SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 15

Baze podataka u sustavima s visokom dostupnošću

Hrvoje Kostić

Voditelj: *Doc.dr.sc. Leonardo Jelenković*

Zagreb, lipanj, 2010

Sadržaj

1. Uvod

Raspodijeljeni sustavi imaju veliku ulogu u modernom računarstvu. Veliki broj usluga dostupnih korisnicima se temelji na njima, od osnovnih poput pregledavanja vremenske prognoze do složenih usluga vrijednih nekoliko milijuna dolara. Tako velika uloga zahtijeva visoku stabilnost, sigurnost i dostupnost sustava.

U telekomunikacijskim, bankovnim, vojnim i sličnim sustavima postoji mnogo nestabilnih i kritičnih komponenata koje je potrebno zaštiti i osigurati od kvarova. Izvori kvarova mogu biti razni, poput zatajenja sklopovlja, programske podrške ili mrežne infrastrukture.

Zaštita se može provesti na više načina, a u ovom diplomskom radu će se detaljnije opisati mehanizmi za održavanje baze podataka u visoko dostupnom sustavu.

U drugom poglavlju se detaljnije upoznaje sa visoko dostupnim sustavima te osnovnim načelima njihove zaštite, trećem poglavlju će biti riječi o metodama održavanja sustava visoko dostupnim, dok će daljini dio rada biti usmjeren na praktično ostvarenje sustava.

2. Visoko dostupni sustavi

Visoko dostupni sustavi su sustavi koji posjeduju potrebne radnje za održavanje sustava dostupnim bez obzira na zatajenja. Zahtijevaju poseban način oblikovanja koji će detektirati zatajenje te pokušati oporaviti sustav od greške.

Postoje dva načina izgradnje visoko dostupnog sustava. Prvi je izgradnja cijelog sustava od početka, a drugi je korištenje visoko-dostupnog međusloja nad željenim sustavom.

Dostupnost se smatra mogućnosti korisnika da pristupe sustavu, dodaju nove stavke, izmjenjuju postojeće i slično. Ako korisnik ne može pristupiti sustavu znači da je sustav nedostupan.

Postoje dvije vrste nedostupnosti: planirana i neplanirana.

Planirana nedostupnost je najčešće rezultat održavanja sustava i vrlo često se ne može zaobići. Primjeri planirane nedostupnosti su postavljanja raznovrsnih zakrpi koje zahtijevaju ponovno pokretanje sustava i sl.

Neplaniranu nedostupnost uglavnom uzrokuje sklopovsko ili programsko zatajenje ili nepredviđeno ponašanje. Primjeri je zatajenje procesora, radne memorije, gubitak mreže, napajanja ili različite programske greške.

Ponekad se planirana nedostupnost ne razmatra u izračunima dostupnosti sustava. Time se stiče dojam prividno stalne ili jako visoke dostupnost. Sustavi koji zaista pružaju stvarno stalnu dostupnost su veoma rijetki i jako skupi. Oni prikrivaju svaku sklopovsku, programsku ili mrežnu grešku te omogućavaju nadogradnju aplikacija, zakrpa i izmjene tokom rada sustava.

Tablica 1. Vremena nedostupnosti

Dostupnost	Nedostupnost tokom godine	Nedostupnost tokom mjeseca	Nedostupnost tokom tjedna
90%	36.5 dana	72 sati	16.8 sati
95%	18.25 dana	36 sati	8.4 sati
98%	7.30 dana	14.4 sati	3.36 sati
99%	3.65 dana	7.20 sati	1.68 sati
99.5%	1.83 dana	3.60 sati	50.4 min
99.8%	17.52 sati	86.23 min	20.16 min
99.9% ("tri devetke")	8.76 sati	43.2 min	10.1 min
99.95%	4.38 sati	21.56 min	5.04 min
99.99% ("četiri devetke")	52.6 min	4.32 min	1.01 min
99.999% ("pet devetki")	5.26 min	25.9 s	6.05 s
99.9999% ("šest devetki")	31.5 s	2.59 s	0.605 s

Tokom izgradnje visoko dostupnog sustava svako dodavanje nove komponente može prouzročiti neželjene posljedice na dostupnost. Razlog tome je što složeniji sustavi imaju veću vjerojatnost kvara te ih je teže ispravno implementirati. Zbog navedenih razloga se nastoji držati dizajn sustava što jednostavnijim. Kako bi se poticao razvoj visoko dostupnih sustava i usluga, računalna i telekomunikacijska industrija je osnovala *Service Availability Forum* (SAF[1]) koji olakšava i bitno ubrzava razvoj takvih proizvoda.

3. Metode visoke dostupnosti

Visoka dostupnost baza podataka u sustavu se odnosi na mogućnost konzistentnog nastavka korištenja sustava nakon zatajenja jedne ili više baza podataka. U tom slučaju sustav koristi sigurnosnu kopiju koja sadrži sve podatke kao i glavna baza do trenutka zatajenja.

Metode koje održavanju baze podataka visoko dostupnim moraju osigurati mehanizme objave novih, izmjenu starih ili brisanje postojećih podataka u svakoj pomoćnoj bazi podataka sustava.

Tokom izgradnje sustava se javljaju mnogi problemi, od kojih je jedan od glavnih otkrivanje promjena ili unosa novih podataka u glavnu bazu podataka. Ostvarenje visoke dostupnosti moguće je na nekoliko načina. Svaki način ima svoje prednosti i nedostatke. U ovom poglavlju će biti riječi o mogućim ostvarenjima.

3.1 Sustav s vremenskim oznakama

Sustav s vremenskim oznakama dodjeljuje oznaku (zastavicu) za svaki novo uneseni, odnosno izmijenjeni podatak.

Postoje dva moguća načina izgradnje ovakvog sustava:

- dodavanje oznaka izmjene/unosa u tablice baze podataka i
- stvaranje novih tablica sa zastavicama o unosu/izmjeni podataka.

Prednosti sustava s vremenskim oznakama su:

- jednostavnost izgradnje,
- robusnost sustava i
- mogućnost indeksiranja zastavica radi bržeg pristupa novim podatcima.

Nedostatci su:

- problemi sa brisanjem podataka tj. ukoliko se neki podatak obriše, kako to označiti,
- razlikovanje operacija unosa i izmjene i
- negativan utjecaj na svojstva izvorišnog sustava.

3.2 Sustav s analizom logičkih dnevnika baze podataka

Sustav s analizom logičkih dnevnika baze podataka koristi logičke dnevnike baze podataka koji su nastali tokom rada sustava. U mogućnosti je pregledati sve zapise i izmjene te objaviti ih kopijama baza podataka u sustavu.

Prednosti navedenog sustava su:

- neutjecanje na rad izvorišnog sustava i
- svi događaji se mogu obnoviti.

Negativne strane su:

- ovisnost o proizvođaču baze podataka i
- složenost implementacije sustava.

3.3 Sustav s dnevnikom događaja

Sustav s dnevnikom događaja radi na sličnom načelu kao i sustavi s analizom logičkih dnevnika baze podataka. Razlika se očituje u tome što sustav sam radi svoj dnevnik događaja i neki vanjski proces bi mogao biti zadužen za objavljivanje novih događaja. Na taj način se smanjuje opterećenje izvorišnog sustava.

Dnevnici podataka mogu biti posebne tablice u koje se unose zapisi pomoću okidača baze podataka. Podatci koji se spremaju su: ključ zapisa, status, vremenska oznaka i ključ operacije koja se obavlja nad podatkom.

3.4 Sustav s centralnim poslužiteljem

Sustav s centralnim poslužiteljem se sastoji od glavnog poslužitelja na kojeg se spajaju svi korisnici i obavljaju upite nad bazom podataka. Centralni poslužitelj je odgovoran za objavu novih podataka i događaja u sustavu.

Prednosti ovakvog sustava su:

- jednostavnost izgradnje sustava,
- jednostavan način objave podataka,
- konzistentnost podataka u svakom trenutku i
- mogućnost umjeravanja opterećenja baza podatka.

Nedostatci su:

- centralni poslužitelj postaje usko grlo sustava te
- pri ispadu poslužitelja sustav postaje nedostupan (poslužitelj postaje točka ispada sustava).

3.5 Raspodijeljeni sustav

Raspodijeljeni sustav sadrži više ravnopravnih baza podataka koje se međusobno sinkroniziraju. Detaljniji opis raspodijeljenog sustava slijedi u nastavku rada.

4. Arhitektura sustava

Ostvareni sustav visoko dostupnih baza podataka je temeljen na raspodijeljenoj arhitekturi međusobno ravnopravnih čvorova.

Svaki čvor se sastoji od četiri međusobno povezana dijela:

- baze podataka,
- upravljača,
- sinkronizatora i
- obnavljača.

Ukoliko je korisnička operacija bila pohrana, unos ili izmjena, tada je potrebno obavijestiti ostale čvorove o nastaloj promjeni u sustavu. Čvor javlja ostalim čvorovima da je došlo do promjene stanja te im šalje odgovarajuće podatke koje trebaju ažurirati.

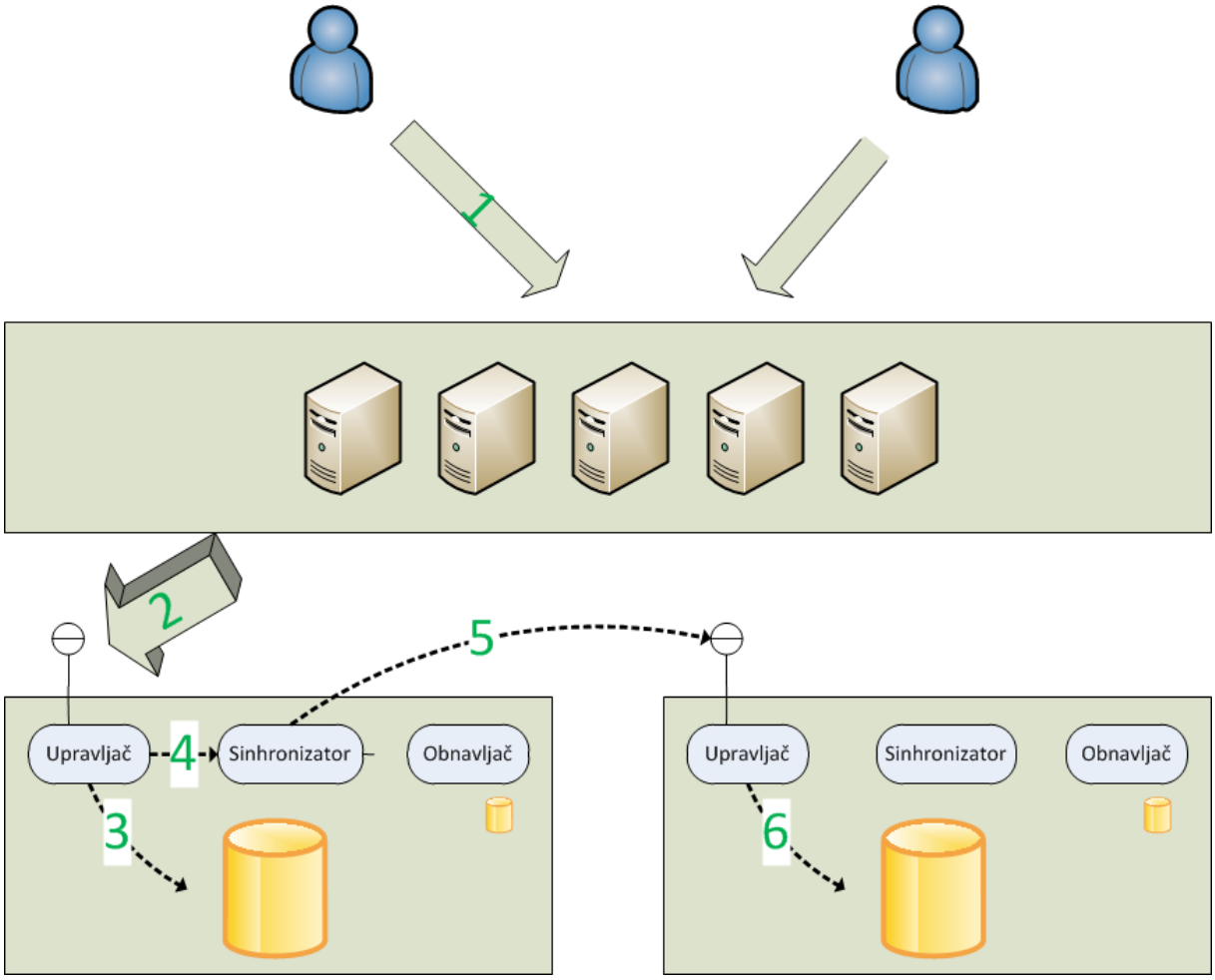
U slučaju da neki od čvorova trenutno nije dostupan, obnavljač sprema podatke u lokalnu bazu podataka i čeka da primi poruku o ponovnoj dostupnosti željnog čvora.

Problem koji se može javiti je kada zataji čvor koji sadrži neobjavljene podatke za čvor koji je postao dostupan. U tom slučaju se vrši međusobna usporedba i sinkronizacija svih podataka sa susjednim čvorom te se dohvaćaju podatci koji nedostaju.

5. Analiza rada sustava za sinkronizaciju baza podataka

U ovom poglavlju će biti obrađeni mogući slučajevi rada sustava za sinkronizaciju baza podataka.

5.1 Uspješno pohranjivanje podataka

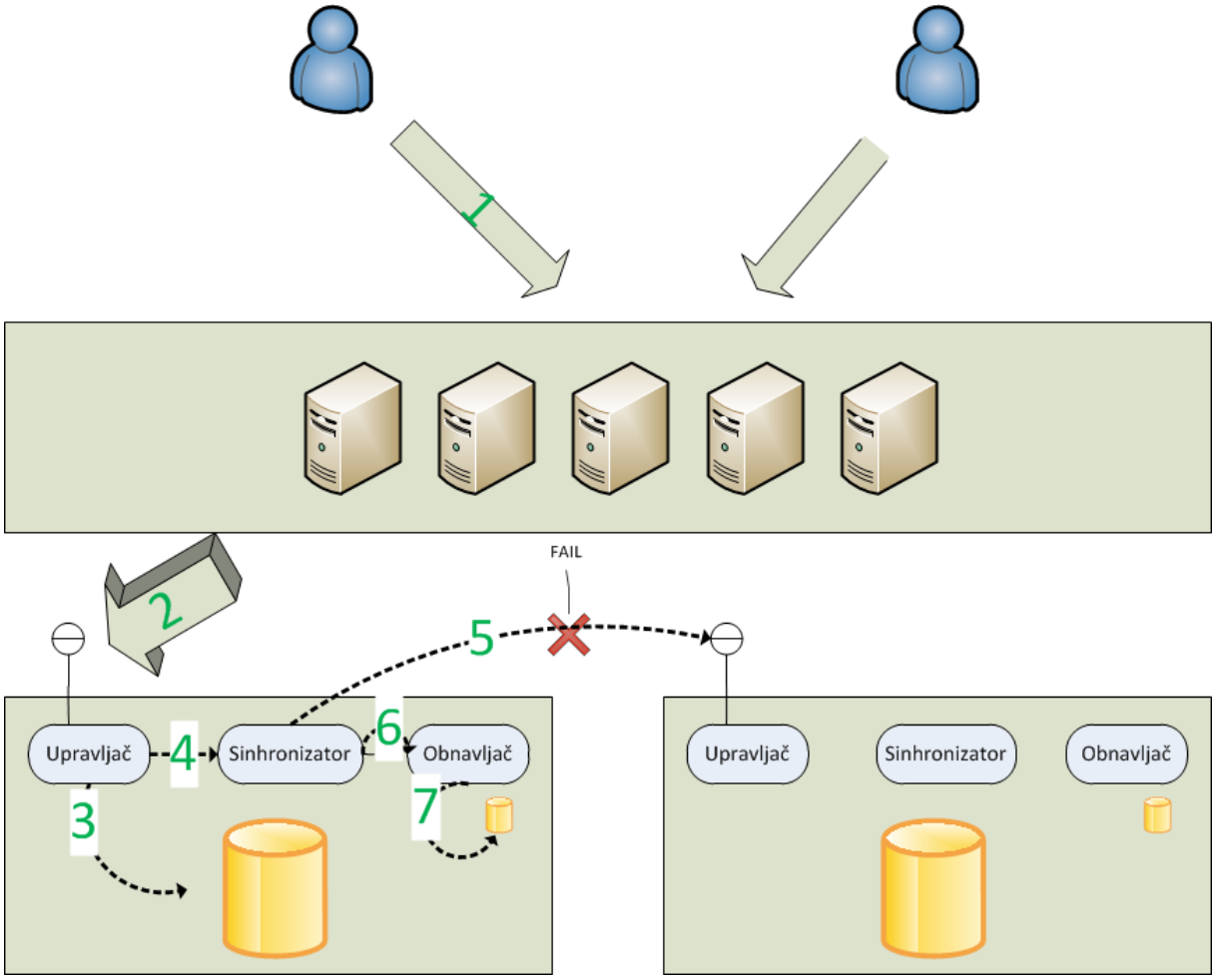


Slika 1. Slučaj ispravnog pohranjivanja podatka u sve baze u sustavu

1. Korisnik stvori novi podatak i šalje ga poslužitelju.
2. Poslužitelj obrađuje podatak i šalje ga odabranom čvoru sustava za sinkronizaciju na pohranu.
3. Upravljač pohranjuje podatak u bazu podataka.
4. Upravljač obavještava sinkronizatora da je pristigao novi podatak.

5. Sinkronizator prosljeđuje podatak svim ostalim čvorovima u sustavu.
6. Upravljači ostalih čvorova spremaju podatak u svoju bazu podataka.

5.2 Neuspješno pohranjivanje podataka

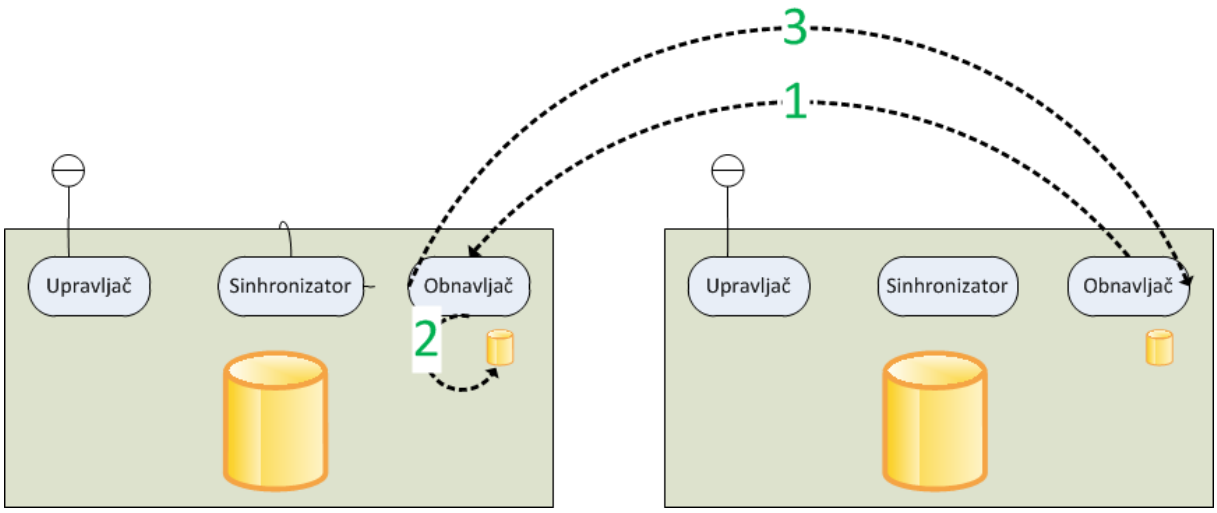


Slika 2. Slučaj neispravnog pohranjivanja podatka u jednu bazu podataka

1. Korisnik stvori novi podatak i šalje ga poslužitelju.
2. Poslužitelj obrađuje podatak i šalje ga odabranom čvoru sustava za sinkronizaciju na pohranu.
3. Upravljač pohranjuje podatak u bazu podataka.
4. Upravljač obavještava sinkronizatora da je pristigao novi podatak.

5. Sinkronizator prosljeđuje podatak svim ostalim čvorovima u sustavu. Sinkronizator dobiva poruku da nije uspješno proslijeđen podatak zbog nedostupnosti željenog čvora.
6. Podatke i adresu nedostupnog čvora prosljeđuje obnavljaču.
7. Obnavljač sprema u svoju lokalnu bazu dobivene podatke i čeka ponovnu prijavu trenutno nedostupnog čvora.

5.3 Tijek obnove čvora



Slika 3. Metoda obnavljanja sustava nakon ispada

1. Čvor nakon vraćanja u aktivno stanje se javlja ostalim čvorovima i potražuje podatke koje nije pohranio dok je bio nedostupan.
2. Ostali čvorovi pregledavaju da li posjeduju tražene podatke.
3. Ukoliko posjeduju, prosljeđuju podatke te ih brišu iz lokalne baze obnavljača.

6. Korištene tehnologije

6.1 SOA

Računarstvo zasnovano na uslugama (engl. *Service-oriented architecture*[9]) je skup oblikovnih obrazaca koji se koriste tijekom faza razvoja i integracije sustava.

Računarstvo zasnovano na uslugama zahtijeva labavo uparivanje (engl. *loose coupling*) usluga s operacijskim sustavima i ostalim tehnologijama na kojima su aplikacije izgrađene. Uslužno orijentirana arhitektura razdvaja mogućnosti sustava u karakteristične jedinice ili usluge, koje su mrežno dostupne kako bi ih korisnici mogli koristiti. Korisnici komuniciraju s uslugom prosljeđujući podatke u dobro definiranom, zajednički korištenom formatu ili koordinirajući aktivnosti između dvije ili više usluga.

Projektanti uslužno-orijentiranih sustava povezuju pojedinačne objekte uslužno orijentiranih arhitektura koristeći orkestraciju. U procesu orkestracije, usluge se povezuju u ne-hijerarhijski poredak koristeći programski alat koji sadrži cjelovitu listu svih dostupnih usluga, njihovih karakteristika i sredstava za izgradnju aplikacije korištenjem ovih sredstava.

Povezivanje usluga zahtijeva detaljne meta-podatke koji opisuju ne samo karakteristike usluga, već i podatke koje koriste. Za strukturiranje meta-podataka koristi se WSDL[4] (engl. *Web Services Description Language*) koji je temeljen na XML-u.

Osnovno načelo uslužno orijentirane arhitekture je apstrakcija. Apstrakcijom usluga predstavlja jednostavno sučelje koje skriva svoju temeljnu kompleksnost. Na taj način korisnici mogu pristupiti neovisnim uslugama bez znanja o implementacijskoj platformi usluge.

Uslužno orijentirana arhitektura se temelji na uslugama koje svoje mogućnosti izlažu putem sučelja. Na taj način druge aplikacije i usluge mogu dohvatiti opis usluge kako bi znale komunicirati s njom.

6.2 WCF

WCF (engl. *Windows Communication Foundation*) je aplikacijsko sučelje unutar .NET radnog okvira (engl. *Framework*) za izgradnju aplikacija i sustava temeljenih na uslugama.

6.2.1 Arhitektura

Arhitektura WCF-a je dizajnirana u skladu s načelima računarstva zasnovanog na uslugama. Usluge su međusobno labavo povezane i time olakšavaju izmjene unutar same usluge, bez utjecaja na korisnike koji pristupaju uslugama. Najčešće imaju WSDL opis preko kojeg korisnici mogu koristiti željenu uslugu. WCF implementira mnoge napredne standarde usluga[6] (engl. *Web Standards*) kao što su *WS-Addressing*, *WS-ReliableMessaging* i *WS-Security*.

6.2.2 Usluge

WCF usluga se sastoji od tri osnovna dijela:

- razreda koja implementira ponašanje usluge,
- okruženja domaćina (engl. *host environment*) koji udomljuje uslugu i
- jedne ili više krajnjih točaka na koje će se klijenti spajati.

Kod operativnih sustava Windows Vista, Windows Server 2008 i Windows 7 (operativnih sustava koji uključuju IIS 7(engl. *Internet Information Services*)), također WAS[7] (engl. *Windows Activation Service*) može biti korišten za udomljavanje WCF usluge.

WCF usluga također može biti udomljena u IIS-u ili samo-udomljena u bilo kojem procesu korištenjem WCF *ServiceHost* razreda. Samo-udomljena WCF usluga može biti korištena putem konzolne aplikacije, *Windows Forms* aplikacije ili Windows usluge.

6.2.3 Krajnja točka

Komunikacija WCF usluge i korisnika se odvija preko krajnjih točaka (engl. *endopoint*). Krajnja točka posjeduje ugovor koji određuje koje će metode usluge biti dostupne korisnicima. Svaka krajnja točka može izlagati različitu skupinu metoda. Također definiraju vezu (engl. *binding*) koja određuje kako će klijent komunicirati s uslugom i adresu gdje će krajnja točka biti udomljena.

Krajnja točka WCF usluge se sastoji od:

- adrese koja specificira gdje poruke mogu biti poslane odnosno gdje je usluga aktivna,
- veze koja opisuje kako slati poruke i

- ugovora (engl. *contract*) koji opisuje što bi poruka trebala sadržavati.

Klijenti moraju poznavati sva tri prethodno navedena parametra kako bi mogli komunicirati s uslugom. Krajnja točka ponaša se kao vrata (*engl. port*) u vanjski svijet. WSDL je namijenjen opisivanju krajnjih točaka usluge na standardiziran način. Opis se sastoji od informacije što usluga može raditi, kako se usluzi može pristupiti te gdje se usluga može pronaći.

Na strani klijenta postoji samo jedna krajnja točka koja se sastoji od adrese, veze i ugovora. U stvarnom sustavu, klijent može komunicirati samo sa metodama usluge koje su predefinirane u ugovoru koji on koristi, ali ne i s cjelokupnom uslugom (ukoliko usluga posjeduje više ugovora). Navedeni način se naziva korištenje posrednika za uslugu. Također, moguće je da korisnik bude spojen na više usluga korištenjem više posrednika sa različitim krajnjim točkama (tj. spojen je na različite usluge).

Usluga može imati višestruke krajnje točke. Kada pristigne poruka na krajnju točku ona je obradi i vrati odgovor na adresu od kuda je pristigla poruka

6.2.4 Adresa

Adresiranje usluge neophodno je za mogućnost korištenja usluge. Nužno je imati adresu usluge kako bi joj korisnici mogli poslati poruku. Adrese u WCF-u su URL-ovi koji definiraju korišten protokol, računalo na kojem je usluga pokrenuta te put (engl. *path*) prema usluzi. Broj vrata u URL-u je polje prepušteno slobodnom izboru i ovisi o korištenom protokolu.

WCF podržava nekoliko protokola, te svaki od njih ima svoj vlastiti pojedinačni adresni format. WS-Adresiranje je nužno kada WCF usluge koriste bilo koji drugi protokol osim HTTP-a. Osnova usluge je protokol SOAP. U SOAP poruci, krajnje su točke izražene kao WS-Adressing krajnje točke pomoću kojih je moguće dodavati specifična zaglavlja unutar same SOAP poruke.

6.2.5 Veza

Veza definira kako korisnici komuniciraju s uslugom. Odabir vrste detalja veze ima najveći utjecaj na obilježja sustava.

Veza upravlja:

- transportnim protokolom (HTTP, MSMQ[8] (engl. *Microsoft Message Queuing*), imenovani cjevovodi,TCP),

- vrstom komunikacije (jednosmjerna, dvosmjerna, upit-odgovor),
- kodiranjem poruka (XML, binarno,MTOM…) i
- uključenim WS.* standardima (*WS-Security*, *WS-Federation*, *WS-Reliability*, *WS-Transactions*).

6.2.6 Ugovor

Ugovori omogućuju strogo odvajanje ponašanja kojeg usluga pruža na korištenje vanjskom svijetu od unutrašnje implementacije same usluge.

Ugovor može deklarirati:

- izloženo ponašanja (ugovor usluge),
- tipove podataka koje usluga koristi (ugovor podatka) ili
- strukturu poruke (ugovor poruke).

U sljedećim odlomcima biti će opisani načini vrste komunikacije između usluge.

6.2.7 Način razmjene WCF poruka

Poruke upit - odgovor

Upit odgovor (Engl. *Request-reply*) je dvostrana operacija gdje se svaki zahtjev uparuje s odgovorom. Korisnik na svaku poslanu poruku čeka pripadajući odgovor.

Jednosmjerne poruke

Jednosmjerne poruke rade na načelu da usluga ili klijent samo pošalju poruku te ih ne zanima odgovor. Znači, poruka je prenesena bez čekanja na odgovor od usluge. Ovo načelo je slično pozivanju asihronih metoda.

Dvostrana poruka

Slanje dvostranih poruka je malo složeniji način komunikacije. Dvostruki kanal je stvarna čvor-čvor veza *(*engl*. Peer to peer connection)* gdje i korisnik i usluga mogu biti i primatelj i pošiljatelji (na različitim kanalima).

6.2.8 Ugovori usluge i operacije

Ugovor usluge (engl*. Service contract*) opisuje metode koje su izložene vanjskom svijetu. Obično se naziva i sučelje usluge ili izloženo ponašanje usluge. Ugovori

usluge implementirani su kao .NET sučelja. Kako bi klase postale WCF ugovori usluga, moraju biti obilježena s *[ServiceContract]* atributom.

Radi utjecaja na ugovor usluge, *[ServiceContract]* atribut ima nekoliko parametara koji imaju svoje vlastite funkcije:

- *CallbackContract* označava tip ugovora povratnog tipa (*engl. Callback*). Povratni tip je koristan kod korištenja obrasca dvostrane razmjene poruka.
- *SessionMode* označava vrijednost koja definira zahtijeva li ugovor korištenje korisničkih sesija. *SessionMode* je enumeracija s mogućim vrijednostima dopušteno, nedopušteno i obavezno.
- ....

Atribut *[OperationContract]* opisuje način korištenja metode usluge. Neki od atributa su:

- *Name* specificira ime operacije.
- *IsInitiating* definira vrijednost koja govori da li metoda započinje sjednicu na poslužitelju usluge.
- *IsOneWay* definira vrijednost koja govori da li operacija vraća povratnu poruku.
- *IsTerminating* definira vrijednost koja govori uzrokuje li operacija zatvaranje sjednice od strane poslužitelja kada je povratna poruka poslana.

6.2.9 Ugovori podataka

WCF usluga mora znati kako pretvoriti korisnički implementirane razrede u standardne .NET tipove. Kod jednostavnih tipova podataka nema potrebe za pretvorbom jer ih WCF zna pretvoriti u SOAP tipove podataka. Kod složenijih tipova postoje dva načina pretvorbe:

1. Ukoliko su vlastiti tipovi izgrađeni na temelju jednostavnih tipova u .NET-u, ugovor podataka se može označiti samo sa [*Serializable*] atributom. Na taj način WCF zna kako koristi implicitne ugovore podataka. U ovom načinu pretvaranja se ne mora definirati ugovor podataka.
2. Za drugi način pretvorbe, gdje korisnik utječe na način na koji želite da se pretvorba obavi, potrebno je definirati eksplicitan ugovor podataka za vlastiti tip podatka. Definiranjem jednostavne klase sa svim potrebnim obilježjima te

dodavanjem [*DataContract*] atributa, omogućujemo WCF usluzi da sama obavi pretvorbu potrebnih podataka.

U nastavku je prikazan primjer ugovora podatka korištenog pri komunikaciji sustava za sinkronizaciju. Ispred imena razreda se nalazi atribut [*DataContract*] koji govori da će taj razred biti potrebno pretvarati u oblik pogodan za prijenos preko mreže. Atributi [*DataMember*] govore da označene članske varijable moraju biti javno dostupne nakon ponovnog pretvaranja u izborni oblik.

```
[DataContract]

public class Row : IRow
    {
        List<object> _values;

        public Row() { _values = new List<object>(); }

        public Row(List<object> values) { _values = values; }

        [DataMember]
        public List<object> Values
        {
            get { return _values; }
            set { _values = value; }
        }

        public override string ToString()
        {
            return _values.Aggregate("", (current, o) => current + (o + "#"));
        }
    }
```

7. Programsko ostvarenje

Programsko ostvarenje sustava za sinkronizaciju baza podataka je ostvareno u Visual Studio 2010 radnoj okolini pod .NET 4.0 [2] radnim okvirom.

7.1 Sučelja korištena za komunikaciju usluga

Sustav za komunikaciju koristi vlastiti oblik prezentacije tablice u bazi kako bi se omogućila što veća interoperabilnost i nezavisnost o platformi.

Tablica se sastoji od imena, skupa redaka te skupa stupaca. Svaki stupac posjeduje informacije o imenu stupca, zastavici koja označava radi li se o primarnom ili o stranom ključu tablice. Redak u tablici sadrži samo listu objekata koje prezentiraju ćelije u tablici.

Svaki razred korišten pri komunikaciji mora implementirati navedeno sučelje kako bi WCF mogao pretvarati podatke u pogodan oblik za prijenos preko mreže.

U nastavku su prikazana konkretna sučelja korištena za komunikaciju.

```
public  interface IColumn
{
   String CloumnName { get; set; }
   bool IsPrimaryKey { get; set; }
   bool IsForeignKey { get; set; }
}

public interface IRow
{
    List<Object> Values { get; set; }
}

public interface ITable
{
    String Name { get; set; }
    List<IColumn> Columns { get; set; }
    List<IRow> Rows { get; set; }
}
```

Kako bi se omogućila neovisnost o konkretnoj bazi podataka, potrebno je ostvariti razred koji implementira sučelje `IDatabase` koje je prikazano u nastavku teksta.

```
public interface IDatabase
    {
        List<String> GetAllTableNames();
        List<String> GetPrimaryKeys(String tableName);
        List<String> GetForeignKeys(String tableName);
        List<String> GetTableColumnsNames(String tableName);
        List<IColumn> GetTableColumns(String tableName);
        List<IRow> GetRowsByPkList(string tableName, List<object> pkList);
        ITable GetTableByName(string tableName);
        bool OpenConnection();
        bool Close();
        ITable ExecuteReader(string sql);
        object ExecuteScalar(string sql);
        bool ExecuteNonQuery(string tableName, int operatoin, List<IRow> rows);
        bool ExecuteNonQuery(string sql);
        List<IRow> ExecuteReaderRows(string sql);
    }
```

7.2 Algoritmi provjere konzistentnosti sinkronizacije

Tokom sinkronizacije podataka vrlo lako može doći do pojave nekonzistencije podataka. Za nekonzistenciju je zadužena usluga obnavljač. Da bi obnavljač saznao da li nedostaju određeni podatci potrebni su algoritmi za detekciju nekonzistencije.

Algoritam provjere radi na načelu izračunavanja sažetka svakog reda u tablici. Kada se izračuna sažetak svih redova, obavlja se operacija EX-ILI nad bitovima svih sažetaka. Time se dobije jedinstveni sažetak tablice veličine 160 bitova.

Podatci u tablici ne moraju nužno pristizati istim redoslijedom te se može dogoditi da poredak nije u svakoj tablici isti. Operacija EX-ILI je asocijativna tako da će rezultat biti jednak bez obzira na redoslijed dohvata sažetaka redova.

7.3 Kardinalnost usluga

Svaki korisnik kada se prijavi na određeni čvor dobije vlastitu instancu upravljača. Primjerice ukoliko u sustavu postoje dva čvora za sinkronizaciju (dvije baze podataka) i na jedan se čvor prijavi 10 korisnika, a na drugi 100, prvi će čvor imati 10 instanci upravljača, a drugi 100 instanci.

Svaka instanca upravljača posjeduje vlastitu instancu sinkronizatora i vlastitu vezu na bazu podataka. Sustav je mogao imati statičke veze na bazu i sve instance upravljača dijeljenog jednog sinkronizatora, što ima svoje prednosti i nedostatke.

Prednosti bi se očitovale u svojstvima rada sustava i manjem zauzeću sredstava, ali jedan veliki nedostatak su transakcije.

Ukoliko bi jedan klijent započeo transakciju i nakon određenog vremena napravio poništenje transakcije, tada bi se svi podatci od svih korisnika, koje je dijeljeni (statički) sinkronizator objavio između početka i poništenja transakcije, poništili.

Usluga obnavljač je dijeljena, tj. svaki čvor posjeduje samo jednu instancu obnavljivača.

Kada se sve zbroji, u prvom čvoru će biti aktivnih 10 upravljača, 10 sinkronizatora, 10 veza na bazu i 1 obnavljač, dok će u drugom biti aktivnih 100 upravljača, 100 sinkronizatora, 100 veza na bazu i 1 obnavljač.

7.4 Upravljač

Upravljač (engl. *controller*) je usluga koja upravlja radom čvora. Sučelje upravljača je jedino izloženo korisnicima za operacije nad podatcima u bazi podataka.

Svaki upravljač ima referencu na sinkronizatora i vezu na lokalnu bazu podataka.

Referenca na sinkronizatora omogućuje objavu pristiglih podataka ostalim čvorovima u sustavu.

Upravljač održava vezu prema bazi podataka u čvoru te obrađuje zahtjeve korisnika za podatcima.

Sustav omogućuje dva načina zahtjeva za podatcima:

- zahtjev koji je potrebno obraditi i izvući podatke i stvoriti SQL naredbe i
- čisti SQL upiti.

Parsiranjem podataka klijentska aplikacija smještaja podatke u liste sučelja `IRow` ili sučelje `ITable`. Podatci pristižu upravljaču koji ih pretvara u SQL naredbe te ih izvršava i prosljeđuje sinkronizatoru.

Metode kojim korisnik može pristupiti su:

- `ExecuteReader`,
- `ExecuteScalar`,
- `ExecuteNonQuery` i
- `ExecuteNonQuerySql`.

`Init` metoda inicijalizira vezu na bazu podataka te stvara i referencira sinkronizatora.

`InitVoid` metoda stvara vezu na bazu i služi samo za primanje podataka od ostalih sinkronizatora. U poglavlju o sinkronizatoru ce biti detaljnije opisano načelo rada.

`ExecuteReader` metoda vraća popunjenu tablicu s podatcima u `ITable` obliku. Prima argument SQL upit kojim želimo dohvatiti podatke.

`ExecuteScalar` također prima kao argument SQL upit te vraća objekt sa rezultatom upita.

`ExecuteNonQuery` metoda omogućuje unos, brisanje ili izmjenu podataka. Kao argument prima listu podataka tipa `IRow`, operaciju koje će se izvršavati (unos, brisanje, izmjena) te ime tablice nad kojom se operacije izvršavaju. Podatci iz sučelja `IRow` će se obraditi te pretvoriti u SQL upit. Pretvorba se vrši unutar razreda koji ostvaruje `IDabase` sučelje.

`ExecuteNonQueySql` prima kao argument SQL naredbu koju će izvršiti nad lokanom bazom te svima ostalima proslijediti. SQL naredba može biti bilo kojeg oblika te se tu stvaraju potencijalni problemi sigurnosti koji prelaze temu ovoga rada.

`NotifiySync` metoda se koristi za javljanje sinkronizatoru da su stigli novi podatci koji se trebaju objaviti.

`IsAlive` metoda omogućuje saznavanje da li je trenutni upravljač dostupan.

Metoda `CalculateXorHash` izračunava sažetak cijele tablice koja se prima kao argument.

Metode `SyncExecuteNonQuery`, `SyncExecuteNonQuerySql` i `InitVoid` će biti objašnjene unutar poglavlja o sinkronizatoru.

```
namespace wsController
{
    [ServiceContract(SessionMode = SessionMode.Required)]
    [ServiceKnownType(typeof(Table))]
    [ServiceKnownType(typeof(Row))]
    [ServiceKnownType(typeof(Column))]
    public interface IController : IConnect
    {
        [OperationContract(IsInitiating = true)]
        bool Init(string connectionSTring, string catalogName);

        [OperationContract(IsInitiating = true)]
        bool InitVoid(string connectionSTring, string catalogName);

        int State { [OperationContract] get; [OperationContract] set; }

        [OperationContract(IsInitiating = false)]
        ITable ExecuteReader(string sql);

        [OperationContract(IsInitiating = false)]
        object ExecuteScalar(string sql);

        [OperationContract(IsInitiating = false)]
        void ExecuteNonQuery(string tableName, int operation, List<IRow> rows);

        [OperationContract(IsInitiating = false)]
        void ExecuteNonQuerySql(string sql);

        [OperationContract(IsInitiating = false, IsOneWay = true)]
        void SyncExecuteNonQuery(string tableName, int operation, List<IRow> rows);

        [OperationContract(IsInitiating = false, IsOneWay = true)]
        void SyncExecuteNonQuerySql(string sql);

        [OperationContract(IsInitiating = false)]
        string CalculateXorHash(ITable tableName);

        [OperationContract(IsInitiating = false)]
        void NotifySync(object data);

        [OperationContract]
        bool IsAlive();
    }
}
```

7.5 Sinkronizator

Usluga sinkronizator objavljuje lokalne promjene baze podataka svim ostalim čvorovima u sustavu.

Pri inicijalizaciji sinkronizatora se dohvaćaju reference na sve ostale upravljače uključene u sustav. Nakon dohvata adresa upravljača, sinkronizator poziva njihovu metodu `InitiVoid` koja stvara vezu na bazu, ali ne i nove sinkronizatore. Na taj način je osigurano da se pristigli podatci ponovo ne prosljeđuju u sustav.

U sustavu moraju biti predefinirani svi čvorovi te se adrese upravljača svakog čvora čitaju iz konfiguracijske datoteke.

Metode `ReceiveData` i `ReceiveDataSql` primaju podatke od svoga upravljača te ih prosljeđuju svim ostalim upravljačima u sustavu.

Prosljeđivanje podataka je omogućeno pomoću metoda `SyncExecuteNonQuery`, `SyncExecuteNonQuerySql` upravljača. Navedene metode samo pohranjuju promjene u lokalnu bazu podataka.

Sinkronizator posjeduje referencu na obnavljivača. Ukoliko jedan od čvorova postane nedostupan, podatke namijenjene čvoru i adresu čvora prosljeđuje obnavljaču koji ih sprema u vlastitu SQLLite bazu podataka.

```
namespace wsSync
{
 [ServiceContract(SessionMode = SessionMode.Required)]
    [ServiceKnownType(typeof(Row))]
    public interface ISync
    {
        [OperationContract]
        bool Init();

        [OperationContract]
        bool IsAlive();

        [OperationContract(IsOneWay = false)]
        void ReciveData(string tableName, int operation, List<IRow> data);

        [OperationContract(IsOneWay = false)]
        void ReciveDataSql(string sql);

        void GetObservers();
    }
}
```

7.6 Obnavljač

Obnavljač ima ulogu vraćanja sustava u konzistentno stanje nakon ispada.

Nakon podizanja sustava, obnavljač stvara reference na sve ostale obnavljače u sustavu. Zatim im javlja da je ponovo došao u aktivno stanje i provjerava da li postoje podatci koje nije primio i, ukoliko postoje, potražuje ih.

Metoda `BackupSendFailData` sprema u lokalnu SQLLite bazu podataka adresu čvora koji nije dostupan, ime tablice u koju se trebaju podatci pohraniti te podatke koji nisu uspjeli se poslati.

Metoda `SendImOnline` javlja ostalim obnavljačima da je čvor postao ponovno dostupan.

Metoda `GetDataForClient` vraća listu podataka za čvor koji je pozvao tu metodu.

```
namespace wsRestorer
{
    [ServiceContract(SessionMode = SessionMode.Required)]
    [ServiceKnownType(typeof(Row))]
    public interface IRestorer
    {
        [OperationContract]
        void Init();

        void GetRestorers();

        [OperationContract]
        void ServiceOnline();

        [OperationContract]
        bool BackupSendFailData(List<IRow> rows, string tableName, string host);

        [OperationContract]
        bool IsServiceOnline();

        bool SendImOnline();

        [OperationContract]
        List<IRow> GetDataForClient(string tableName);

        void ReceiveRestoreData(List<IRow> rows, string tableName);

        [OperationContract]
        bool CheckDataForClient();

        [OperationContract]
        void ReceiveHashData(object data);
        void CheckWhoIsOnline();
    }}
```

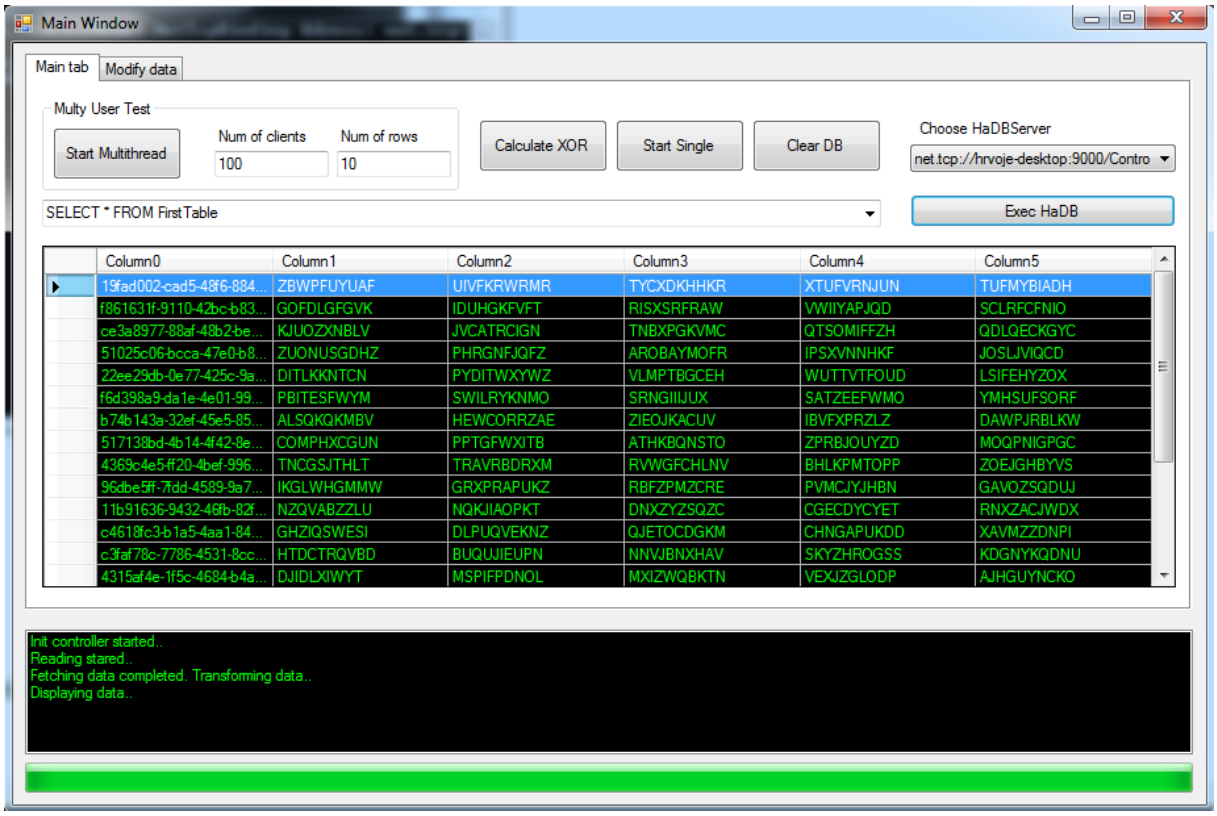
7.7 Testna aplikacija

Za prikaz rada sustava izrađena je aplikacija sa mogućnostima unosa, brisana, izmjene i dohvata podataka. Također su ugrađeni testovi za simuliranje jednog korisnika koji unosi proizvoljan broj zapisa i test za simulaciju proizvoljnog broja istovremenih korisnika s proizvoljnim brojem zapisa u bazu.

Odabirom padajućeg izbornika 'Choose HaDBServer' klijent može birati na koji će čvor slati zahtjeve.

Ugrađena je mogućnost brisanje svih zapisa u bazu radi lakše provjere provedenih akcija.

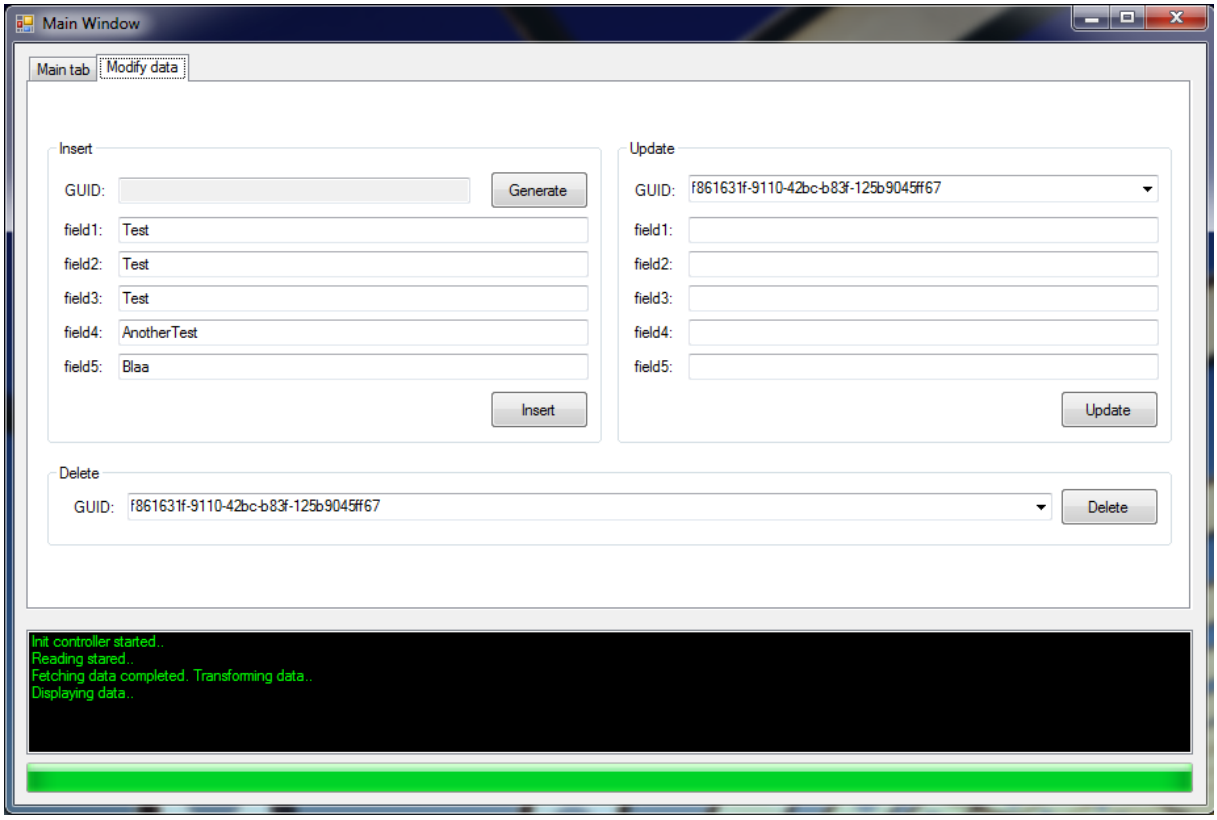
Izvršavanje SQL naredbi nad sustavom je omogućeno preko linije za upis teksta i pritiskom na gumb 'Exec HaDB'.



Slika 4. Slika glavnog prozora testne aplikacije

Nakon svake operacije se može provjeriti uspješnost izvedene operacije. Provjera se vrši pritiskom na gumb 'Calculate Xor'. Ukoliko se sažetci podudaraju operacija je uspješno završena. Ukoliko se ne podudaraju, moguće je da još nisu pristigli svi podatci u sve čvorove ili da sinkronizacija nije uspješno obavljena.

U prozoru 'Modify data' omogućen je unos, izmjena i brisanje podataka u sustavu.



Slika 5. Prozor za izmjenu podataka

7.8 Pokretanje aplikacije

U privitku (CD-u) diplomskog rada se nalaze sve potrebne datoteke i upute za pokretanje i testiranje izrađenog programskog ostvarenja.

8. Rezultati ispitivanja učinkovitosti sustava

Sustav za sinkronizaciju je testiran na dva čvora, temeljenih na Windows 7 operacijskom sustavu.

Tablica 2. Konfiguracija računala korištenih pri testiranju

	RAM	CPU
Čvor 1	2 GB(667 Mhz)	Intel Pentium D 920@3.2
Čvor 2	6 GB(1866 Mhz)	Core i7 930@2.8

Podatci su slučajno generirani te su se unose u tablicu oblika:

```
FirstTable(
      ID uniqueidentifier PRIMARY KEY,
      field1 varchar(50),
      field2 varchar(50),
      field3 varchar(50),
      field4 varchar(50),
      field5 varchar(50)
)
```

Prvo mjerenje je obavljeno sa jednim korisnikom koji je generirao i pohranjivao različite količine podataka u bazu podataka. Vrijeme potrebno za pohranu i sinkronizaciju se vidi na slici 6.

Slika 7. prikazuje mjerenje ovisnost broja korisnika i potrebnog vremena unosa 100 zapisa u bazu podataka.

Slika 6. Rezultat testa s jednim korisnikom i promjenjivom količinom podataka

Test s jednim korisnikom pokazuje kako vrijeme potrebno za sinkronizaciju počinje naglo rasti za više od 1000 zapisa. Razlog naglog porasta je povećanje vremena potrebnog za pretvaranje podataka u pogodan oblik za prijenos preko mreže.

Slika 7.Rezultat testa s promjenjivim brojem korisnika i konstantnim brojem zapisa (100)

Kao i u prethodnom testu, test s promjenjivim brojem korisnika ima točku prijeloma. Za više od 100 istovremenih korisnika vrijeme potrebno za sinkronizaciju naglo raste. Razlog ovakvog porasta je ograničenost računalnih sredstava na testnom sustavu. Testna računala pokreću sustav za sinkronizaciju, generiraju virtualne korisnike i

stvaraju testne podatke, što su računalno zahtjevne operacije. Nakon 100 istovremenih korisnika dolazi do preopterećenja testnih računala i svojstva sustava za sinkronizaciju naglo počinju padati.

9. Zaključak

Danas usluge i korisnici postaju sve zahtjevniji za kvalitetnim sustavima. Ukoliko usluga nije odgovarajuće kvalitete korisnici traže zamjenu. Da bi neka organizacija zadržala korisnike i potencijalno povećala profit, potrebna je što kvalitetnija pružena usluga.

Visoka dostupnost sustava je jedan od glavnih čimbenika kvalitete sustava. Mogućnost korisnika da u svakom trenutku mogu pristupiti sustavu pruža veće zadovoljstvo i povjerenje u korišteni sustav.

Cilj ovog diplomskog rada bio je istražiti mehanizme ostvarivanja visoke dostupnosti u sustavima s bazama podataka. Tokom izrade sustava, riješeni su razni arhitekturalni problemi koje je trebalo dobro proučiti kako bi se osigurala što veća skalabilnost te poboljšale značajke sustava.

Izgrađeni sustav pokazuje zadovoljavajuće vrijeme odziva za sinkronizaciju. Za stvarno korištenje sustava bi trebalo osigurati sigurnosne mehanizme kako svaki prosječan korisnik ne bi mogao poslati destruktivne naredbe koje bi ugrozile dostupnost sustava.

10. Literatura

[1] SAF, *Service Availability forum*
http://en.wikipedia.org/wiki/Service_Availability_Forum , www.opensaf.org

[2] .NET Framework 4.0, http://en.wikipedia.org/wiki/.NET_Framework

[3] Amit Bahree, Shawn Cicoria, Dennis Mulder, Nishith Pathak, Chris Peiris, *Pro WCF: Practical Microsoft SOA Implementation,* Siječanj 2007

[4]WSDL, *Web Services Description Language*,
http://en.wikipedia.org/wiki/Web_Services_Description_Language

[5]SOAP, *Simple Object Access Protocol***,** http://en.wikipedia.org/wiki/SOAP

[6] WS-*, *Web Standards*, http://en.wikipedia.org/wiki/WS-*

[7] WAS, *Windows Activation Services*
http://en.wikipedia.org/wiki/Windows_Activation_Services

[8] MSMQ, *Microsoft Message Queuing*,
http://en.wikipedia.org/wiki/Microsoft_Message_Queuing

[9]SOA, *Service-oriented architecture,*

http://en.wikipedia.org/wiki/Service-oriented_architecture

11. Sažetak

U modernom računarstvu sve se više javljaju zahtjevi za visoko dostupnim sustavima. Ovaj rad obrađuje metode osiguravanja sustava visoko dostupnim.

Kako se sve više očituju prednosti raspodijeljenog pristupa u izgradnji sustava, izgrađeni primjer sustava koji osigurava baze podataka od višestrukog zatajenja, se temelji na navedenoj arhitekturi.

Sustav osigurava metode međusobne sinkronizacije baza podataka. Također, omogućuje vraćanje u konzistentno stanje svih baza podataka ukoliko dođe do ispada jednog ili više čvorova.

Ključne riječi: visoka dostupnost, baze podataka, sinkronizacija, replikacija, WCF, SOA

12. Summary

Title: Databases in high available systems

Modern computing demands high availability. The aim of this thesis was to research and implement the methods and algorithms to keep systems highly available.

There are many benefits in building systems in a distributed way and therefore, the implemented system which keeps the system databases highly available, is built on that principle.

The system implements database synchronization methods. Also, it provides database recovery after a system crash.

Keywords: High availability, databases, synchronization, replication, WCF, SOA