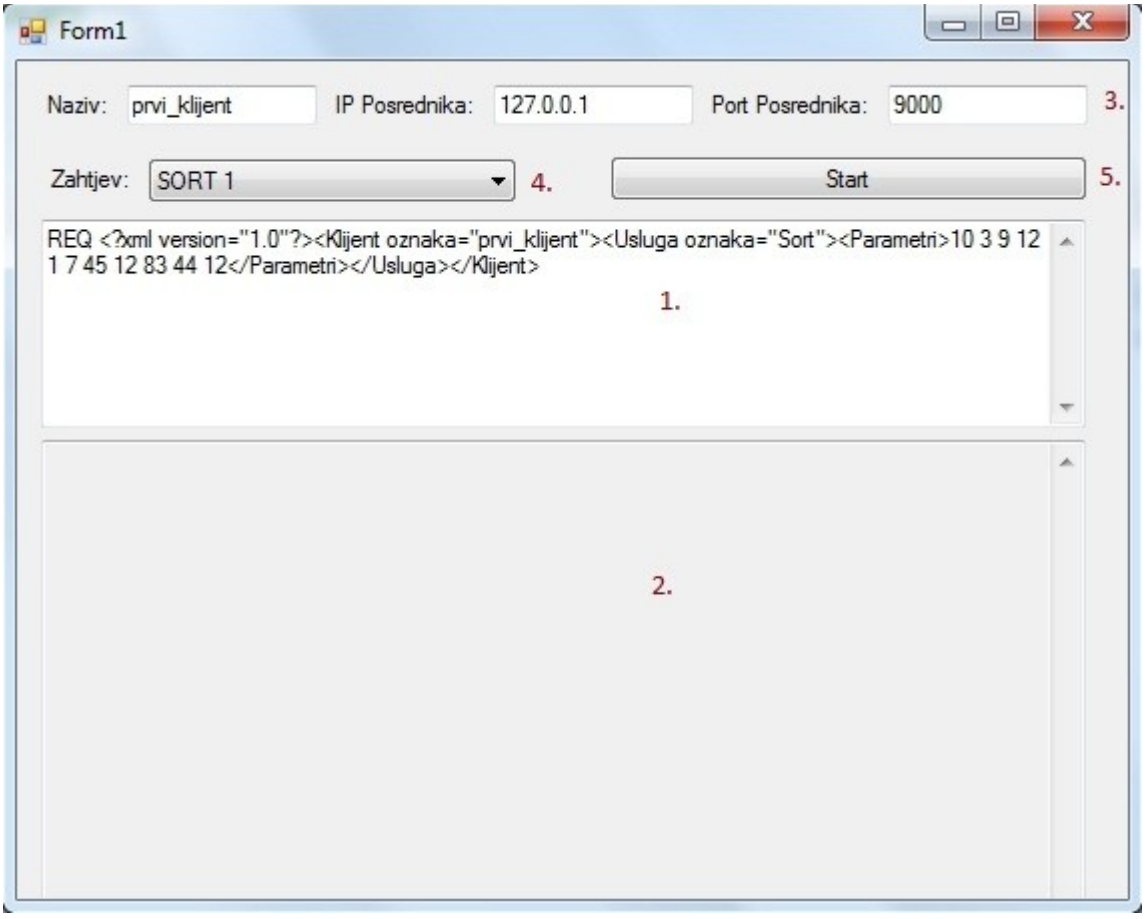
Glavna funkcionalnost klijenta ostvarena je u klasi Klijent koja se sastoji od sljedećih članova i metoda:

```
Klijent{
članovi:
      IPAddress ipPosrednika;
      int portPosrednika;
      string zahtjev;
metode:
      public bool PosaljiZahtjev();
      private void slanjeZahtjeva();
      private bool azurirajPodatkeSpajanja(string podaciOdgovora);
      private void ispisiRezultat(string podaciOdgovora);
      private bool validirajZahtjev(string zahtjev);
}
```

Za rad je klijentu potrebna mrežna adresa posrednika i poruka zahtjeva. Te podatke korisnik unosi u grafičko sučelje pri pokretanju klijenta. Rad započinje metoda `PosaljiZahtjev` koja prvo provjerava ispravnost zahtjeva pomoću metode `validirajZahtjev`, a onda stvara dretvu u kojoj se izvodi metoda `slanjeZahtjeva` koja ga šalje. Metoda `validirajZahtjev` provjerava strukturu XML-a, no ne pomoću datoteke XLS, već jednostavnim parsiranjem i provjerom sadrži li potrebne elemente. Nakon slanja klijent čeka jedan od tri moguća odgovora. Ako je odgovor poruka RES, rezultati se ispisuju u korisničko sučelje u metodi `ispisiRezultat`. Za poruku SRV poziva se metoda `azurirajPodatkeSpajanja` koja mijenja mrežnu adresu na koju klijent šalje zahtjev i ponavlja se postupak slanja. Ako je vraćena poruka greške ERR, onda samo ispisuje opis greške sadržan u poruci.

4.3.1. Korištenje programa klijenta

Klijent pri pokretanju prikazuje sljedeće grafičko sučelje:



Slika 4.4: Izgled grafičkog sučelja klijenta.

Korisnik prije slanja zahtjeva mora ispuniti sve tražene podatke u dijelu 3.. To su naziv klijenta koji će se koristiti kao oznaka te mrežna adresa posrednika kojem se zahtjev šalje. Nakon toga korisnik može ili odabrati jedan od pripremljenih zahtjeva iz padajućeg izbornika pod 4. ili može napisati vlastiti zahtjev u dijelu 1.. Moguće je i jednostavno izmijeniti jedan od već pripremljenih. Prije sastavljanja zahtjeva mora biti definiran naziv jer se koristi pri sastavljanju poruke. Klikom na *Start* (5.) počinje slanje zahtjeva. Svi se podaci ispisuju u dio 1., a to mogu biti ili poruke odgovora na zahtjev ili poruke o greškama za vrijeme rada.

5. Zaključak

Računalni će oblaci postati sve popularniji u budućnosti, no ne samo u poslovnom svijetu nego i za obične korisnike. Danas sve više vidimo prijelaz na modele koji uključuju pristup uslugama i sadržajima samo preko Interneta. Korisnici sve više posla obavljaju preko web-preglednika, od provjere elektroničke pošte, pisanja dokumenata do gledanja filmova i slušanja glazbe. Pritom se izbjegavaju instalacije programa lokalno ali i sadržaj se često samo prikazuje u toku podataka i ne ostaje trajno zapisan. Kako korisnici sve više prihvaćaju pristup svemu preko Interneta, ideja oblaka na koje poslovni korisnici stavljaju svoje aplikacije sve je privlačnija. Umjesto kupovanja vlastite opreme mogu pružati razne usluge uz relativno nisku cijenu, a u današnjem svijetu raznim je tvrtkama sve važnije imati dio poslovanja na Internetu. Moguće je da će u budućnosti korisnici većinom preći na operacijske sustave koji uopće ne instaliravaju aplikacije lokalno, već svemu pristupaju preko Interneta. Primjer takvog operacijskog sustava je Googleov Chrome OS koji je za vrijeme pisanja ovog teksta još uvijek u razvoju. U takvoj će budućnosti tehnologije i modeli poslovanja vezani uz oblak postati nezaobilazni većini tvrtki. Postoje mnogi negativni aspekti vezani uz to, od sigurnosti do problema vlasništva, koji će se pritom morati riješiti.

U ovom je radu cilj bio bolje se upoznati s pojmom računarstva u oblacima osmišljavanjem jednostavnog komunikacijskog protokola i njegove implementacije. Rezultat je vrlo jednostavan računalni oblak koji može pružati razne usluge, a pritom pokušava biti fleksibilan u izvedbi i omogućiti jednostavna proširenja mreže i pružanih usluga. Ozbiljna implementacija oblaka danas mora uzeti u obzir skalabilnost i upravljanje resursima, najčešće metodom virtualizacije koja u ovom radu nije korištena. Ovdje je naglasak bio na komunikaciji i upravljanju mrežnim porukama, dok nedostaje dio za optimizaciju korištenja procesorske moći. Ipak i ovakvom implementacijom se dobiva dojam o problemima i značajkama koje su važne pri izgradnji računalnog oblaka.

Literatura

1. Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia, travanj 2010., *A View of Cloud Computing,* http://cacm.acm.org/magazines/2010/4/81493-a-view-of-cloud-computing/fulltext, 29.5.2011.
2. Geva Perry, 28.9.2008., *Ellison's Anti-Cloud Computing Rant*, http://gevaperry.typepad.com/main/2008/09/larry-ellisons-anti-cloud-computing-rant.html, 29.5.2011.
3. http://dictionary.reference.com/browse/cloud+computing, 29.5.2011.
4. *Amazon Elastic Compute Cloud Getting Started Guide,* http://docs.amazonwebservices.com/AWSEC2/latest/GettingStartedGuide/, 30.5.2011.
5. *SOAP*, http://en.wikipedia.org/wiki/SOAP, 30.5.2011.
6. *Google App Engine FAQs*, http://code.google.com/appengine/kb/, 30.5.2011.
7. *Windows Azure Features*, http://www.microsoft.com/windowsazure/features/, 31.5.2011.
8. *MSDN Library*, http://msdn.microsoft.com/en-us/library/ms123401.aspx

Sažetak

Naslov: Osmišljavanje računalnog oblaka

Ovaj je rad podijeljen na tri dijela. Prvi dio opisuje model računarstva u oblacima, njegove prednosti i nedostatke. Dodatno je ukratko opisano nekoliko popularnih komercijalnih rješenja koja nude usluge po tom modelu. Drugi dio rada opisuje protokol koji je osmišljen za ovaj rad. On prikazuje mogući način komunikacije u oblaku i potrebne poruke za ostvarenje rada protokola. Treći dio rada prikazuje implementaciju temeljenu na tom protokolu koja ostvaruje jednostavan računalni oblak. Ona se sastoji od klijenta koji šalje zahtjeve, poslužitelja koji pruža usluge i posrednika koji prosljeđuje poruke između klijenata i poslužitelja.

Ključne riječi: oblak, računarstvo, računarstvo u oblacima, računalni oblak, klijent, poslužitelj, posrednik, Windows Azure, Google App Engine, Amazon EC2, Amazon Elastic Cloud Computing, aplikacija kao usluga, platforma kao usluga, infrastruktura kao usluga.

Summary

Title: Developing cloud infrastructure

The first part of this document defines what is cloud computing and its advantages and disadvantages. There are also some examples of modern cloud services from providers like Google and Microsoft. The second part of this document describes a communication protocol that was designed for a simple cloud. It contains definitions of network messages and descriptions of a communication flow within the cloud. The third part describes an implementation of a cloud that uses the defined protocol. Parts of this implementation are a client that sends requests, a server that provides services and a proxy that forwards messages between them.

Key words: cloud, cloud computing, client, server, proxy, Google App Engine, Windows Azure, Amazon EC2, Amazon Elastic Cloud Computing, software as a service, platform as a service, infrastructure as a service.