UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jože Matko

PRIMERJAVA WINDOWS COMMUNICATION FOUNDATION S SORODNIMI OGRODJI ZA GRADNJO PORAZDELJENIH APLIKACIJ TIPA ODJEMALEC-STREŽNIK

DIPLOMSKO DELO NA
UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Matija Marolt

Ljubljana, 2008

ZAHVALA

Zahvaljujem se svojemu mentorju, doc. dr. Matiji Maroltu, za pomoč in potrpežljivost pri izdelavi diplomskega dela, svoji družini pa za podporo med časom študija.

KAZALO

POVZETEK

V diplomskem delu predstavljam primerjavo tehnologije Windows Communication Foundation (WCF) z nekaterimi primerljivimi sorodnimi tehnologijami za gradnjo aplikacij tipa odjemalec strežnik. Namen dela je predstaviti problematiko izdelave aplikacije tipa odjemalec strežnik, proučiti nekatere obstoječe tehnologije za implementacijo ter, na podlagi ugotovitev primerjave, uporabiti primerno tehnologijo pri implementaciji aplikacije za upravljanje s sredstvi in obveznostmi. Primerjane tehnologije so višjenivojske, temeljijo na principu klica oddaljene metode in objektno usmerjenem modelu programiranja. Ti principi omogočajo razvijalcu abstrakcijo od nižjenivojskih podrobnosti komunikacije. Ugotovitve primerjave sem nato uporabil pri izbiri primerne tehnologije za izdelavo komunikacijskega modula aplikacije za upravljanje s sredstvi in obveznostmi. Glavne značilnosti aplikacije so: velika količina podatkov, majhna količina komunikacije in varnost. Na izbiro tehnologije je imel, poleg značilnosti aplikacije, velik vpliv programski model in nekatere druge podrobnosti povezane z njeno uporabo. Izbral sem tehnologijo WCF in v zaključnem delu implementiral komunikacijski modul aplikacije za upravljanje s sredstvi in obveznostmi.

KLJUČNE BESEDE: odjemalec, strežnik, WCF, porazdeljene aplikacije, varnost, pristnost, pooblastila, velika količina podatkov

ABSTRACT

This B. Sc. degree discusses the use of the Windows Communication Foundation (WCF) for building client server application and compares it with other similar technologies. The purpose of this work is to analyse the client server architecture for building distributed applications, to overview some high level technologies that can be used for implementation and, finally, to implement an asset liability application using the most appropriate technology. I will focus on technologies that are based on remote procedure call and object oriented programming models. The use of these models allows the developer to forget about the low level communication issues and focus on the problem domain. Further on, I will use the conclusions of the overview to pick the most appropriate technology for implementing the communication module of the asset liability application. The specifics of the application are large amount of data, low communication footprint and security. Another criterion for choosing the technology is the programming model. I chose WCF and use it for implementing the communication module of the business application.

KEYWORDS: client, server, WCF, distributed application, security, authentication, authorization, large amount of data

1 UVOD

Računalništvo in informatika je področje, ki se je v kratkem času hitro razvilo v nepogrešljiv del vsakdanjega življenja. Tradicionalno se je računalnik uporabljal kot samostojno orodje za preproste analize podatkov, kreiranje predstavitev, pisanje besedil in podobno. Komunikacija ni bila možna oziroma je bila težavna. S prihodom računalniških omrežji so se začeli računalniki povezovati in med seboj komunicirati. Računalnikom oziroma programom, ki se nahajajo na več računalnikih in med seboj komunicirajo, rečemo porazdeljeni sistemi oziroma porazdeljene aplikacije.

Pri vsakem računalniškem sistemu je potrebno zagotoviti konsistentno, zanesljivo in varno delovanje. Porazdeljeni sistemi niso izjema. Da bi dosegli ta cilj, pa moramo pri porazdeljenih sistemih rešiti nekatere probleme, ki jih pri navadnih aplikacijah ne srečamo. Moduli pri porazdeljeni aplikaciji se nahajajo na različnih računalnikih. Prvi problem je lokacija podatkov. Vsi moduli morajo imeti v kateremkoli trenutku dostop do enakih skupnih podatkov. Drugi problem je zanesljivost delovanja. Stoodstotno zanesljivega delovanja nikoli ne moremo doseči, saj vedno obstaja možnost okvare kateregakoli dela aplikacije. Poskrbeti pa moramo za kontrolirano odzivanje na okvare oziroma napake pri delovanju. Pri poslovnih aplikacijah pa se pojavi še en problem, ki je najbolj pereč. Varnost je v poslovnem okolju na prvem mestu, ko govorimo o prenašanju podatkov preko omrežja, še posebej če prenašamo podatke preko svetovnega spleta.

Vse te probleme je potrebno rešiti. Rešitve niso vedno optimalne in zahtevajo poznavanje problema, ki ga poskušamo rešiti.

1.1 ARHITEKTURA ODJEMALEC STREŽNIK

Arhitektura odjemalec strežnik opisuje relacijo med dvema računalnikoma oziroma programoma, kjer je eden strežnik drugi pa odjemalec. Odjemalec je pobudnik komunikacije in pošlje zahtevo strežniku, ki jo sprejme, izvede zahtevano operacijo in rezultate vrne odjemalcu. Odjemalcev je lahko več, strežnik pa je za določeno storitev logično eden, fizično pa jih je lahko več. Na ta način se poveča dostopnost storitve, potreben pa je nek mehanizem za porazdeljevanje zahtev med fizične strežnike. V primeru, da mora klic potekati v obratno smer, sta vlogi odjemalca in strežnika zamenjani. Arhitektura se lahko uporabi za komunikacijo na enem samem računalniku med dvema programoma, tipično pa se uporablja v omrežju. V njem ta model ponuja prikladen način za integracijo porazdeljenih aplikacij.

Uporaba tega modela je zelo razširjena in je osnovni model pri gradnji porazdeljenih aplikacij. Uporablja se v večini danes zgrajenih poslovnih aplikacijah in storitvah na internetu, kot so elektronska pošta, FTP, spletne storitve, spletne strani in podatkovna skladišča.

Slika 1: Pri arhitekturi odjemalec strežnik odjemalec pošlje zahtevo. Strežnik izvede zahtevano operacijo in vrne odjemalcu odgovor.

Tudi znotraj arhitekture odjemalec strežnik lahko sisteme delimo naprej. Poznamo arhitekturo iz dveh nivojev, kjer odjemalec komunicira direktno s strežnikom. Največkrat je to podatkovna baza. Druga arhitektura oziroma skupina arhitektur pa ima dodaten nivo med odjemalcem in strežnikom. Dodaten nivo ima lahko različne funkcije. Lahko je samo nek vmesnik za prispele zahteve dostopa do baze in hkrati služi kot medpomnilnik za večkrat zahtevane podatke. Lahko pa ima tudi določeno logiko, ki je skupna vsem odjemalcem in se zato lahko nahaja na strežniku.

Z arhitekturo odjemalec strežnik pridobimo na centralizaciji določenih segmentov sistema. Ta centralizacija nam olajša in poenostavi izgradnjo sistema. Podatki, ki se uporabljajo v sistemu, so shranjeni na enem mestu. Takšne podatke je lažje zavarovati, saj so tipično strežniki bolje vzdrževani in varovani, kot osebni računalniki oziroma odjemalci. Zagotovljena je konsistentnost podatkov, saj se te nahajajo na enem mestu oziroma so edini pravilni. Podatki, ki se nahajajo na odjemalcu, so kopija tistih na strežniku ali pa so lokalni in za celoten sistem nepomembni. Spreminjanje podatkov na strežniku je možno pod točno določenimi pogoji, ki jih določa strežnik. Ustvarjanje varnostne kopije podatkov je veliko lažje, saj ustvarimo kopijo samo podatkov na strežniku. Centraliziran je tudi dostop do podatkov, saj lahko vsak odjemalec zahteva podatke samo od strežnika. Ta preveri akreditacijo odjemalca in na podlagi te sprejme ali zavrne zahtevo po podatkih. Varovanje podatkov in upravljanje dostopa do njih je zelo pomembno v poslovnih aplikacijah. Odjemalec ne ve ničesar o implementaciji strežnika in z njim komunicira preko dogovorjenih vmesnikov.

Arhitektura pa ima tudi določene pomanjkljivosti. Z večanjem števila odjemalcev, se poveča tudi obremenjenost strežnika. Če je strežnik preobremenjen, se odzivnost poslabša in lahko za preproste operacije čakamo dolgo časa. Še večji problem se pojavi, če strežnik odpove in nimamo več delujoče aplikacije. Ta problem se rešuje z uporabo nadomestnih strežnikov, na katerih se vzdržuje konsistentna kopija podatkov. Ti strežniki samodejno prevzamejo delo odpovedanega strežnika. Še boljša rešitev je uporaba več strežnikov za obdelavo zahtev odjemalcev, ki logično predstavljajo en strežnik. V takšni konfiguraciji ima lahko vsak strežnik vse podatke ali pa jih ima samo del. V obeh primerih je storitev dostopna preko

posebnega strežnika, ki porazdeljuje breme med dejanske strežnike. Tudi ta strežnik ima lahko redundanco v primeru, da pride do okvare.

1.2 STORITVENO USMERJENA ARHITEKTURA

Servis ali storitev je funkcija, ki jo program ali naprava nudi drugim. Storitveno usmerjeno arhitekturo[1] lahko definiramo, kot skupino storitev, ki med seboj komunicirajo oziroma opravljajo neko nalogo. Na znotraj so storitve nepovezane med sabo in komunicirajo preko zunanjih vmesnikov. Vsaka storitev opravlja neko funkcijo, ki je logična enota in je lahko uporabljena tudi zunaj sistema. Take storitve združimo v sistem, ki opravlja neko kompleksnejšo nalogo. Takšni povezanosti storitev rečemo ohlapna povezanost, saj delujejo neodvisno ena od drugih. Če pride do izpada določene storitve, druge še vedno delujejo naprej in nemoteno opravljajo svoje naloge, ki niso povezane z izpadlo storitvijo.

Storitveno usmerjena arhitektura je nadgradnja arhitekture odjemalec strežnik. Pri tej arhitekturi je vsak modul sistema lahko strežnik ali/in odjemalec. Vsak modul, ki ponuja neko storitev, je strežnik, in vsak modul, ki uporablja neko storitev drugega modula, je odjemalec.

Storitveno usmerjena arhitektura je bolj fleksibilna, saj lahko storitve dodajamo, dopolnjujemo in celo spreminjamo njihovo implementacijo. Če se držimo dogovorjenih vmesnikov za komunikacijo, ostale storitve ne opazijo razlike. Lažja je tudi integracija obstoječih aplikacij. Razvijalec hitreje razvije aplikacijo.

1.3 VARNOST

Varnost računalniškega sistema je obširna tema. Zajema tako fizično zavarovanje sistema, kot tudi vzpostavitev programskih omejitev in kontrolnih točk, ki skrbijo, da sistem deluje v okviru predvidenih nalog in pooblastil. Ko govorimo o varnosti, se ne pogovarjamo o stanju, ampak o postopku. Za aplikacijo, ki danes velja za varno, se lahko izkaže, da jutri več ni. Pojavijo se nova dognanja o sistemu, za katere se pred tem ni vedelo. S postopkom varnosti taka dognanja spremljamo in poskrbimo, da se morebitne pomanjkljivosti odpravijo v najkrajšem možnem času. Varnost temelji na logiki in vsak sistem je potrebno zavarovati glede na situacijo oziroma problemsko področje, ki ga sistem pokriva.

Varnost je velikega pomena predvsem pri poslovnih aplikacijah. Če nepooblaščene osebe dobijo dostop do podatkov, lahko nastane velika škoda. Vzemimo kot primer spletno bančništvo. Pri zavarovanju dostopa do podatkov in storitev, ki nam jih nudi spletna banka, moramo poskrbeti za preverjanje pristnosti uporabnika, mu omogočiti uporabo samo storitev, za katere ima pooblastila, in pa zavarovati prenos podatkov po omrežju. Za zavarovanje storitev in prenosa podatkov nam omenjena arhitektura ne nudi posebnih mehanizmov. Razviti so bili posebni splošni mehanizmi, ki jih lahko uporabimo za dopolnitev omenjenih arhitektur. Mehanizma, ki se uporabljata za zavarovanje podatkov med prenosom in pa sam dostop do podatkov, sta kriptografija ter dokazovanje pristnosti.

Pri varnosti računalniškega sistema pa moramo omeniti še en segment varnosti, ki lahko privede do nepooblaščenega dostopa do sistema. Socialni inženiring je pristop, ko poskuša nepooblaščena oseba dobiti podatke, potrebne za dostop do zavarovanega sistema od pooblaščene osebe s prevaro. Danes pogost primer so prevare za pridobitev pooblastil za dostop do podatkov spletnih bank. Neuki uporabniki lahko prevare ne prepoznajo in utrpijo finančno škodo. Za te napade so ranljivi tudi tehnično dobro zavarovani sistemi. Vzpostavljena morajo biti merila in mehanizmi, ki preprečujejo uporabnikom izdajo takšnih informacij. Eden takšnih mehanizmov je uporaba biometričnih podatkov za dostop do varovanih sistemov.

1.4 IZKORIŠČENOST VIROV

Vzdrževanje in delovanje računalniških sistemov postaja vedno dražje. Organizacije težijo k vedno večji izkoriščenosti sredstev, ki so jim na razpolago. V računalništvu lahko gledamo na izkoriščenost virov s stališča izkoriščenosti procesorske moči, zasedenosti pomnilnika in vseh ostalih komponent računalnika. Ker pa ni možno vedno optimalno izkoristiti vseh komponent, se je smiselno osredotočiti na tiste, ki predstavljajo največji strošek. Porazdeljena arhitektura omogoča boljšo izkoriščenost virom, saj lahko strojno opremo prilagodimo vsakemu modulu aplikacije posebej. Tako lahko modulom, ki opravljajo računsko intenzivno delo dodelimo hitrejši procesor in več delovnega pomnilnika, tistim, ki pa so namenjeni shranjevanju podatkov, namenimo večje in hitrejše magnetne diske. Lažje je tudi nadgrajevanje zmogljivosti, saj se nadgradi samo oprema preobremenjenih modulov. Module, ki ne potrebujejo veliko virov, je smiselno združiti na skupni strojni opremi, saj lahko pretirano povečanje količine strojne opreme privede do obratnega efekta in se stroški povečajo. Večja količina strojne opreme poleg cene nakupa prinese tudi večje obratovalne stroške.

V diplomskem delu bom predstavil tehnologije, ki so na voljo pri gradnji porazdeljenih aplikacij in jih primerjal s tehnologijo Windows Communication Foundation.

2 TEHNOLOGIJE ZA GRADNJO PORAZDELJENIH SISTEMOV

Pri gradnji porazdeljenih aplikacij imamo na voljo različne tehnologije. Nekatere med njimi so zelo nizko nivojske in zahtevajo podrobno poznavanje delovanja omrežij. Med te spadajo predvsem vtičnice in imenovane cevi. Težave, povezane z uporabo te tehnologije so v tem, da moramo sami zgraditi celoten višjenivojski protokol za komunikacijo, ki pa potem večinoma ni združljiv z nobeno drugo tehnologijo. Implementirajo se samo tiste podrobnosti komunikacije, ki jih potrebujemo in le redko je uporabljena arhitektura dovolj odprta za naknadno razširjanje. Sama izdelava teh mehanizmov pa nam vzame tudi veliko časa, ki bi ga lahko uporabili za izdelavo ostalih delov aplikacije. Še posebej se ta problem izraža pri preprostih aplikacijah, kjer je čas, potreben za izdelavo ostalih delov aplikacije, minimalen in se večino časa ukvarjamo z reševanjem problemov omenjenih mehanizmov za komunikacijo. Poleg teh mehanizmov poznamo še različne tehnologije, ki poskušajo poenostaviti gradnjo porazdeljenih aplikacij z abstrakcijo komunikacijskih mehanizmov. Poskušajo uvesti programske vmesnike, s katerimi na čim bolj enostaven način kličemo oddaljeno metodo oziroma funkcijo. Najbolj aktualne tehnologije so CORBA, Java RMI, .NET Remoting in spletne storitve. Skupno vsem tem tehnologijam je objektna usmerjenost in zato morajo vse te tehnologije na nek način rešiti probleme, ki se pojavijo kot posledica objektnega pristopa programiranja.

Objektom, ki se nahajajo na drugem računalniku, pravimo oddaljeni objekti. Glavni problem pri klicanju metod oddaljenega objekta je, kako do objekta pridemo na strani odjemalca oziroma kako narediti referenco[2] na tak objekt. Pri klicu metod je lahko parameter tudi objekt in zato je potrebno določiti, kako prenašamo objekte po vrednosti. Na strežniku se pojavijo problemi, povezani s sproščanjem pomnilnika. Potrebno je uvesti mehanizem, ki določa, katerih objektov ne uporablja nobeden odjemalec. Na strani odjemalca pa se pojavi problem, kaj narediti z referenco, če se strežnik preneha odzivati. Poleg reference na objekte je potrebno določiti, kako se objekti oziroma podatki prenašajo po omrežju. Tu nas najbolj zanima, kateri protokoli se uporabljajo, kako so podatki predstavljeni med prenosom in kako podatke preoblikujemo v obliko za prenos.

Kot je v nadaljevanju razvidno iz opisov posameznih tehnologij, so nekateri pristopi pri vseh tehnologijah enaki. Predvsem je podoben način, kako se dostopa do oddaljenih objektov, saj pri vseh nekako srečamo poseben posredniški objekt, ki ga uporabljamo lokalno in ta potem opravlja klice do oddaljenih objektov. Razlike pa se pojavijo predvsem pri podrobnostih posamezne izvedbe in kako se ti posredniki generirajo.

2.1 CORBA

CORBA[3] je standard, definiran s strani organizacije Object Management Group (OGM). Standard je implementiran s strani različnih ponudnikov programske opreme in v različnih programskih jezikih, kar omogoča veliko mero svobode pri implementaciji aplikacij. Nekatere implementacije niso popolne oziroma imajo dodane opcije, ki jih standard ne določa. To lahko predstavlja problem pri vključitvi obstoječih programskih rešitev, ki temeljijo na kateri drugi implementacij standarda CORBA.

Glavna prednost tehnologije CORBA je široka podpora na različnih sistemih in programskih jezikih. Na ta način se lahko povežejo skupaj komponente, ki so napisane s strani različnih razvijalcev in organizacij. To je še posebej uporabno pri dopolnjevanju oziroma razširitvi funkcionalnosti starejših sistemov, ki se še vedno veliko uporabljajo predvsem v različnih poslovnih okoljih.

Slika 2: Pri arhitekturi CORBA lahko med sabo komunicirajo različne tehnologije. Vmesnik odjemalca in vmesnik strežnika ne komunicirata neposredno med sabo, ampak je vmes še nivo, ki mu rečemo posrednik objektov. Ta skrbi za pravilno pretvorbo objektov v jezik IDL in pošiljanje med strežnikom in odjemalcem.

Podpora različnim jezikom pa prinese tudi nekatere omejitve, ki jih v primeru tehnologije s podporo enemu jeziku ne srečamo. Prenos podatkov oziroma objektov iz naslovnega prostora enega programskega jezika v drugi ni nikjer definiran. Problem je še večji, ker imamo opravka z več kot dvema jezikoma. Standard CORBA ne določa preslikav med različnimi programskimi jeziki, saj bi to pomenilo, da ni programsko neodvisen in bi bila implementacija v nekem novem programskem jeziku podrejena podpori v standardu. Da bi se rešil ta problem, je v standardu določen nek vmesni jezik oziroma IDL. Na ta način problemi, povezani z predstavitvijo podatkov, odpadejo. IDL se uporablja za definicijo vmesnikov do objektov. Objekti se pri tehnologiji CORBA prenašajo po referenci, v novejših različicah pa je podprto tudi prenašanje po vrednosti. Podpora je vpeljana nekoliko drugače kot pri ostalih tehnologijah. Pretvorba objektov v binarni tok na strani pošiljatelja objekta in rekonstrukcija istega objekta na strani prejemnika ni možna zaradi uporabe različnih programskih jezikov.

Da bi kljub temu lahko prenašali objekte po vrednosti, jezik IDL vsebuje možnost opisa stanja nekega objekta s standardnimi tipi, ki so definirani v jeziku IDL. Za uspešen prenos objekta po vrednosti moramo tako imeti implementacijo razreda na obeh straneh aplikacije in pa definicijo, kako se ti razredi preslikajo v IDL. V IDL so na voljo podatkovni tipi, kot so števila in tekst in pa tudi nekateri drugi kompleksnejši tipi, kot so tabele, sekvence in podatkovne strukture. Iz IDL se na strani odjemalca in strežnika generirajo posebni posredniki, ki se nato uporabijo za klice oddaljenih objektov. Za komunikacijo med objekti se uporablja poseben protokol IIOP. Podatki, ki se prenašajo, pa se preoblikujejo v poseben binarni format.

Pri sproščanju pomnilnika CORBA predvideva, da odjemalec obvesti strežnik, ko več ne potrebuje reference na objekt. Vsak objekt ima tudi števec, ki pove, kdaj ni več nobene reference nanj in takrat se objekt sprosti.

Razvoj aplikacije v tehnologiji CORBA se začne pri definiciji IDL vmesnikov za dostop do objektov. Če se kot parametri pri klicu metod objektov uporabljajo tudi drugi objekti, je te potrebno definirati v jeziku IDL z uporabo valuetype oznake.

```
module Primer
{
   valuetype Oseba
   {
     private string _ime;
     private string _sporocilo;
   };

   interface Pozdrav
   {
     Oseba LepPozdrav(in Oseba oseba);
   };
};
```

V naslednjem koraku se iz definicije IDL vmesnikov generira preslikava v željen programski jezik na strani odjemalca in strežnika. Če uporabimo preslikavo v programski jezik C#, se generirajo ustrezni vmesniki in abstraktni `valuetype` razredi. V zadnjem koraku je potrebno te vmesnike in abstraktne razrede implementirati v ustreznih razredih. Implementacija abstraktnega razreda je takšna.

```
namespace Primer
{
  public class OsebaImpl: Primer.Oseba
  {
    public OsebaImpl()
    {
    }

    public OsebaImpl( string name, string message )
    {
      _ime = ime;
      _sporocilo = sporocilo;
    }

    public  string ime
    {
```

```
        set { _ ime = value; }
        get { return _ ime; }
      }


      public  string sporocilo
      {
        set { _ sporocilo = value; }
        get { return _sporocilo; }
      }
    }
  }
```

Ko imamo implementirane vse potreben razrede na odjemalcu in strežniku, lahko vzpostavimo komunikacijo. Potrebno je povedati, da se lahko implementacija objektov na odjemalcu in strežniku razlikuje. Potrebno je samo, da oba implementirata isti vmesnik.

```
Primer.Pozdrav oPozdrav
   = Primer.PozdravHelper.narrow( oPozdravObj );

Primer.OsebaImpl oOseba = new Primer.OsebaImpl();

oOseba.name = "Janez";
oOseba.message = "Vreme je lepo!";

Primer.Oseba oPovratniPozdrav = oPozdrav.LepPozdrav ( oOseba );
```

Zgornji primer kode se začne z uporabo metode `narrow`. Ta del koda nam vrne referenco na oddaljen objekt. Objekt oseba pa se prenaša po vrednosti.

2.2 JAVA RMI

Tehnologija RMI[4] temelji na okolju Java in je testno povezana z njim. Ker je RMI Java specifičen je mogoče pisati aplikacije izključno za to okolje, ker pa je Java okolje mogoče pognati na različnih sistemih, ta omejitev ni tako pomembna. Cilj pri nastanku te tehnologije je bil omogočiti razvijalcem gradnjo porazdeljenih aplikacij na podoben način kot ostale aplikacije. Osnovni princip RMI okolja je, da je definicija neke funkcionalnosti in implementacija te iste funkcionalnosti različna stvar. Definicija in implementacija funkcionalnosti sta lahko del različnih Java navideznih strojev. To se lepo ujema z odjemalec strežnik modelom, kjer je za odjemalca pomembna definicija funkcionalnost, strežnik pa jo lahko implementira. Definicijo določa vmesnik, implementacijo pa v RMI določata dva razreda. Prvi je razred na strežniku, ki implementira funkcionalnost, drugi pa je posrednik, ki posreduje klice odjemalca.

Okolje RMI omogoča podajanje primitivnih tipov in objektov kot parametre klicev metod oddaljenih objektov. Vsi parametri pri klicu metode se prenašajo po vrednosti, ker je nekoliko drugače kot pri klicu lokalne metode, ko se objekti prenašajo po referenci. Za prenos objektov preko omrežja pa jih je potrebno preoblikovati v neko obliko, primerno za prenos. V ta namen se uporablja serializacija, ki preoblikuje objekte v binarno obliko. Tretja vrsta parametrov pa

so oddaljeni objekti. Klic metode lahko vrača tak parameter in v tem primeru se ne vrne objekt, ampak posrednik do takega objekta.

Še ena pomembna prednost RMI tehnologije je sposobnost prenosa implementacije iz oddaljene lokacije. Implementacija nekega vmesnika se lahko prenese iz strežnika. Ta lastnost je dobra, saj se v poslovnem okolju velikokrat spreminjajo podrobnosti implementacije in v tem primeru ni potrebno ročno posodobiti vseh odjemalcev ampak se spremembe samodejno prenesejo iz strežnika. Na ta način je strežnik razbremenjen izvajanja določenih nalog, odjemalec pa ima vedno najnovejšo implementacijo brez potrebe po ponovnem prevajanju. Drug primer pa je, ko se implementacija prenaša iz odjemalca na strežnik. To je primer računskega strežnika, ki v izračun sprejme tudi implementacijo izračuna.

Slika 3: RMI je namenjen komunikaciji aplikacij v Java okolju.

Pri tehnologiji RMI nam tako kot pri vseh naslednjih ni potrebno definirati posebnega jezika za prenos. Prvi korak implementacije je definicija vmesnika objekta, ki ga bo klical odjemalec.

```
import java.rmi.*;

public interface Primer extends Remote
{
  public Oseba LepPozdrav (Oseba oseba) throws RemoteException;
}
```

Ko definiramo vmesnik, ga moramo še implementirati. Strežnik mora imeti definicijo vmesnika in njegovo implementacijo, odjemalec pa potrebuje samo vmesnik. Ko imamo implementacijo strežnika, je klic s strani odjemalca preprost.

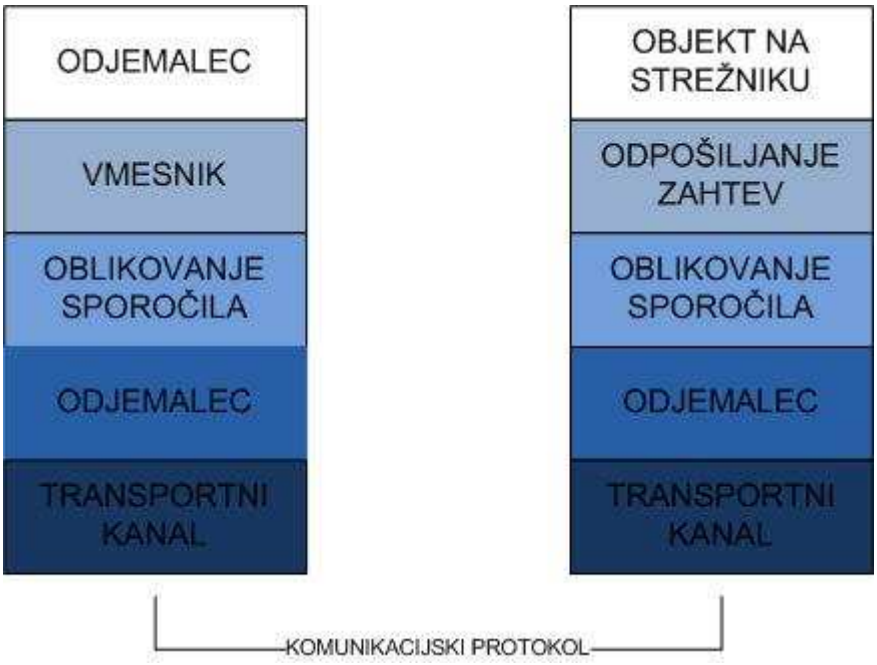
```
PrimerImp oddaljenObjek
```

```
        = (PrimerImp)Naming.lookup(mrezniNaslovStreznika);

    oddaljenObjetk.LepPozdrav(oseba);
```

2.3 .NET REMOTING

.NET Remoting[5] je Microsoftova infrastruktura, ki razvijalcu ponuja veliko različnih razredov za gradnjo porazdeljenih aplikacij. Tehnologija je omejena na .NET okolje in s tem jezike, ki so tu podprti. Vpeljuje veliko mero abstrakcije in odstrani kompleksnost pri izdelavi in upravljanju oddaljenih objektov. Klic metode oddaljenega objekta je skoraj identična klicu lokalne metode. Ko odjemalec ustvari oddaljen objekt, dobi posrednika do instance objekta na strežniku. Vsak klic metode posrednika je posredovan strežniku in rezultat je posredovan nazaj odjemalcu. Iz perspektive odjemalca takšen klic ni drugačen od klica lokalne metode.



Slika 4: Arhitektura .NET Remoting je predhodnik tehnologije WCF. ŽE pri tej arhitekturi je videti neko mero modularnosti, ki se pri WCF še poveča.

Za izgradnjo aplikacije z dvema komponentama v različnih naslovnih prostorih, ki uporabljata komunikacijski model .NET Remotinga, mora razvijalec narediti samo tri stvari. Skonstruirati mora objekt, ki se ga lahko kliče iz drugega procesa oziroma drugega naslovnega prostora, gostiteljsko aplikacijo takšnega objekta, ki posluša za klice objekta, in pa odjemalca, ki izvršuje klice takšnega objekta. Tudi pri gradnji kompleksnih strežnik odjemalec sistemov lahko uporabimo te osnovne principe .NET Remoting modela.

Objekti, ki jih ustvari odjemalec na strežniku, so na odjemalcu dostopni preko reference. Pri klicu metod oddaljenega objekta se parametri in rezultat privzeto prenašajo po vrednosti. Lahko pa eksplicitno zahtevamo prenos po referenci. Objekti, ki se prenašajo po referenci, morajo implementirati poseben vmesnik. Ravno tako morajo biti objekti, ki se prenašajo po vrednosti, označeni s posebnim atributom, in pa opcijsko lahko implementirajo poseben vmesnik, ki določa, kako se objekt serializira. Objekte, ki ne zadostujejo enemu od teh pogojev, ni možno uporabljati pri komunikaciji med dvema procesoma.

Za prenos podatkov lahko v .NET Remoting okolju uporabimo tri protokole, in sicer HTTP, TCP in SMTP. Za pretvorbo podatkov v obliko, primerno za prenos, se pri HTTP protokolu uporablja XML zapis, pri ostalih dveh pa se uporablja kompaktnejša binarna oblika zapisa.

Implementacija preproste odjemalec strežnik aplikacije v tehnologiji .NET Remoting je enostavna. Definirati moramo razred, ki ga bo klical odjemalec.

```
public class Primer : MarshalByRefObject
  {
   private Oseba osebaKiOdgovaraj;

   public SampleObject()
   {
   }

   public Oseba LepPozdrav(Oseba oseba)
   {
     return osebaKiOdgovaraj;
   }
  }
```

Na strani odjemalca lahko potem takšen objekt oziroma njegove metode kličemo na naslednji način.

```
Primer objektPrimer = (Primer)Activator.GetObject(
         typeof(Primer),
         "naslov_storitve_ki_ponuja_objekt");

objektPrimer.LepPozdrav(oseba);
```

2.4 SPLETNE STORITVE

Spletne storitve[6] je standard, ki določa standarden način, kako naj programske komponente na različnih sistemih komunicirajo med sabo. Ne določa, kako naj se te storitve implementirajo, in ne določa nobenih omejitev, kako naj se povezujejo. Zgrajen je na osnovi odprtih standardov in na ta način je omogočena komunikacija med različnimi sistemi. Standardi kot so XML, SOAP in WSDL in IP protokol so osnova. Spletna storitev je samo vmesnik, ki je dostopen preko računalniške mreže oziroma interneta in se izvaja na sistemu, ki gosti zahtevano storitev.

XML ali razširljiv označevalni jezik je jezik za izmenjavo strukturiranih podatkov. Elemente jezika določa uporabnik sam in tako omogoča definicijo strukture jezika po meri podatkov. Pri spletnih storitvah se XML uporablja za predstavitev podatkov, ki se pošiljajo med odjemalcem in strežnikom. Ker si pri XML razvijalec sam določi oznake za podatke, mora tudi poskrbeti, da se podatki iz podatkovnih struktur uporabljenega jezika pravilno zapišejo in seveda ponovno preberejo iz takšne oblike. Standard spletnih storitev tudi določa nekatere oznake za splošne podatkovne tipe, ki se uporabljajo v programskih jezikih.

SOAP je protokol za izmenjavo sporočil, ki temeljijo na XML jeziku in se uporablja tudi v spletnih storitvah. Prednost tega protokola je, da omogoča uporabo različnih transportnih protokolov. SOAP določa, kaj je v sporočilu in kako se ga obdela, pravila kodiranja podatkov in način, kako podati klice oddaljenih procedur.

Za opis spletnih storitev se uporablja WSDL jezik in tudi temelji na XML jeziku. V tem jeziku so storitve opisane kot zbirka komunikacijskih končnih točk, ki so sposobne izmenjevati sporočila. Odjemalec, ki želi uporabljati funkcionalnost neke spletne storitve, lahko preko WSDL ugotovi, katere funkcije storitev ponuja. Odjemalec lahko nato uporabi SOAP sporočila za klicanje posamezne funkcije.

Pri spletnih storitvah se tako XML uporablja za označevanje podatkov, ti podatki se prenašajo z uporabo SOAP protokola, WSDL pa služi za opisovanje storitev.

Za razliko od ostalih predstavljenih tehnologij spletne storitve ne nudijo določenih mehanizmov za delo z objekti. Standard spletnih storitve ne določa ničesar o načinu reference na določen oddaljen objekt. Za referenco in nekatere druge podrobnosti moramo poskrbeti sami.

3 WINDOWS COMMUNICATION FOUNDATION

Windows Communication Foundation[7] je enoten programski model za gradnjo porazdeljenih sistemov na Microsoftovi platformi. Združuje prejšnje Microsoftove tehnologije, ki vsaka predstavlja poseben model, kot so sporočilna vrsta, spletne storitve, delo z oddaljenimi objekti. Z združitvijo teh tehnologij pod enoten programski model se je povečala produktivnost pri izgradnji porazdeljenih aplikacij, saj razvijalcu ni potrebno vedeti veliko različnih tehničnih podrobnosti glede posamezne tehnologije. Aplikacije, napisane v WCF, so tako lahko združljive s starejšimi aplikacijami, napisanimi v katerikoli Microsoftovi tehnologiji. Ker pa se danes vedno bolj pojavlja potreba po povezovanju nehomogenih sistemov, je v WCF poudarek na povezljivosti po standardu spletnih storitev. WCF podira množico standardov spletnih storitev, ki govorijo o varnosti, transakcijah in podobno. Na ta način je omogočena povezljivost z aplikacijami, ki podpirajo standarde spletnih storitev, ki danes vedno bolj pridobivajo na veljavi.

Ker je WCF tehnologija, ki podpira različne načine komunikacije, so se morali njeni razvijalci potruditi pri njeni arhitekturi. Potrebno je bilo ločiti komunikacijo na posamezne logične dele, v teh delih podpreti obstoječe načine komunikacije in omogočiti morebitne razširitve posameznih delov. Tako je nastal model, katerega lahko brez problema razširimo z novim transportnim protokolom, načinom predstavitve podatkov med prenosom, varnostnim mehanizmom in podobno.

Kljub vsem različnim načinom, ki jih WCF ponuja, pa je programski model enak ne glede na vse podrobnosti komunikacije. Vsako storitev oziroma funkcionalnost, ki jo neka aplikacija ponuja, definiramo na enak način. Vse ostale podrobnosti glede komunikacije nato določimo preko atributov in različnih konfiguracij, ki jih lahko podamo preko posebne datoteke ali pa programsko. Tak način gradnje aplikacije omogoča hitro spreminjanje načina komunikacije, ne da bi spreminjali poslovno logiko.

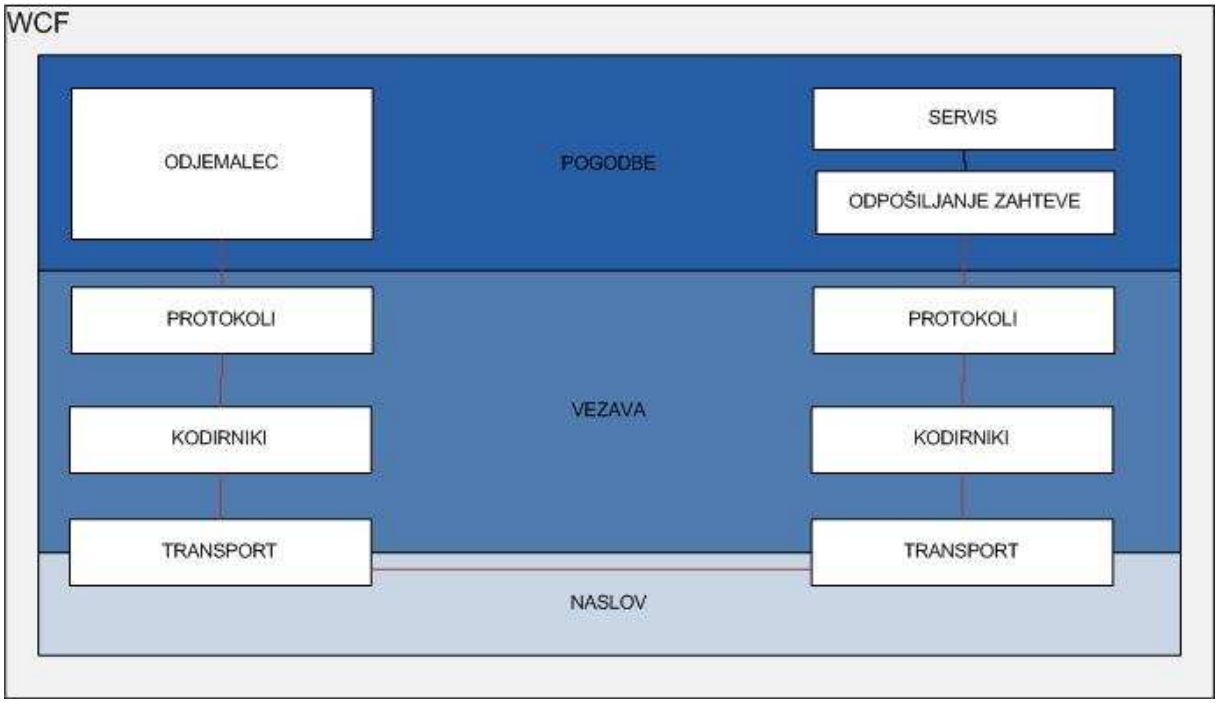
Način, na katerega je WCF zgrajen, omogoča razvijalcu tudi poseben način abstrakcije ob podrobnosti komunikacije. Poseben v tem smislu, da si ga lahko izbere razvijalec sam. Lahko gradimo porazdeljene aplikacije brez znanja o transportnih protokolih in njihovem delovanju, v tem primeru uporabimo že vgrajene mehanizme, lahko pa uvedemo nov protokol. Iz tega razloga lahko rečemo, da je WCF primeren za uporabo pri preprostih nezahtevnih aplikacijah pa vse do zahtevnih varnostno in izvršilno kritičnih aplikacij.

V nadaljevanju bom opisal in predstavil WCF arhitekturo in kako se posamezna rešitev primerja s tehnologijami, opisanimi v prejšnjem razdelku. Poudarek pri tej predstavitvi bom naredil na varnostnih mehanizmih, ki jih WCF podpira.

3.1 ARHITEKTURA WCF

WCF komunikacija temelji na prenašanju sporočil. Razvijalcu ni potrebno vedeti ničesar o obliki sporočila. Potrebno je, da samo definira servisno pogodbo, na podlagi katere se potem generirajo sporočila. Vse klice storitev oziroma oddaljenih procedur odjemalec opravi preko posrednika, ki je lokalna predstavitev vmesnika storitve. Ta način z uporabo posrednika za abstrakcijo komunikacijskih podrobnosti od razvijalca je prisoten tudi pri ostalih tehnologijah. Razlika je v tem, da uporabljata tehnologiji CORBA in Java RMI princip oddaljenega objekta, WCF pa uporablja princip spletne storitve in pošiljanje sporočil med njima. Ko odjemalec pokliče metodo vmesnika, ta preoblikuje klicni zapis v sporočilo. To se nato pošlje preko več kanalov, ki skrbijo za varnost, kodiranje sporočila v primerno obliko, zanesljiv prenos in podobno. Zadnji v verigi kanalov je transportni kanal, ki poskrbi za prenos sporočila do gostitelja WCF storitve. Na strani gostitelja gre sporočilo ravno tako skozi serijo kanalov, ki opravljajo ravno obratno funkcijo kot na strani odjemalca. Zadnji v seriji kanalov na gostitelju pa je odpravitelj, pretvori sporočilo v klicni zapis in izvede sam klic. Ko se klicana storitev izvede do konca, se kontrola vrne odpravitelju. Ta generira povratno sporočilo, ki se po istem postopku vrne odjemalcu.

Največja prednost takšne arhitekture je možnost dodajanja lastnih kanalov za katerokoli fazo komunikacije. Kateri kanali se uporabijo, se definira z vezavo. Standardne vezave, ki so na voljo v WCF, so implementirane na enak način kot vezave, ki jih definiramo sami. Pri ostalih tehnologijah je arhitektura veliko bolj zaprta in nimamo možnosti dodajanja novih oblik komunikacije. Vendar pa lahko na podlagi napisanega sklepamo na možnost podpore tehnologijam, kot so CORBA in Java RMI s strani WCF na enak način, kot je vgrajena podpora za .NET Remoting.



Slika 5: Arhitektura tehnologije WCF

Kot pri vseh primerljivih tehnologija je potrebno sporočilo preoblikovati v obliko, primerno za prenos po omrežju. WCF privzeto sporočilo serializira v obliko, ki je definirana s SOAP standardom. Na ta način lahko vsaka aplikacija, ki pozna ta standard, prebere sporočilo. Ni pa izključena možnost drugih oblik serializacije. Napišemo lahko svojo obliko, saj je ta v primeru, da hočemo podpreti komunikacijo s katerokoli drugo tehnologijo, nujno potrebna. Za komunikacijo po standardu spletnih storitev pa je v WCF že vse na voljo. V primeru, ko na eni strani komunikacije nimamo WCF okolje, mora biti zagotovljena podpora enakim spletnim standardom, kot so uporabljeni na WCF strani komunikacije. Trenutno WCF podpira veliko standardov in zato mora biti razvijalec nehomogenih sistemov pazljiv, da uporabi tiste, ki so podprti tudi na drugi strani komunikacije.

3.2 KONČNA TOČKA

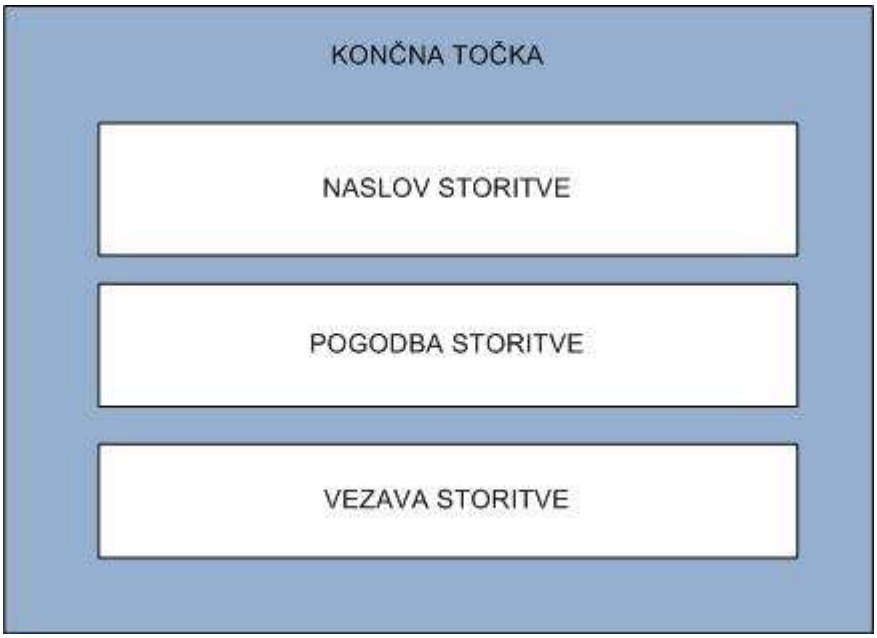
Vsaka WCF storitev je aplikacija, ki ponuja eno ali več končnih točk. Logično gledano je končna točka vmesnik storitve, preko katerega dostopamo do storitve. Za uspešno uporabo neke storitve moramo vedeti, kje se storitev nahaja, kako s storitvijo komuniciramo in pa, kaj nam storitev ponuja. Končna točka določa natančno te tri stvari. Vsaka storitev ima vsaj eno končno točko, lahko pa jih ima tudi več. Z uporabo več končnih točk lahko dosežemo uporabo različnih načinov komunikacije za različne odjemalce. Kot primer lahko vzamemo storitev, ki določeno storitev ponuja odjemalcem na nekem notranjem zavarovanem omrežju in pa odjemalcem na internetu. V primeru odjemalcev na lokalnem omrežju ni potrebna uporaba varnostnih mehanizmov. Če pa odjemalec dostopa do storitve preko interneta, se uporabijo napredni mehanizmi, ki poskrbijo za varno komunikacijo. Bistveno pri vsem tem je, da je implementacija poslovne logike, ki se uporablja pri obeh končnih točkah enaka.

Lokacijo storitve določa naslov. Poleg samega naslova pa je podan tudi transportni protokol, ki se uporablja za prenos podatkov po omrežju, in pa nekatere opcijske stvari, kot so vrata, cev in podobno. Vsem podatkom, ki se nahajajo v naslovu, rečemo tudi transportna shema. WCF podpira nekaj standardnih shem, kot so HTTP, TCP, omrežje vrstnikov in podobne.

Kaj nam storitev ponuja, določajo pogodbe. V WCF poznamo več vrst pogodb, ki se nanašajo na to, kakšne funkcije oziroma metode nam storitev ponuja, kakšni podatki se prenašajo, vrste napak oziroma izjem in pa pogodbe, povezane s sporočili. Storitveno pogodbo lahko primerjamo z referenco na oddaljen objekt pri tehnologih, kot so Java RMI in CORBA. Pri teh tehnologijah je namreč posrednik, preko katerega odjemalec komunicira s strežnikom referenca na tak objekt. WCF takšnega principa ne pozna in deluje na ideji storitev, ki si izmenjujejo sporočila.

Vezava je tretja stvar, ki jo končna točka vsebuje. Z vezavo definiramo, kako poteka komunikacija med odjemalcem in strežnikom. Ko govorimo o načinu komunikacije, imamo v mislih več aspektov. Odločiti se moramo, ali je komunikacija sinhrona ali asinhrona, kakšen je vrstni red obdelave prispelih sporočil, ali sporočila dostavimo takoj ali se shranjujejo v

vrsto in izvajajo v intervalih, kakšen je transportni protokol, kakšne varnostne mehanizme uporabljamo in še bi lahko naštevali. Kljub temu da so nekatere kombinacije medsebojno izključujoče, je število možnih kombinacij zelo veliko. V WCF lahko grupiramo odločitve o posameznem delu komunikacije v vezave. Da bi bilo delo še lažje, WCF ponuja že veliko standardnih vezav. Od teh so nekatere primerne za komunikacijo s starejšimi aplikacijami, saj podpirajo prejšnje Microsoftove tehnologije, nekatere so namenjene za komunikacijo izključno med WCF komponentami, tretje pa so namenjene za komunikacijo po standardih spletnih storitev in tako omogočajo komunikacijo z ostalimi platformami. Vse skupaj pa lahko razširimo tudi s svojimi vezavami, ki lahko uporabljajo obstoječe rešitve posameznega aspekta komunikacije ali pa implementiramo svoje rešitve. Vezave so neodvisne od posamezne implementacije poslovne logike in zato lahko za posamezno storitev uporabimo v več različnih vezavah.



Slika 6: Končno točko sestavljajo naslov, pogodba in pa vezava.

Končna točka z naslovom, pogodbo in vezavo opisuje, kje, kaj in na kakšen način se dostopa do storitve. Končne točke, ki jih želimo imeti pri neki storitvi, določimo v nastavitveni datoteki. .NET okolje poskrbi za inicializacijo končne točke. Na podlagi informacij, ki so potem dostopne preko končne točke, se generira posrednik na strani odjemalca. Odjemalec se lahko generira samodejno z uporabo posebnega orodja ali pa se ga napiše ročno.

```
<endpoint address="http://localhost:1000/Storitev/"
          binding="wsHttpBinding"
          contract="IStoritev" />
```

3.3 UPRAVLJANJE Z INSTANCAMI

WCF ima za upravljanje z instancami več možnosti. Vsak klic lahko dobi svojo instanco storitve, lahko je instanca vezana na odjemalca ali pa je ena instanca za vse klice. Pri odločitvi kateri način bomo izbrali, moramo biti pozorni na to, koliko virov bo posamezna instanca zavzela. Glede na izbiro načina se lahko pojavi problem z porabo pomnilnika ali pa z odzivnostjo storitve. Če dobi vsak klic svojo instanco, se pri velikem številu klicev in storitvi, ki ima veliko virov, hitro pojavi problem s porabo pomnilnika. Pri načinu, ko je ena storitev odgovorna za vse klice, pa morajo posamezni klici čakati, da se predhodni izvrši do konca. Pri tem lahko pride do slabe odzivnosti storitve ali celo do izgube posameznih klicev zaradi predolgega čakanja. V tehnologijah Java RMI in CORBA je ta problem nekoliko bolj odvisen od implementacije same poslovne logike. V RMI tehnologiji nove instance oddaljenih objektov na strežniku razvijalec kreira podobno kot lokalne objekte. V WCF je to stvar konfiguracije in po potrebi lahko to hitro spremenimo. Pri RMI je potrebno spremeniti logiko klicev oddaljenih objektov.

Pri gradnji aplikacij, združljivih s tehnologijami, ki temeljijo na objektih, je mehanizem upravljanje z instancami zelo pomemben. Ko se v takšni tehnologiji ustvari nov oddaljen objekt, je referenca na tak objekt veljavna, dokler ga ne uničimo. Če imamo na drugem koncu neko WCF aplikacijo, bo takšna referenca kazala na WCF storitev. Z upravljanjem instanc je potrebno poskrbeti, da odjemalec za vsak nov oddaljen objekt dobi svojo sejo na strani WCF storitve. Na ta način lahko v tehnologiji WCF simuliramo oddaljen objekt za uporabo v aplikacijah, pisanih v drugih tehnologijah.

3.4 GOSTOVANJE PROCESA

Da postane WCF storitev aktivna, mora gostovati znotraj izvajajočega se procesa, ki mu rečemo gostiteljski proces. Tak proces je lahko kateri koli Windows proces, ki podpira izvajanje .NET kode. Načini gostovanja v WCF se razlikujejo predvsem po tem, kako posamezno okolje upravlja s storitvijo. Pomembno pri tem je predvsem, kako in kdaj se storitev aktivira, kako se upravlja z življenjsko dobo storitve in kakšne varnostne mehanizme okolje nudi.

Storitev lahko gosti v samostojni aplikaciji, ki jo napišemo sami. Prednost tega načina je prilagodljivost. Uporaba je primerna predvsem v času razvoja storitve, predvsem zaradi enostavnosti uporabe in implementacije. Slabost pa je predvsem v pomanjkanju mehanizmov, ki bi skrbeli za takšen proces. Nekoliko naprednejši način je uporaba gostovanja znotraj procesa operacijskega sistema. Tak način gostovanja nam nudi vse prednosti, ki jih imajo ostali procesi operacijskega sistema, kot sta spremljanje in upravljanje z življenjsko dobo procesa. Ker WCF temelji na spletnih storitvah, je mogoče storitev gostovati tudi znotraj spletnega strežnika IIS. Pri tem načinu je življenjska doba in aktivacija storitve upravljana s strani spletnega strežnika. Pomanjkljivost tega načina, predvsem pri starejših različicah spletnega strežnika, je v uporabi protokola za komunikacijo, ki je omejena na HTTP.

Microsoft pa nudi v novejših operacijskih sistemih način gostovanja, ki omogoča uporabo vseh modelov komunikacije, aktivacije storitve in komunikacijskih protokolov, podprtih s strani WCF. Ta način je tudi najbolj primeren za gostovanje WCF storitev.

V ostalih tehnologijah veliko teh prednosti nimamo. Predvsem je tu pomembno povedati, da WCF nudi toliko možnosti, ker je tesno povezan z operacijskim sistemom. Java RMI je vezana na okolje Java. Tudi ta ima svoje načine gostovanja, ki pa so nekoliko bolj standardni in neodvisno od operacijskega sistema. V tehnologiji CORBA pa okolje za gostovanje ni definirano in ga je potrebno zagotoviti posebej.

3.5 VARNOST

Pri zavarovanju porazdeljenega sistema se pojavijo problemi, ki jih moramo rešiti, da bi bila komunikacija varna. S tehničnega stališča so to predvsem medsebojna overitev pristnosti odjemalca in strežnika, določiti pooblastila overjene strani pri uporabi virov in pa zavarovati sporočilo med prenosom tako, da ga ne more nihče spremeniti ali prebrati. V kolikšni meri bomo zavarovali oziroma rešili vse navedene probleme, je odvisno tudi od okolja in problemskega področja, ki ga aplikacija rešuje.

Različne tehnologije za gradnjo porazdeljenih aplikacij nudijo različne mehanizme in načine, kako zavarovati porazdeljene sisteme. Nekatere nudijo zelo malo in je zato za varnost potrebno poskrbeti z drugimi metodami.

WCF tudi za varnost ponuja široko paleto možnosti. Vsi vidiki varnosti so nastavljivi v okviru programskega modela in zato lahko na enostaven način spremenimo varnostne mehanizme, ki so uporabljeni.

3.5.1 PREVERJANJE PRISTNOSTI

S preverjanjem pristnosti se ugotovi, ali je sporočilo dejansko poslal tisti, ki pravi, da ga je poslal. Vsaka aplikacija, ki poskuša zagotoviti vsaj osnovno mero varnosti, mora imeti neke mehanizme za preverjanje pristnosti uporabnika oziroma programskega modula, ki pošlje neko zahtevo strežniku. Najosnovnejši mehanizem je uporaba uporabniškega imena in gesla. Slabost pri tem pristopu je predvsem izpostavljenost različnim načinom vdiranja v zavarovan sistem s silo. Uporabniki takšnih sistemov imajo tipično možnost izbire gesel, ki pa so velikokrat prekratka in jih lahko ugotovimo, če poznamo nekatere osebne podrobnosti uporabnika. Zaželeno je, da so gesla daljša nelogična zaporedja dovoljenih znakov. To pomanjkljivost se lahko nekoliko omili s pravili, katerim mora izbrano geslo ustrezati. Največkrat se uporablja pravilo dolžine gesla in pa obvezna uporaba nekaterih znakov.

Varnejši način overjanja je z uporabo digitalnih certifikatov. X.509 je standard, ki se danes uporablja pri implementaciji varnostnih rešitev na podlagi certifikatov. Vsi podatki, potrebni za overitve klica, so prisotni v certifikatu. Tehnologija certifikatov je osnovana na paru

ključev. Kar zaklenemo z enim ključem, lahko odklenemo samo z drugim. Enemu ključu rečemo privatni in je poznan samo nam, drugi pa je javni in ga dajemo drugim, da zavarujejo podatke, ki so namenjeni nam. Pri overjanju pristnosti pa sodeluje še izdajatelj certifikata. Ta na podlagi privatnega ključa in poslanega javnega ključa preveri pristnost. Prednost tega načina je predvsem uporaba kriptografije. Stopnja zaščite je odvisna od velikosti ključa. Ta model je dober, ker uporabnik nima vpliva na velikost in nastavitve glede ključev, ampak neka organizacija, ki poskrbi za zadostno varnost. Mora pa uporabnik poskrbeti za varno shranjevanje ključev.

V WCF je možno te standardne mehanizme uporabljati na zelo enostaven način. Tudi v tehnologijah Java RMI in CORBA lahko uporabljamo te mehanizme, vendar njihova uporaba oziroma integracija v aplikacijo ni tako preprosta kot pri tehnologiji WCF. Poleg tega pa WCF nudi še nekatere druge mehanizme, ki pa so vezani na Microsoftovo okolje in zato njihova uporaba ni podprta s strani drugih tehnologij.

3.5.2 PREVERJANJE POOBLASTIL

Avtorizacija oziroma preverjanje pooblastil je postopek odločanja, ali ima nekdo pravico dostopati do nekega podatka oziroma storitve. Ali povedano drugače, samo pooblaščene osebe imajo pravico dostopati do podatkov in storitev. Pri preverjanju pristnosti se lahko ponovno uporabijo podatki o uporabniku, kot so uporabniško ime in certifikati. Lahko pa se uporabljajo tudi drugi podatki posameznika, na podlagi katerih se lahko odloči, kaj uporabnik sme in kaj ne. Taki podatki so lahko starost uporabnika, elektronski naslov in drugi podatki, ki so vezani na uporabnika. Na podlagi teh podatkov se nato odloča o odobritvi dostopa do podatkov. V primeru starosti lahko tako omejimo dostop do določenih vsebin mladoletnim osebam ali kaj podobnega.

3.5.3 ZAŠČITA PODATKOV MED PRENOSOM

Preverjanje pristnosti in pooblastil bi bilo brez pomena, če bi lahko nekdo med prenosom videl vsebino sporočila ali ga celo spremenil. Mehanizem za varovanje integritete in zasebnosti sporočila je kriptografija. Podatke lahko zaščitimo na nivoju sporočila ali na nivoju transportnega protokola.

Zaščita na nivoju transportnega protokola deluje hitrejše in je zato primerna za uporabo pri sistemih, ki so bolj obremenjeni oziroma je komunikacije veliko. Ta način je tudi bolj podprt s strani različnih tehnologij in sistemov. WCF podpira SSL protokol, ki je podprt tudi s strani Java RMI, .NET Remoting in CORBA tehnologije. Odjemalec in strežnik se dogovorita, kateri kriptografski algoritem bosta uporabljala in pa izmenjata si javne ključe. Na podlagi teh podatkov se vzpostavi varna komunikacije med njima. SSL lahko uporablja za kriptiranje različne algoritme. Vedno se uporabi najboljši algoritem, ki ga poznata odjemalec in strežnik.

Z uporabo zaščite na nivoju transportnega protokola je zagotovljena integriteta in zasebnost podatkov med odjemalcem in strežnikom.

Pri zaščiti na nivoju sporočila se uporablja kriptiranje na nivoju sporočila. Zaščita je zagotovljena tudi prek nezaščitenih protokolov. Takšen način traja dlje in je zato primeren za storitve, kjer komunikacija ni tako intenzivna. V WCF zaščita na tem nivoju temelji na WS-Security in WS-SecureConversation standardih spletnih storitev. Standarda govorita o tem, kako SOAP sporočilo digitalno podpisati in uporabiti kriptografijo za prenos podatkov. Ker ta način komunikacije deluje po standardu spletnih storitev, je zagotovljena kompatibilnost z ostalimi platformami, ki implementirajo ta standard. Tehnologiji Java RMI in CORBA tega načina ne podpirata.

WCF podpira tudi mešanico obeh načinov. Tu se uporabiti dobre lastnosti obeh. Četrta možnost pa je uporaba obeh mehanizmov.

3.5.4 VARNOST SKOZI PROGRAMSKI MODEL WCF

Dosedanja predstavitev govori samo o možnostih, ki nam jih WCF ponuja za vzpostavitev varne komunikacije in kako je v drugih okoljih. Razvidno je, da tudi druge tehnologije nudijo standardne mehanizme za zaščito, kot so certifikati in kriptografija. Področje, kjer ima WCF prednost, je podpora standardom spletnih storitev. Vse te možnosti pa je potrebno tudi na nek način uporabiti v implementaciji storitve. Ravno v tem pogledu pa ima WCF prednost, saj model na zelo enostaven način omogoča definicijo varnosti, ki je neodvisna od poslovne logike aplikacije.

Pri gradnji varne porazdeljene aplikacije se moramo najprej odločiti, kakšne so varnostne zahteve na posameznem področju varnosti. Ko imamo izbrane vse aspekte, se varnost nastavi s konfiguracijo storitve.

Vseh kombinacij pri uporabi različnih mehanizmov za zavarovanje komunikacije je veliko, vendar pa se nekateri med seboj izključujejo. Izbiro posameznih podrobnosti nam omeji že izbira vezave, ki določa transportni protokol in kodiranje sporočil. WCF ima na voljo standardne vezave, ki imajo nastavljeno varnost glede na namen uporabe posamezne vezave. Ena takšnih vezav, ki so privzeto varne, je `NetTcpBinding`. Uporablja zaščito na nivoju transportnega protokola, TCP za prenos sporočil, binarni format za kodiranje sporočil in uporabnike operacijskega sistema Okna za preverjanje pristnosti. Na podlagi teh podrobnosti lahko sklepamo, da je ta vezava namenjena aplikacijam, pri katerih sta odjemalec in strežnik zgrajena v tehnologiji WCF. Ta ugotovitev je povezana predvsem z binarnim formatom za predstavitev sporočila med prenosom in pa uporabo uporabnikov operacijskega sistema Okna pri preverjanju pristnosti. Sklepamo lahko tudi, da je namenjena uporabi v lokalnem omrežju oziroma intranetu. Predvsem zato, ker se kot transportni protokol uporablja TCP in pa binarni format za sporočila. `NetTcpBinding` ima veliko podrobnosti komunikacije, kot so zagotavljanje zanesljivega prenosa sporočil ali pa podpora različnim standardom spletnih

storitev, izključenih. To pomaga predvsem pri hitrejšemu delovanju komunikacije, ker je še en pokazatelj čemu je ta vezava namenjena.

Privzete nastavitve vezave lahko spreminjamo in tako komunikacijo prilagodimo svojim potrebam. Vezava `NetTcpBinding` pri zaščiti na nivoju transportnega protokola ne dopušča uporabe uporabniškega imena in gesla za preverjanje pristnosti. Če bi hoteli ta način kljub temu uporabljati, lahko uporabimo katero koli drugo vezavo ali pa uporabimo zaščito na nivoju sporočila. Za takšno spremembo je potrebno nastavitve dopolniti z naslednjimi vrsticami kode.

```
<bindings>
  <netTcpBinding>
    <binding name="netTcp">
      <security mode="Message">
        <message clientCredentialType="UserName" />
      </security>
    </binding>
  </netTcpBinding>
</bindings>
```

Poleg teh nastavitev lahko nastavljamo tudi, kateri algoritmi se uporabljajo za kriptografijo, podajanje javnega ključa storitve in podobno. Vse kombinacije niso dovoljene. Več svobode imamo, če je izbran način zaščite na nivoju sporočila. V tem primeru so omejitve posledica standardov spletnih storitev, ki pri posamezni vezavi skrbijo za združljivost. Če vzamemo kot primer `BasicHttpBinding` vezavo, vidimo, da so tu omejitve vezane predvsem na dejstvo, da je namenjena povezovanju aplikacij, temelječih na WS-I Basic standardu. To je najosnovnejši standard spletnih storitev, ki je podprt v večjem številu različnih tehnologij. S to vezavo lahko tako zagotovimo povezljivost širokemu krogu starejših odjemalcev, pri čemer pa izgubimo možnost uporabe nekaterih naprednih mehanizmov. Pri vezavah z zaščito na nivoju transportnega protokola pa so lahko omejitve tudi posledica transportnega protokola.

Programski model WCF ima tudi določene mehanizme za preverjanje pooblastil. Na voljo imamo dva modela. Prvi temelji na vlogah[8], na podlagi katerih dovoljuje uporabnikom izvajanje procesov in dostop do virov. Vloga je simbolična kategorija, ki povezuje skupaj uporabnike z istimi pravicami dostopa. Pri tej metodi se najprej določi uporabnik in njegova vloga. Nato se uporabnikova vloga primerja z vlogo, ki lahko dostopa do vira. Velik poudarek je tako na uporabniku in vlogah, ki jih lahko zavzema. Podatki uporabnika, kot so datum rojstva in podobni, niso pomembni. Tak način overjanja uporabnikov lahko pri velikih sistemih postane zelo kompleksen. Število različnih vlog, ki jih ima lahko nek uporabnik, je zelo veliko. Poveča se možnost napak pri dodeljevanju vlog uporabnikom. V tem primeru lahko pride do nedovoljenega dostopa do podatkov ali storitev. V WCF lahko za vloge uporabimo skupine v okolju Okna ali pa so v podatkovni bazi. Drugi model temelji na trditvah. Trditev je neka informacija, povezana z entiteto v aplikaciji. Tipično je to uporabnik, lahko pa je tudi storitev ali vir, ki ga aplikacija nudi. Trditev je sestavljena iz treh delov. Tip trditve, vrednost oziroma vsebina trditve in pa vrednosti, ki pove, ali trditev opisuje identiteto ali sposobnost subjekta. Za preverjanje pooblastil se uporablja več trditev. Ena trditev določa identiteto, druge pa sposobnost subjekta, na katerega so vezane trditve.

3.6 PREPROST PRIMER IMPLEMENTACIJE WCF STORITVE

Za lažjo predstavitev, kaj vse je potrebno storiti pri implementaciji preproste storitve v tehnologiji WCF, tukaj podajam preprost primer storitve. Storitev omogoča pošiljanja pozdravnega sporočila odjemalca in strežnika. V obe smeri se pošlje objekt, ki vsebuje podatke o imenu pošiljatelja in pa njegovo sporočilo.

Najprej moramo definirati vmesnik storitve in njegovo implementacijo. Storitev se lahko implementira tudi brez vmesnika. V tem primeru se z ustreznimi atributi označi razred, ki storitev implementira.

```
[ServiceContract]
public interface IPrimer
{
    [OperationContract]
    Oseba LepPozdrav(Oseba value);
}

public class Primer : IPrimer
{
    public Oseba LepPozdrav(Oseba oseba)
    {
        return new Oseba();
    }
}
```

Razred, ki se prenaša med odjemalcem in strežnikom, moramo ravno tako označiti z ustreznimi atributi.

```
[DataContract]
public class Oseba
{
    string _ime;
    string _sporocilo;

    [DataMember]
    public bool ime
    {
        get { return _ime; }
        set { _sporocilo = value; }
    }

    [DataMember]
    public string sporocilo
    {
        get { return _sporocilo; }
        set { _sporocilo = value; }
    }
}
```

Tako definirana storitev je pripravljena za izvajanje. Potrebno je določiti samo način gostovanja, končne točke in vezavo. Če uporabljamo za izdelavo storitve MS Visual Studio, nam ta generira privzeto nastavitveno datoteko. To lahko potem spremenimo, da ustreza našim potrebam.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" />
  </system.web>
    <system.serviceModel>
    <services>
      <service behaviorConfiguration="Primer.Service1Behavior"
               name="Primer.Primer">
        <endpoint address=""
                  binding="wsHttpBinding"
                  contract="Primer.IPrimer">
          <identity>
            <dns value="localhost" />
          </identity>
        </endpoint>
        <endpoint address="mex"
                  binding="mexHttpBinding"
                  contract="IMetadataExchange" />
        <host>
          <baseAddresses>
            <add
              baseAddress="http://localhost:8731/Primer/Service1/" />
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="Primer.Service1Behavior">
       <serviceMetadata httpGetEnabled="True"/>
       <serviceDebug includeExceptionDetailInFaults="False" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Pri razvoju odjemalca se nato s pomočjo posebnega orodja generira posrednika, ki bi ga lahko generirali tudi ročno. Ročno generiranje posrednika je dolgotrajen, vendar preprost postopek, rezultat pa je skoraj enak, kot pri samodejnem generiranju.

Ko je posrednik generiran, nudi odjemalcu metode za izvajanje operacij na strežniku in tudi objekte, ki niso definirani na strani odjemalca. Klic strežnika s strani odjemalca preko posrednika za predstavljen primer je takšen.

```
PrimerVmesnik.PrimerClient primer
    = new PrimerVmesnik.PrimerClient();


primer.LepPozdrav(new PrimerVmesnik.Oseba());
```

Kot vidimo, je implementacija preproste aplikacije v tehnologiji WCF preprosta. Če primerjamo tehnologijo z ostalimi, ugotovimo, da ima WCF pred tehnologijo CORBA

prednost predvsem pri preprosti uporabi, saj ni potrebno definirati IDL vmesnikov in pa tudi sam programski model je preprostejši. Pri primerjavi s tehnologijami Java RMI in .NET Remoting pa lahko ugotovimo, da tudi tu programski model WCF ponuja prednosti, vendar pa je velika prednost tudi podpora različnim spletnim standardom.

Tabela 1: Povzetek primerjave tehnologij.

	WCF	Java RMI	CORBA	.NET Remoting
Referenca	Na storitev	Na objekt	Na objekt	Na objekt
Prenašanje objektov po vrednosti	Vsi označeni z atributom DataContract	Vsi označeni z atributom Serializable	Objekti definirani v jeziku IDL	Vsi označeni z atributom Serializable
Prenos implementacije objekta	Ni možna	Je možna	Ni možna	Ni možna
Združljivost z drugimi tehnologijami	Po standardih spletnih storitev in s predhodnimi Microsoftovimi tehnologijami	Privzeto je implementirana podpora CORBA standarda	Je standard in je zato lahko implementiran v različnih programskih jezikih	Ni združljivo
Varnost	Zaščita na nivoju transportnega protokola in sporočila. Uporaba SSL in naprednih standardov spletnih storitev za varnost	Zaščita na nivoju transportnega protokola z uporabo SSL protokola	Zaščita na nivoju transportnega protokola z uporabo SSL protokola	Zaščita na nivoju transportnega protokola z uporabo SSL protokola

4 PRIMER UPORABE TEHNOLOGIJE WCF V POSLOVNI APLIKACIJI

Poslovne aplikacije so danes ena bolj razširjenih področij uporabe programske opreme. Primer aplikacije, ki je nastal v okviru tega diplomskega dela, se nanaša na področje upravljanja s sredstvi in obveznostmi. Specifičnost tega področja in pa obstoječa implementacija določenih modulov sta v veliki meri vplivala na izbiro tehnologije WCF za potrebe komunikacije.

4.1 UPRAVLJANJE S SREDSTVI IN OBVEZNOSTMI

Upravljanje s sredstvi in obveznostmi je postopek ugotavljanja in ravnanja s tveganji, ki se pojavijo zaradi razlik med sredstvi in obveznostmi. Sredstva in obveznosti so si lahko zelo različna. Razlikujejo se po načinu določanja trenutne cene, obrestih, zapadlosti in podobno. Glavni problem, ki se lahko pojavi zaradi teh razlik, je, da finančna institucija v določenem trenutku nima kapitala za pokrivanje tekočih obveznosti oziroma posluje z izgubo.

Vzemimo kot primer banko, ki vzame posojilo v vrednosti sto milijonov evrov za obdobje enega leta po obrestni meri tri odstotke letno. Nato celoten znesek posodi za pet let po obrestni meri štiri odstotke letno. Na prvi pogled banka služi en odstotek letno in je posel dobičkonosen. Problem se pojavi, ko mora banka konec prvega leta ponovno vzeti posojilo, saj se ji prvo posojilo izteče in ga mora vrniti. Če se v tem času obrestna mera zviša, na recimo šest odstotkov, ima banka s tem posojilom izgubo.

Upravljanje s sredstvi in obveznostmi odkriva ravno takšne situacije. Pojavilo se je več metod za analizo danih sredstev in obveznosti. Najbolj uveljavljene so analiza odmikov med sredstvi in obveznostmi, analiza trajanja in podobne. Skupno vsem analizam je uporaba scenarijev za podajanje pričakovanega gibanja naložb in obrestnih mer. Scenarije določa strokovnjak na podlagi izkušenj, lahko pa se uporabi tudi naključno generiranje večjega števila scenarijev, katerih rezultat izračunov se potem ustrezno uteži.

Tovrstne analize se lahko uporabljajo za analizo različnih področji, predvsem pa se uporablja v finančnih institucijah, kot so banke in zavarovalnice. Te institucije pridobijo sredstva z depoziti, zavarovanji in podobno. Ta sredstva vlagajo naprej v različne finančne inštrumente, nepremičnine, dajejo posojila. Temu pravimo obveznosti, ki jih mora institucija v določenem trenutku pokriti s kapitalom.

Posameznemu sredstvu ali obveznosti rečemo z eno besedo pozicija, ki zajema podatke o stanju finančnega inštrumenta in je tako lahko sredstvo ali obveznost. Množici pozicij, ki predstavljajo njihovo stanje v nekem trenutku, pa pravimo presek. Izračuni se izvajajo na posameznem preseku, kjer lahko zajete pozicije še omejimo glede na atribute, ki opisujejo

posamezno pozicijo. Kljub omejitvam, ki jih podamo pri posameznem izračunu, je lahko pozicij, ki so zajete v izračun, še vedno veliko.

4.2 ZAHTEVE SISTEMA

Zahteve sistema za upravljanje s sredstvi in obveznostmi so splošne in se ne razlikujejo od tipičnih poslovnih aplikacij. Rešitve teh zahtev pa so nekoliko bolj specifične zaradi velikega števila podatkov.

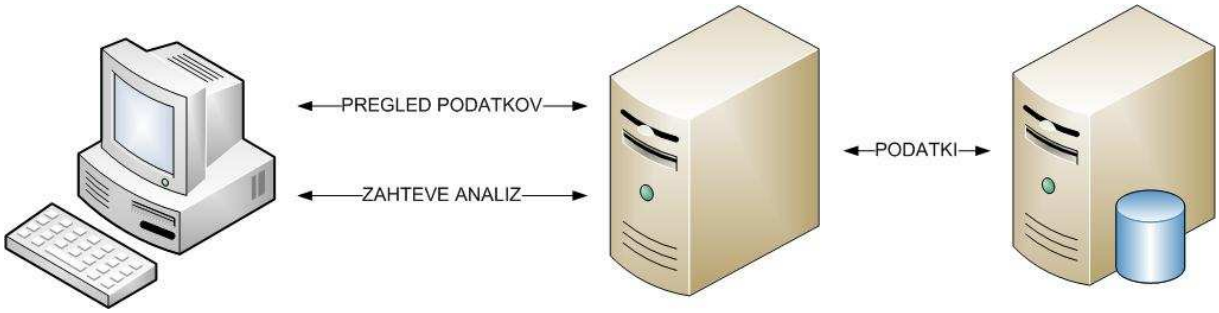
Zahteve sistema so naslednje:

- Sistem mora imeti možnost poganjanja različnih analiz na veliki količini podatkov. Ta pomeni daljše izvajanje analiz in s tem večjo zasedenost računalnika.
- Med izvajanem analize se mora aplikacija primerno odzivati in podati smiselna sporočila uporabniku. Zaradi velikega števila podatkov to pomeni, da se mora analiza izvajati asinhrono, saj v nasprotnem primeru aplikacija postane neodzivna. Neodzivnost sistema lahko privede do nezadovoljnih uporabnikov, ki funkcionalnosti ne uporabljajo v polni meri, ampak imajo odpor do uporabe.
- Sistem mora imeti možnost pregleda podatkov, ki se bodo uporabljali v analizah. Zaradi velike količine podatkov ni smiselno prikazovati vseh pozicij v določenem preseku. Prikaz pozicij omejimo na tiste omejene z izbranimi atributi.
- Podatke in parametre za analizo je možno urejati oz. dodati nove.
- Sistem omogoča shranjevanje in pregledovanje opravljenih analiz.
- Število hkratnih odjemalcev pri posamezni namestitvi aplikacije ni veliko in tudi količina podatkov, ki se prenašajo, ni velika. Tipično se podatki preko odjemalca spreminjajo in dodajajo redko. Tudi pri posamezni analizi je čas klica kratek v primerjavi s samo analizo.
- Zaradi velike količine podatkov in dolgotrajnih izračunov je potrebno minimizirati število klicev SQL podatkovnega strežnika. Pri izračunih velikokrat pride do različnih pretvorb vrednosti oziroma uporabe šifrantov. V ta namen se uporablja medpomnilnik za tečajnice in šifrante, kjer so ti dostopni v pomnilniku in jih ni potrebno vedno ponovno prenašati iz baze podatkov.

4.3 ZASNOVA SISTEMA

Zaradi velike količine podatkov in zahtev sistema je smiselna zasnova sistema odjemalec strežnik. S to zasnovo pridobimo na manjši obremenjenosti uporabnikovega računalnika, tako da lahko uporabnik med računanjem analize nemoteno uporablja računalnik. Kljub temu pa lahko uporabnik prejme ali zahteva stanje izvajajočih analiz. Zasnova je trinivojska.

Sestavljena je iz lahkega odjemalca, ki omogoča podajanje analiz v izračun, pregleda opravljenih analiz z morebitnimi napakami, pregleda podatkov pozicij in njihovo spreminjanje ter vnašanje filtrov in scenarijev za izračun. Drugi del aplikacije predstavlja strežnik, ki izvaja izračune ter je vmesni člen med odjemalcem in podatkovno bazo. Tretji del je SQL podatkovna baza, ki služi za shranjevanje pozicij, izračunov in ostalih podatkov, potrebnih pri izračunih, kot so tečaji valut in podobno. Odjemalec nikoli ne dostopa do SQL podatkovne baze in vsa komunikacija poteka preko strežnika.



Slika 7: Trinivojska arhitektura sistema omogoča razbremenitev uporabniškega računalnika in centraliziran dostop do podatkov in analiz.

Razlogi za izbiro trinivojske arhitekture odjemalec strežnik so naslednji:

- Ker se lahko izračuni analiz izvajajo dlje časa in so tudi računsko zahtevni, je izvajanje na uporabniškem računalniku nesmiselno. Uporabnik v tem času računalnika ne bi mogel uporabljati oziroma bi bila uporaba zelo upočasnjena. Iz tega razloga je smiselno, da se na uporabniškemu računalniku poda samo zahteva za izračun, ki se nato izvede na nekem namenskem računalniku oziroma strežniku.
- Arhitektura odjemalec strežnik omogoča organizaciji, da ima centraliziran pregled nad opravljenimi analizami in se zato te ne bodo podvajale. Vse analize so tudi shranjene na enem mestu in so zato dostopne vsem pooblaščenim osebam od koder koli. V primeru samostojne aplikacije bi se lahko nekatere analize podvajale.

4.4 GOSTOVANJE PROCESA STORITVE

WCF za gostovanje ponuja nekaj možnosti in vsaka ima svoje prednosti in slabosti. Pri izbiri pravega načina je pomembno predvsem, da mora aplikacija ohranjati stanje. Izračuni analiz temeljijo na že implementiranih metodah, ki predvidevajo, da so določeni podatki na voljo v pomnilniku in ni potrebno vedno ponovno dostopati do podatkovne baze. Taki podatki so predvsem različne tečajnice in šifranti. Analize tipično naredijo veliko pretvorb valut in vrednosti pri nekem izračunu in zato je razlog, da so ti podatki vedno v pomnilniku, predvsem večja učinkovitost oziroma hitrost pri izvajanju teh izračunov. Na podlagi teh

potreb sistema moramo izbrati način gostovanja, ki nam omogoča vzpostavitev takšnega okolja za izračune.

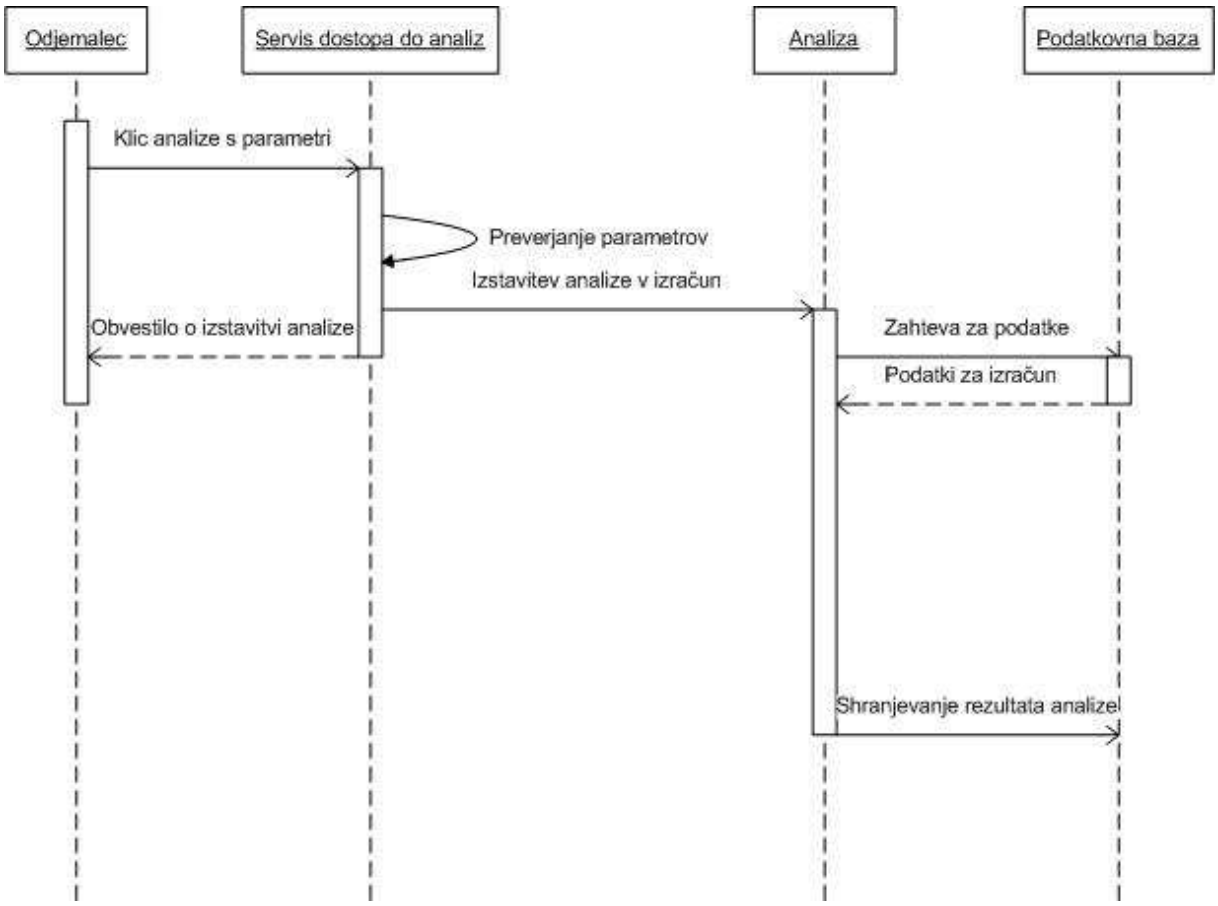
Zaradi opisanih razlogov gostovanje na podlagi Microsoftovega IIS spletnega strežnika ni možno, saj ta način ne ohranja stanja. Slabost je tudi omejenost na HTTP protokol. Ostaneta nam tako način gostovanja v namenski aplikaciji in pa gostovanje v procesu operacijskega sistema Okna. Oba načina nudita veliko mero fleksibilnosti pri uporabi različnih protokolov in pa pri upravljanju z življenjsko dobo storitve. S stališče razvoja je nekoliko bolj primerno gostovanje v namenski aplikaciji, ker se nam ni potrebno po vsaki spremembi storitve ukvarjati s posodobitvijo procesa v operacijskem sistemu. Aplikacijo enostavno prevedemo in zaženemo. Kljub temu pa ima gostovanje v procesu Oken svoje prednosti, ki bi jih lahko uporabili predvsem pri končni aplikaciji. Te prednosti so povezane predvsem z varnostjo takšnega procesa in pa pri aktivaciji procesa.

Za gostovanje končne aplikacije je tako najprimernejše gostovanje znotraj procesa Oken. Za razvoj pa znotraj namenske aplikacije. Storitev se ne razlikuje, pri obeh načinih implementacije pa je zelo preprosta. Pri obeh načinih pa je poleg ustreznih razredov .NET okolja potrebno dodati tudi nalaganje tečajnic in šifrantov.

4.5 KLICI IZRAČUNOV

Posamezne analize lahko trajajo dlje časa. Pri navadnih klicih se izvajanje kode pri klicu ustavi in se nadaljuje potem, ko metoda vrne rezultat. Rečemo, da se izvajanje blokira. Ta način v našem primeru ni primeren, saj bi odjemalec predolgo čakal na odgovor in v tem času uporabnik ne bi mogel uporabljati aplikacije. Uporabiti moramo nek način, ki ne blokira. Ena od možnosti je uporaba asinhronega klicanja storitve. V tem primeru se odjemalec ne blokira, vendar pa se pojavi problem, kaj storiti z odgovorom na klic v primeru, ko uporabnik odjemalca zapre. Strežnik mora v tem primeru ohranjati tudi referenco na vse odjemalce.

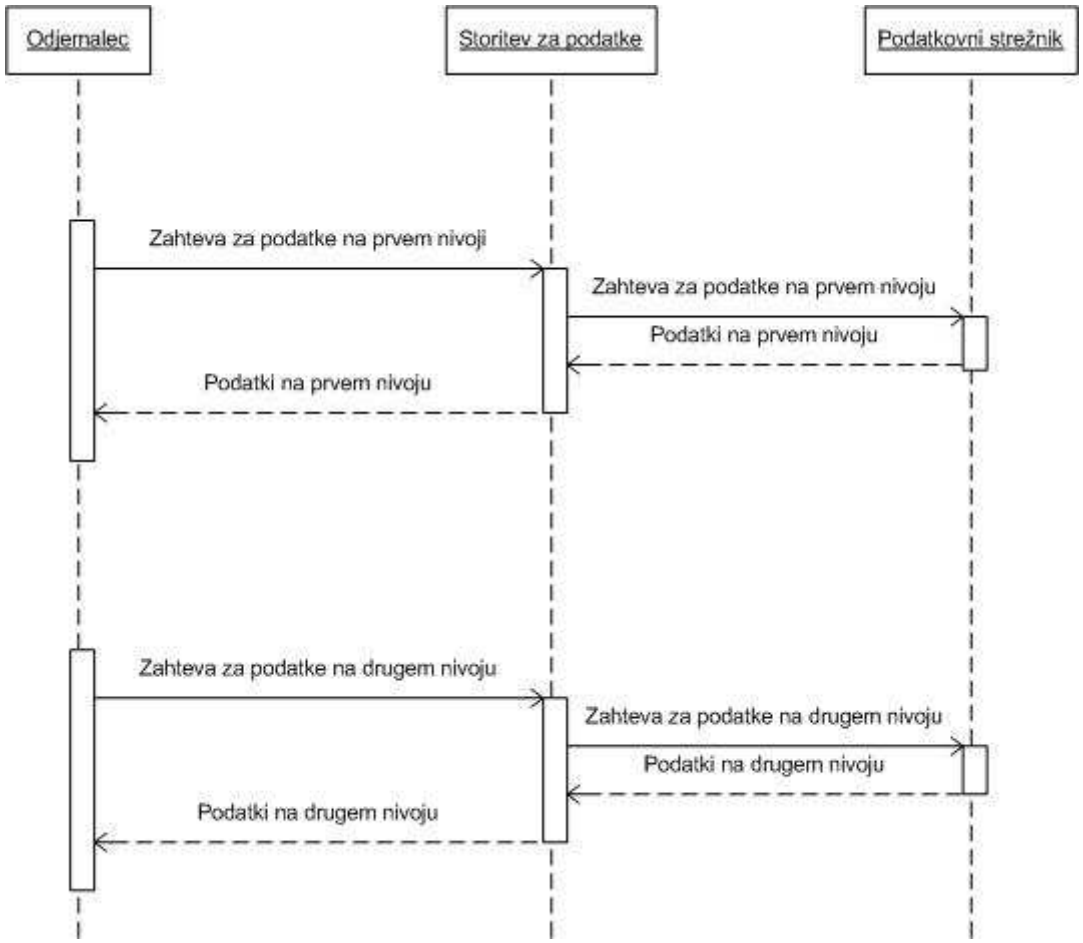
Tretji način, ki je tudi uporabljen v aplikaciji, je tak, da odjemalec samo izstavi izračun v opravilo, ta pa mu vrne sporočilo o uspešnosti tega izstavljanja. Ta del je zelo kratek in v njem se opravi tudi preverjanje atributov posameznega izračuna. O uspešnosti samega izračuna ne vemo ničesar. Obvestilo o uspešnosti izračuna lahko odjemalec zahteva naknadno. Strežnik odjemalca ne obvešča o zaključku analize. Pri tem načinu ni problema z blokiranjem odjemalca in ohranjanjem referenc.



Slika 8: Sekvenčni diagram za klic analize

4.6 PREGLED IN UREJANJE PODATKOV

Pri sistemu lahko pregledujemo in urejamo različne podatke. Večina teh podatkov, kot so filtri, scenariji in pregled opravljenih analiz, ni problematičnih in je njihova implementacija enostavna. Problem pa se pojavi pri pregledovanju pozicij. Teh je lahko zelo veliko in lahko se pojavi problem pri prenosu vseh podatkov in njihovem prikazu uporabniku. Da bi omejili podatke se uporabljajo atributi za filtriranje in pa dva atributa, ki služita za grupiranje pozicij. Zahteva za pregled podatkov je razdeljena na dva dela, in sicer na prvi del, kjer se glede na podana atributa prenesejo podatki o vrednostih teh atributov na prvem in drugem nivoju. Dobimo drevesno strukturo z vrednostmi teh atributov. Ko uporabnik nato razširja to drevesno strukturo in zahteva tretji nivo, se prenesejo podatki o vseh pozicijah, ki ustrezajo vrednostim izbranih atributov na prejšnjih dveh nivojih. Na ta način se omeji količina prenesenih in prikazanih podatkov.



Slika 9: Sekvenčni diagram za pregled pozicij

4.7 VARNOST

Pri vsaki poslovni aplikaciji je varnost na prvem mestu. Pri vzpostavitvi varnosti pa moramo zelo dobro poznati okolje, v katerem bo aplikacija delovala. Aplikacija, katero poskušamo zavarovati, bo v veliki večini primerov delovala na lokalnem omrežju, ni pa izključeno tudi delovanje preko interneta. Tudi lokalna omrežja pa so lahko zelo velika in lahko pride do zlorab znotraj same organizacije. V osnovi je aplikacija namenjena več strankam in njihova okolja so lahko zelo različna. Potrebno je poskrbeti, da bo zagotovljena varnost ne glede na okolje. Druga stvar, ki pomembno vpliva na varnost, je hitrost delovanja. Implementacija in nastavitve varnosti naj ne bi moteče vplivale na uporabniško izkušnjo, ki jo ima z aplikacijo. Ta problem se lahko pojavi pri aplikacijah, pri katerih je veliko komunikacije. Ker je strežnik preobremenjen z zahtevami, mora uporabnik dlje čakati na odgovor. V primeru implementirane aplikacije pa je komunikacije malo. V tem primeru je čas, potreben za vzpostavitev varne komunikacije, majhen v primerjavi z ostalim delom, ki ga opravlja aplikacija.

Pri aplikaciji za upravljanje s sredstvi in obveznostmi bo zaščita temeljila na kriptografiji in že obstoječi implementaciji uporabnikov. Na izbiro imamo zaščito na nivoju transportnega protokola ali na nivoju sporočila. Zaščita na nivoju transportnega protokola deluje nekoliko hitreje, vendar pa nas lahko ta izbira omejuje pri nekaterih drugih odločitvah. Glede na to, da je komunikacije pri naši aplikaciji relativno malo, nam ta hitrost ne prinese nobene posebne prednosti. V aplikaciji za upravljanje s sredstvi in obveznostmi je tako uporabljena zaščita na nivoju sporočila. S tem se pridobi na fleksibilnosti pri zaščiti sporočila. Ta več ni odvisna od prenosnega protokola. Ta način ponuja tudi veliko več možnosti pri uporabi različnih načinov preverjanja pristnosti in pooblastil. Za preverjanje pristnosti pa se uporabljata uporabniško ime in geslo. Boljša rešitev bi bila uporaba certifikatov, vendar pa je trenutna implementacija preizkušena in deluje dobro. V kolikor se pojavi potreba po varnejšem načinu overjanja pristnosti odjemalca, pa se lahko obstoječa implementacija uporabnikov razširi s certifikati. Uporaba certifikatov je primerna tudi v primeru, da se za dostop uporabi kakšen drug odjemalec oziroma uporabnik. Dostop lahko recimo omogočimo uporabnikom preko interneta. V tem primeru je preverjanje pristnosti uporabnika s certifikati še toliko bolj primerno, saj nam izdajatelj certifikata garantira pristnost uporabnika. V lokalnem omrežju to ni tako velik problem, saj poznamo vse uporabnike omrežja in lahko morebitnega kršitelja hitreje najdemo in uvedemo sankcije proti njemu. Na internetu pa je takšno osebo težje najti in uvesti sankcije proti njej, saj je lahko v drugi državi in je podrejena drugim zakonom. Z uporabo certifikatov se v veliki meri temu izognemo.

Preverjanje pooblastil v aplikaciji za upravljanje s sredstvi in obveznostmi tako temelji na trditvah, in ne vlogah. Kot trditve v sistemu uporabimo uporabniška imena in gesla in pa digitalne certifikate. V primeru, da želimo za preverjanje pristnosti uporabiti še kakšen drug način, je to zelo enostavno dodati. Videli smo, da lahko za uporabnika na internetu uvedemo certifikate. Podobno bi lahko za lokalne uporabnike, ki nimajo pooblastil za izvajanje analiz, uvedli neko enostavnejšo metodo za pregled že izvedenih analiz. Uporabimo lahko recimo njihov naslov elektronske pošte, kamor se jim recimo pošlje tudi poročilo zahtevane analize.

4.8 VEZAVA

Večino podrobnosti komunikacije in varnosti se v tehnologiji WCF nastavlja pri vezavah. Da bi lahko vse prej opisane podrobnosti komunikacije in zaščite tudi implementirali, je potrebno izbrati ustrezno vezavo in jo prilagoditi potrebam aplikacije. Poleg tega da lahko v WCF definiramo svoje vezave, ima WCF definirane vezave, kjer je vsaka namenjena določenemu načinu in namenu komunikacije. Glavne značilnosti, ki vplivajo na izbiro vezave pri aplikaciji za upravljane s sredstvi in obveznostmi, so razmeroma malo komunikacije, možnost varnega delovanja preko interneta, morebitna uporaba drugih odjemalcev in uporabnikov ter možnost uporabe različnih načinov za preverjanje pristnosti uporabnika.

Pri izbiti vezave moramo tako izbrati takšno, ki bo omogočila vse podrobnosti komunikacije in varnosti. Hitro lahko ugotovimo, da ena sama vezava ne bo dovolj. Problem se pojavi predvsem pri preverjanju pristnosti. Posamezni vezavi lahko določimo samo en način

preverjanja. Da bi lahko uporabljali več različnih načinov preverjanja pristnosti, moramo uporabiti več različnih konfiguracij vezav, kjer je vsaka dodeljena svoji končni točki. Vsaka vrsta odjemalca lahko potem dostopa do storitve preko svoje končne točke. Tak način povezovanja odjemalcev pa nam prinese še dodaten nivo varnosti. Strežnik lahko v topologijo omrežja vpeljemo tako, da imajo do določene končne točke dostop samo določeni odjemalci. Tako lahko odjemalcu na internetu omogočimo samo dostop do končne točke, ki uporablja preverjanje pristnosti s certifikati. Iz notranjega omrežja pa lahko dostopajo vsi odjemalci.

Ker se uporablja več vezav in končnih točk, bi lahko uporabili za posamezno končno točko vezavo, ki je za določeno situacijo najprimernejša. V aplikaciji za upravljanje s sredstvi in obveznostmi je glede na zahteve uporabljena enaka vezava z nekoliko spremenjenimi nastavitvami. Tako se pridobi tudi na večji splošnosti aplikacije in pri morebitnih težavah lažje ugotavljamo vzrok le teh.

Vezava, ki zadostuje naštetim kriterijem, temelji na naprednih standardih spletnih storitev. Takšna vezava je `WSHttpBinding`. Zadostuje vsem naštetim kriterijem po varnosti in hkrati omogoča uporabo nekaterih drugih naprednih mehanizmov, kot so transakcije in zanesljivost prenosa, v kolikor se v prihodnosti pojavi potreba po njih. Uporaba spletnih standardov tudi omogoča uporabo različnih odjemalcev ob čim manjšem številu različni vezav in končnih točk. Lahko bi sicer za vsako novo vrsto odjemalca uvedli novo vezavo, ki bi bila prilagojena tehnologiji odjemalca in namenu njegove uporabe. Z uporabo spletnih standardov pa to ni potrebno.

5 SKLEPNE UGOTOVITVE

Za izdelavo varnih odjemalec stražnik aplikacij obstajajo različne tehnologije, vendar pa lahko pri njih opazimo veliko podobnosti. Glavna razlika se pojavi pri njihovi uporabi. Novejše tehnologije uvajajo vedno enostavnejše načine implementacije in konfiguracije posameznih podrobnosti, ki pa jih je zelo veliko. S poenostavitvijo programskega modela pa se lahko razvijalec posveti problemu, ki ga poskuša rešiti, in predvsem izbiri in proučevanju najprimernejših mehanizmov komunikacije in varnosti.

WCF je najnovejša tehnologija, ki jo za gradnjo porazdeljenih aplikacij ponuja Microsoft. Velik poudarek je na spletnih storitvah in to je tudi glavna prednost te tehnologije. Spletne storitve se vedno bolj uveljavljajo predvsem v poslovnem okolju, saj omogoča hitro dodajanje nove funkcionalnosti in integracije obstoječe ne glede na uporabljeno tehnologijo.

Ko enkrat poznamo programski model WCF, je implementacija novih storitev enostavna. Aplikacija za upravljanje s sredstvi in obveznostmi je lep primer uporabe te tehnologije v poslovnem okolju.

SLIKE

TABELE

VIRI

[1] Chris Peiris and Dennis Mulder, *Pro WCF Practical Microsoft SOA Implementation, Apress*, 2007

[2] František Plášil & Michael Stal, *An Architectural View of Distributed Objects and Components in CORBA, Java RMI, and COM/DCOM*, Springer 98. Dostopno na: http://www.stal.de/Downloads/springer98.pdf

[3] CORBA: Common Object Request Broker Architecture, http://www.omg.org.

[4] Java RMI, http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

[5] .NET Remoting, http://msdn.microsoft.com/en-us/library/kwdt6w2k(VS.71).aspx

[6] W3C definicije spletnih storitev, http://www.w3.org/2002/ws/

[7] Windows Communication Foundation, http://msdn.microsoft.com/en-us/library/ms735119.aspx

[8]Craig McMurtry, Marc Mercuri, Nigel Watling, *Microsoft® Windows® Communication Foundation Hands-on, Chapter 4: Security, Claims-based Authorization Versus Role-based Authorization* , Sams, 2006

IZJAVA

Študent Jože Matko izjavljam, da sem avtor tega diplomskega dela, ki sem ga napisal pod mentorstvom doc. dr. Matijo Maroltom, in dovolim objavo diplomskega dela na fakultetnih spletnih straneh.

V Ljubljani, dne 25.11.2008 Podpis:

_____