Prirodno-matematički fakultet

Kragujevac

Diplomski rad

-Windows Communication Foundation-

Profesor :
doc. dr Boban Stojanović

Student:
Ana Jovanović 28/01

Kragujevac, novembar 2009.

Uvod

.NET Framework 3.0 obuhvata skup upravljivih API-a koji čine sastavni deo Windows Vista i Windows Server 2008 operativnih sistema. .NET Framework 3.0 koristi CLR .NET Framework verzije 2.0. Najvažnije novine koje je sa sobom doneo .NET Framework 3.0 su sledeća četiri nova paketa:

- **Windows Presentation Foundation (WPF)**, novi podsistem zasnovan na XAML-u namenjen izgradnji aplikacija sa vizuelno razvijenim korisničkim interfejsom.
- **Windows Communication Foundation (WCF)**, je servisno-orijentisan sistem razmene poruka koji omogućava programima da lokalno ili u mreži komuniciraju, slično web servisima.
- **Windows Workflow Foundation (WF)**, je tehnologija za definisanje, izvršavanje, i upravljanje tokovima (workflows). XAML se uobičajeno koristi za deklarisanje strukture tokova. Međutim, tok se takođe može izraziti korišćenjem koda u bilo kom .NET jeziku. Tok se sastoji od aktivnosti. WF omogućava .NET developerima razdvajanje logike od pozadinskog izvršavanja komponenti što doprinosi jasnijem i boljem upravljanju aplikacijom. Ovakav pristup predstavlja metodologiju procesnog razvoja aplikacije (process-driven application methodology) i teži da razdvoji logički tok aplikacije od njenih komponenata koje se izvršavaju.
- **Windows CardSpace** je softverska komponenta koja bezbedno skladišti digitalni identitet osobe i obezbeđuje jedinstven interfejs za izbor identiteta za konkretnu transakciju, kao što je npr. logovanje na website.

Tokom vremena izdato je nekoliko API-a za razvijanje distribuiranih sistema. Uopšteno gledano, distribuiran sistem se može definisati kao sistem u kome se vrši razmena podataka između dva izvršna programa. Imajući u vidu ovu definiciju, izbor distribuiranog API-a u velikoj meri zavisi od odgovora na pitanje: „Da li će sistem biti zatvoren (u okviru organizacije, kuće, firme...) ili će i spoljni korisnici pristupati aplikaciji?“

Ako se distribuiran sistem razvija za zatvorenu upotrebu, lako je postaviti sistem tako da svaki umreženi računar koristi isti operativni sistem, isti razvojni framework (.NET, COM, J2EE,...) i lako je uticati na sigurnost sistema, na autentifikaciju i autorizaciju korisnika. U ovom slučaju može se izabrati API koji će se vezati za određeni operativni sistem ili razvojno okruženje.

Sa druge strane, ako se pravi sistem kome će se pristupati spolja, ne može se diktirati korisnicima koji će operativni sistem koristiti ili u kom frameworku će razvijati softver. Ne može se ni određivati kako će konfigurisati svoje opcije za sigurnost. Zato je u ovim slučajevima potrebno izabrati fleksibilniji API koji će proširiti mogućnost pristupa spoljnim korisnicima.

Nakon kratkog podsećanja na neke od glavnih distribuiranih API-a, lako će se uočiti prednosti Windows Communication Foundation-a:

- **DCOM (Distributed Component Object Model)** - Korišćenjem DCOM-a bilo je moguće napraviti distribuirani sistem pomoću COM objekata i sistemskih registara. DCOM je bio API za, prvenstveno, Windows operativni sistem i iako ga je još nekoliko drugih operavnih sistema podržavalo, sam DCOM nije obezbedio inteligentan način za istovremeno korišćenje više operativnih sistema (Windows, Unix, Mac) niti razmenu podataka između različitih arhitektura (npr. COM i J2EE).

- **COM+/Enterprise Services** - nije korišćen samo od strane COM programera, već je bio potpuno dostupan .NET programerima. Ova tehnologija se i danas koristi, ali je ovo rešenje orijentisano samo ka Windows-u, tako da je pogodna samo za zatvorene sisteme.

- **MSMQ (Microsoft Message Queuing)** - API koji omogućava razvijanje distribuiranih sistema tako da je moguća sigurna razmena podataka u obliku poruka. Pošto u svakom distribuiranom sistemu postoji rizik da server ne bude u funkciji ili da se prekine konekcija sa bazom, od velike pomoći je aplikacija koja može da sačuva podatke u formi poruka koje će kasnije biti prosleđene. Bez obzira na to koji programski model se koristi za interakciju sa MSMQ-om krajnji rezultat je pouzdano dopremanje poruka u realnom vremenu. Kao i COM+, MSMQ je korišćen za izgradnju distribuiranog softvera na Windows operativnom sistemu.

- **.NET Remoting** - API koji omogućava većem broju računara da distribuiraju objekte, pod uslovom da se sve aplikacije izvršavaju na .NET platformi. S obzirom na tu činjenicu, saradnja između različitih arhitektura nije bilo moguće.

- **XML Web Services** - posebnu pažnju treba posvetiti ovom API-u kako bi se WCF mogao u potpunosti razumeti

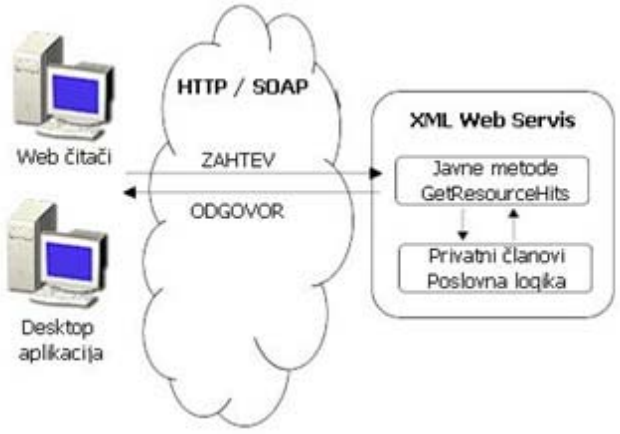
Web servisi predstavljaju osnovne gradivne blokove budućih informacionih sistema. To su aplikacije koje su rasplożive na mreži. Drugim rečima, to su resursi koji se adresiraju primenom URL-a koji vraća informaciju korisniku koji želi da je koristi.

Web servisi omogućavaju jednostavan način razmene podataka i pristupa funkcionalnostima aplikacija na udaljenim računarima putem standardnih web protokola, pri čemu same aplikacije mogu biti razvijene u različitim tehnologijama i raditi na različitim platformama.

Postoji nekoliko osnovnih razloga koji su inicirali razvoj web servisa.

- **Interoperabilnost** (interoperability) je potreba za komunikacijom aplikacija razvijenih u različitim tehnologijama, u različitim programskim jezicima i na različitim platformama. Zbog toga su web servisi i zamišljeni kao standardizovan način komunikacije, nezavistan od korišćenih tehnologija i platformi.
- **SOA** (Service Oriented Architecture) je pristup u razvoju softvera koji podrazumeva razdvajanje funkcija u odvojene servise, dostupne preko mreže ili interneta. Razdvojeni servisi olakšavaju kombinovanje i korišćenje već implementiranih funkcionalnosti bez potrebe njihovog ponovnog razvoja.
- **Skalabilnost** predstavlja potrebu da se postojeći skupovi funkcionalnosti menjaju (povećavaju ili smanjuju) uz minimalne troškove i najmanji mogući uticaj na rad sistema.

Na slici 1 prikazan je način rada web servisa



Slika 1

S obzirom na to da je bilo neophodno omogućiti neometanu komunikaciju različitih tehnologija, bilo je potrebno da web servisi koriste sandardizovane principe i tehnologije. Komunikacija između klijenta i sistema odvija se preko HTTP-a ili HTTPS-a, zavisno od potrebnog nivoa sigurnosti. Što znači da se klijenti konektuju na server preko standardnog porta za Web, a podaci putuju kao XML set podataka uz korišćenje SOAP-a.

XML (eXtensible Markup Language) je jezik koji razumeju različite platforme i različiti programski jezici, i samim tim vrlo pogodan za razmenu podataka i komunikaciju između takvih sistema.

Sa druge strane, **HTTP** protokol kao najrasprostranjeniji web protokol je bio najbolji izbor.

Osnovne komponente web servisa su:

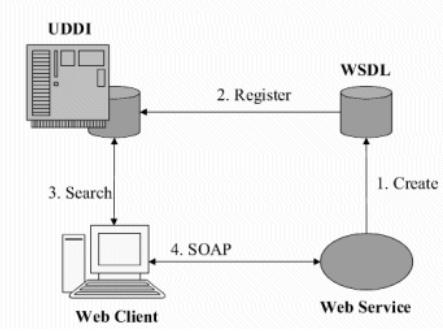
- **WSDL** (Web Services Description Language)
- **SOAP** (Simple Object Access Protocol)
- **UDDI** (Universal Description, Discovery and Integration)

WSDL je dokument zasnovan na XML-u koji opisuje i locira Web servis. WSDL praktično predstavlja ugovor između servera i klijenta na osnovu koga se tačno definiše šta web servis pruža. WSDL sadrži sledeće informacije:

- Tipove i format podataka koje web servis koristi
- Poruke koje je moguće razmenjivati sa web servisom
- Operacije (funkcije i metode) koje web servis poseduje
- Komunikacioni protokol koji web servis koristi.

SOAP predstavlja okruženje za razmenu poruka zasnovanih na XML-u u mreži. Sam SOAP se služi aplikativnim HTTP protokolom koji koriste svi Web serveri (može se reći sledeće: HTTP + XML = SOAP). Glavni deo SOAP specifikacije definiše skup pravila za upotrebu XML za reprezentaciju podataka. Drugi delovi SOAP specifikacije definišu proširivi format poruka, konvencije za udaljeno pozivanje metoda i povezivanje sa HTTP protokolom. SOAP je nezavisan od korišćenih platformi i tehnologija, odnosno programskih jezika. Takođe, omogućava lako zaobilaženje zaštite (firewall-a) računara koji međusobno komuniciraju. U kontekstu rada sa web servisima, SOAP se koristi za specificiranje pozivnih parametara i rezultata rada servisa.

UDDI je odgovor na pitanje: „Gde se može pronaći potreban servis?“. Web Servisi se objavljuju na jedinstvenoj lokaciji i nude se kao usluge. Kako bi usluga bila kompletna, nudi se i potpuna specifikacija interfejsa. UDDI predstavlja centralizoanu lokaciju koja obezbeđuje mehanizam za registrovanje i pronalaženje web servisa. Koristi SOAP za komunikaciju i omogućava klijentima da pronađu servis i omogućava serveru da ga objavi.



Slika 2

Windows Communication Foundation

WCF

1. Osnovni pojmovi i kompozicija WCF aplikacija

WCF je alat za distribuirano programiranje koji je uveden sa .NET 3.0. Korišćenjem ovog alata, servis postaje dostupan spoljnim pozivima različitih tehnologija. Na primer, jednom zatvorenom sistemu, gde su sve mašine na istom operativnom sistemu i koja koristi različite TCP protokole radi ostvarivanja što boljih performansi, mogu pristupiti i spoljni korisnici korišćenjem web servisa baziranim na XML protokolu kako bi izvršavali svoje funkcionalnosti bez obzira na programski jezik ili operativni sistem.

Programska osnova WCF-a je predstavljena sklopovima instaliranih u GAC-u(Global Assembly Cache).

Osnovni WCF sklopovi:

Sklop	**Značenje**
System.Runtime.Serialization.dll	Definiše namespace-eve i tipove koji se mogu koristiti za serijalizaciju i deserijalizaciju objekata u WCF okruženju
System.ServiceModel.dll	Osnovni sklop koji sadrži tipove koji se koriste za izgradnju bilo koje vrste WCF aplikacije

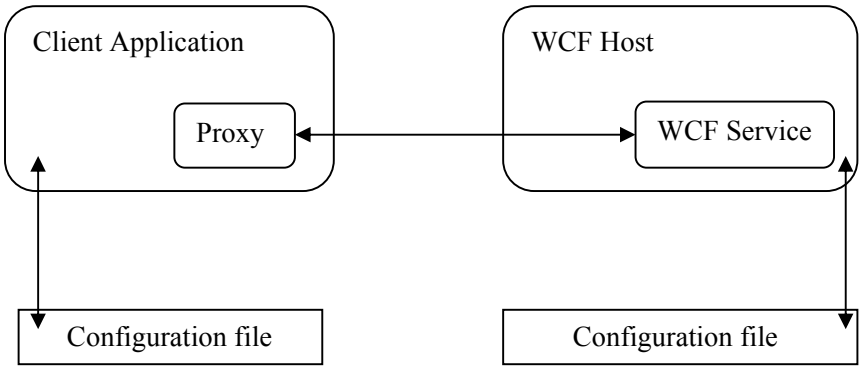
Osnovne WCF oblasti imena:

Namespace	**Značenje**
System.Runtime.Serialization	Definiše brojne tipove koji kontrolišu kako se podaci serijalizuju i deserijalizuju u okviru WCF okruženja
System.ServiceModel	Osnovna oblast koja definiše povezivanje i hostovanje (usluživanje) tipova, kao i sigurnost i transakcije tipova
System.ServiceModel.Configuration	Definiše veći broj tipova koji omogućavaju programski pristup konfiguracionim fajlovima WCF-a
System.ServiceModel.Description	Definiše tipove koji omogućavaju objektni model za adresiranje, povezivanje i ugovaranje u okviru WCF konfiguracionih fajlova
System.ServiceModel.MsmqIntegration	Sadrži tipove za integraciju sa MSMQ servi-som
System.ServiceModel.Security	Definiše veliki broj tipova za kontrolu nivoa sigurnosti

WCF distribuirani sistem se gradi kreiranjem tri povezana sklopa:

- **WCF Service assembly:** Sadrži klase i interfejse koji predstavljaju funkcionalnosti, koje se otkrivaju spoljnim pozivima. Sadrži WCF ugovore (contracts) i njihovu implementaciju
- **WCF Service host:** Celina koja uslužuje service assembly
- **WCF Client:** Aplikacija koja pristupa funkcionalnostima servisa

Slika 1.1 arhitekturom visokog nivoa, na jedan način, prikazuje veze između ova tri sklopa:



Slika 1.1

Korišćenje konfiguracionih fajlova i sa klijentske i sa serverske strane je opciono.

1.1. ABC

ABC je skraćenica za osnovne blokove WCF aplikacije:

- **Address**: Lokacija servisa. U kodu, adresa se predstavlja pomoću *System.Uri* tipa. Ova vrednost se najčešće čuva u konfiguracionim fajlovima
- **Binding**: Način povezivanja klijenta i servera. Način podrazumeva specificiranje mrežnih protokola, šifriranje mehanizama i transportni sloj.
- **Contract**: Opis svake metode koja se otkriva od strane WCF servisa.

Server i klijent mogu da komuniciraju ako se dogovore po ABC stavkama. Skraćenica ABC ne uslovljava programera da mora definisati prvo adresu, zatim povezivanje i pisanje ugovora. U većini slučajeva, počinje se sa definisanjem ugovora, a zatim se definišu adresa i način povezivanja.

1.1.1. WCF Contracts

Pojam ugovora je ključan u izgradnji WCF servisa. Iako ne obavezno, najčešće razvoj WCF aplikacija počinje definisanjem skupa interfejsa koji se koriste za predstavljanje funkcionalnosti koje će WCF tip usluživati. Interfejsi koji predstavljaju WCF ugovore zovu se servis ugovori (service contracts). Klase ili strukture koje ih implementiraju nazivaju se servis tipovi (service types).

WCF servis ugovori su upotpunjeni različitim atributima, od kojih su najčešći definisani u *System.ServiceModel* namespace-u.

Za neke jednostavnije ugovore koji će sadržati samo proste tipove podataka, kompletan WCF servis se može izgraditi pomoću dva atributa: *[ServiceContract]* i *[OperationContract]*. Ako su u pitanju složeniji, korisnički tipovi, biće potrebno koristiti tipove iz namespace-a *System.Runtime.Serialization*. Tu se nalazi veći broj atributa (kao što su *[DataMember]* i *[DataContract]*) za preciznija definisanja interfejsa.

Programer nije u obavezi da koristi interfejse. Većina ovih atributa se mogu koristiti na javnim klasama ili strukturama. Međutim, uzimajući u obzir mnoge prednosti programiranja baziranog na interfejsima (npr. polimorfizam), sigurnije je koristiti interfejse za opisivanje WCF kotrakata.

1.1.2. WCF bindings

Kada je ugovor definisan i implementiran, sledeći korak je izgradnja hosting agenta za sam WCF servis. Postoji veći broj hostova, ali svaki od njih mora specificirati povezivanje (bindings) koje koriste spoljni pozivi za dobijanje pristupa funkcionalnostima servisa. Ako nijedan od ponuđenih povezivanja ne odgovara potrebama aplikacije, može se definisati korisnički tip povezivanja.

WCF povezivanje podrazumeva skup sledećih pojmova:

- Ugovori implementirani od strane servisa
- Transportni nivo za prenos podataka (HTTP, MSMQ, TCP)
- Kanali koji se koriste za transport (one-way, request-reply, duplex)
- Mehanizam šifriranja koji se koristi za rad sa samim podacima (XML, binary…)
- Bilo koji podržani web servis protokoli kao što su WS-Security, WS-Transactions, WS-Reliability…

1.1.2.1. HTTP - based bindings

BasicHttpBinding, WSHttpBinding, WSDualHttpBinding i WSFederationHttpBinding su opcije usmerene ka otkrivanju tipova ugovora putem XML web servis protokola. Ako se zahteva najširа

moguća upotreba distribuiranog servisa (da ga mogu koristiti bilo koji operativni sistemi i bilo koja programska arhitektura), ovo su povezivanja na koja se treba fokusirati, jer ove vrste povezivanja šifriraju podatke tako što se baziraju na XML reprezentaciju i koriste HTTP.

WCF binding se može predstaviti u kodu (pomoću klase u okviru *System.ServiceModel* namespace-a ili kao XML atribut definisan u konfiguracionom fajlu.

- **BasicHttpBinding** je najjednostavniji od svih navedenih protokola. Koristi HTTP za transport i Text/XML za podrazumevano šifriranje poruka. Ovo povezivanje garantuje da je WCF servis postavljen po WS-I Basic Profile 1.1. Glavni razlog korišćenja ovog povezivanja je da se održi kompatibilnost sa aplikacijama koje su prethodno napravljene da komuniciraju sa ASP.NET web servisima (koji su bili deo .NET biblioteka od verzije 1.0).
- **WSHttpBinding** protokol je sličan BasicHttpBinding protokolu, sa tim što omogućava više dodataka za web servis. Npr. dodata je podrška za transakcije, pouzdanu razmenu poruka i adresiranje web servisa. Ne samo da podržava transakcije, bezbednost i pouzdane sesije, već i podržava binarno šifriranje podataka korišćenjem MTOM-a (Message Transmission Optimization Mechanism).
- **WSDualHttpBinding** je sličan kao i WSHttpBinding, sa tim što se koristi za dvostruke ugovore (duplex contracts). Glavna korist WSDualHttpBinding-a je ta da dozvoljava primaocu i pošiljaocu da komuniciraju korišćenjem duplex messaging-a (dvostruka komunikacija).
- **WSFederationHttpBinding** je protokol koji je najbolje rešenje ako je sigurnost od najveće važnosti. Ovaj protokol podržava WS-Trust, WS-Security i WS-SecureConversation specifikacije.

1.1.2.2. TCP-Based Bindings

Ako se razvija distribuirana aplikacija na mašinama koje koriste .NET3.0/3.5 biblioteke (što znači da sve mašine koriste Windows XP, Windows Server 2003 ili Windows Vistu), tada se korišćenjem TCP povezivanja postižu bolje performanse šifriranjem podataka u binarni format pre nego u XML. U ovom slučaju i klijent i server moraju biti .NET aplikacije.

- **NetTcpBinding** klasa koristi TCP za premeštanje binarnih podataka između klijenta i WCF servisa. Ovo doprinosi boljim performansama nego kod web servis protokola, ali postoji ograničenje na Windows rešenja u okviru lokalne mreže. NetTcpBinding klasa podržava transakcije, sigurnu komunikaciju i pouzdane sesije.
- **NetNamedPipeBinding** takođe podržava transakcije, sigurnu komunikaciju i pouzdane sesije, ali nema mogućnost da pravi pozive između mašina. Ovaj protokol je odlično

rešenje ako je cilj da se na najbrži mogući način prenose podaci u okviru WCF aplikacija na istoj mašini.

- **NetPeerTcpBinding** omogućava sigurno povezivanje za P2P mrežne aplikacije.

1.1.2.3. MSMQ-Based Bindings

Ako je cilj integracija sa Microsoft MSMQ serverom, MSMQ-Based povezivanje je dobro rešenje.

- **MsmqIntegrationBinding** se može koristiti da omogući WCF aplikaciji da šalje i prima poruke od i do postojećih MSMQ aplikacija koja koriste COM, C++ ili tipove definisane u *System.Messaging* namespace-u.
- **NetMsmqBinding** je pogodan za komunikaciju između dve .NET aplikacije na različitim računarima.

1.1.3. WCF Adresses

Specificiranje adrese WCF servisa je jako bitan korak, jer spoljni korisnici neće moći da komuniciraju sa udaljenim tipovima ako ne znaju njihovu lokaciju. Adresa može biti navedena u sklopu ili u konfiguracionom fajlu.

U svakom slučaju format adrese će se razlikovati u zavisnosti od vrste povezivanja (Http-based, TCP-based, MSMQ-based). WCF adresa može sadržati sledeće informacije:

- **Scheme**: Transportni protokol (HTTP, ...)
- **MachineName**: Pun naziv računara
- **Port**: Zadavanje broja porta je opciono u većini slučajeva
- **Path**: Putanja do WCF servisa

Šablon za zapis adrese je: *Scheme://<MachineName>[:Port]/Path*

Kada bi se koristilo Http-based povezivanje, adresa bi bila *http://localhost:8080/MojWCFService*

Ako se ne navede broj porta, Http-based protokol će biti podrazumevano na portu 80.

Ako bi se koristio TCP-binding, adresa bi bila: *net.tcp://localhost:8080/MojWCFService*

MSMQ adrese su malo drugačije forme budući da se MSMQ koristi samo na lokalnoj mašini, tako da broj porta nema značaja. Jedan primer ovakve adrese, koja opisuje privatni red (queue) MojPrivateQ bio bi: *net.msmq://localhost/private$/MojPrivateQ*

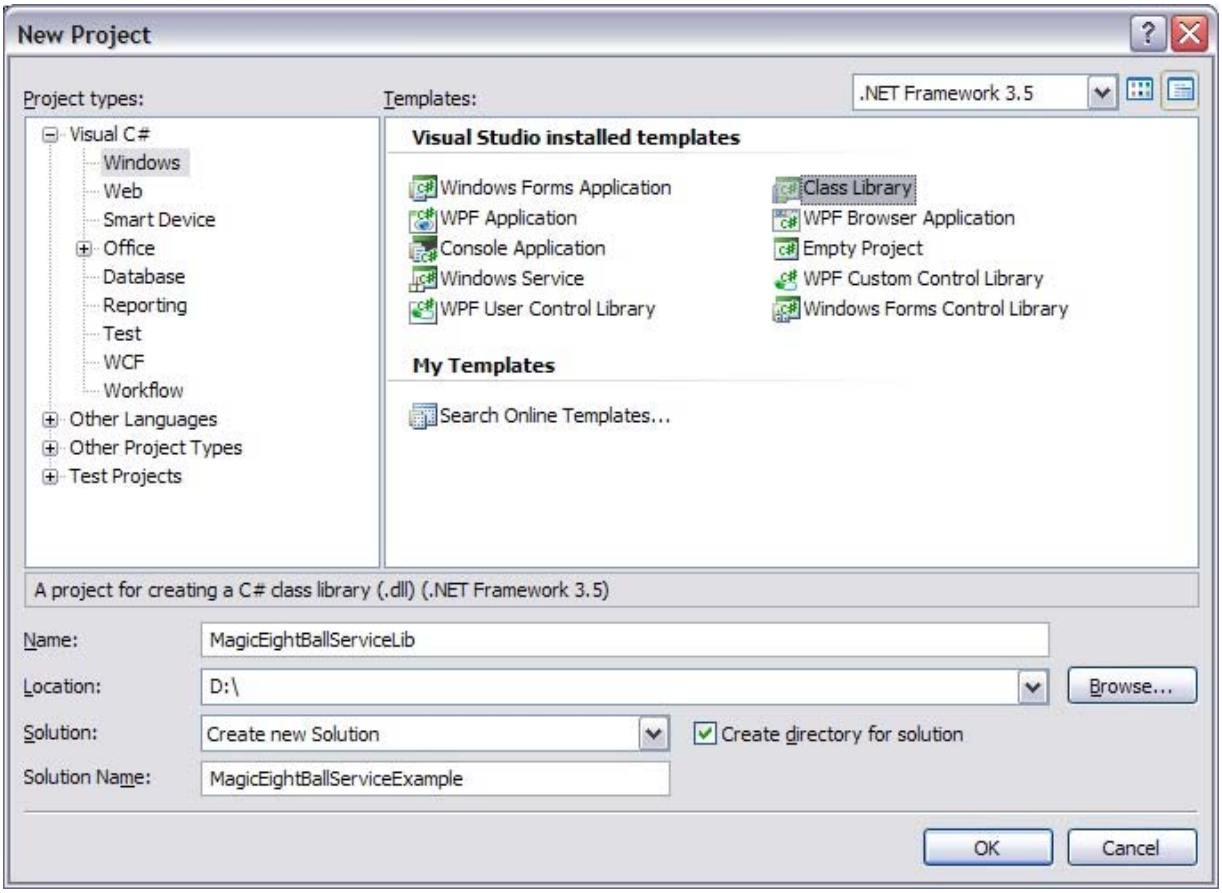
Na kraju, adresa za named-pipe povezivanje bila bi *net.pipe://localhost/MojWCFService*

Dok jedan WCF servis može da otkrije samo jednu adresu, može se konfigurisati kolekcija jedinstvenih adresa (sa različitim povezivanjima). Ovo se može uraditi u okviru konfiguracionog fajla definisanjem višestrukih <endpoint> elemenata. Na taj način, može se specificirati bilo koji broj ABC-ova za isti servis. Ovaj pristup je dobar ako želimo da dozvolimo korisnicima da biraju koji protokol će koristiti kada komuniciraju sa servisom.

2. Izgradnja WCF servisa

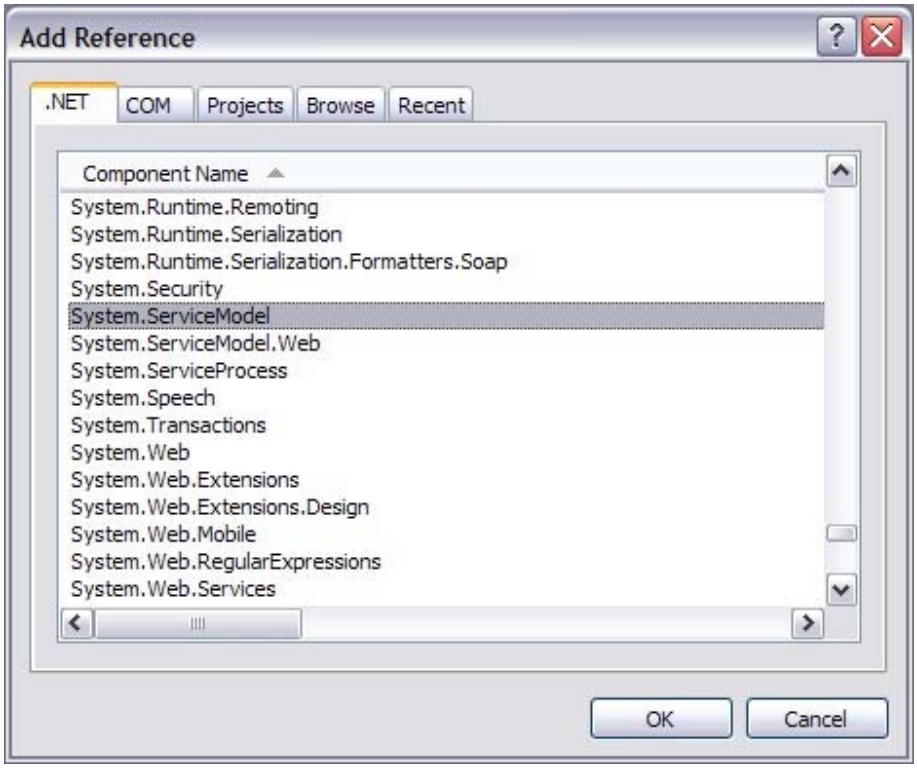
Magična 8-loptica je igračka koja daje jedan od 8 unapred utvrđenih odgovora na pitanje koje osoba postavi. Na jednostavnom primeru magične 8-loptice, proćićemo kroz izgradnju WCF servisa.

Prvi korak u izgradnji WCF servisa je definisanje WCF biblioteka koje u svojoj implementaciji sadrže ugovore. Kreirajmo projekat u jeziku C# koji je tipa klasa biblioteka (Class Library), pod imenom *MagicEightBallServiceLib*:



Slika 2.1

Zatim promeniti ime inicijalnog fajla iz *Class1.cs* u *MagicEightBallService.cs* i dodati referencu na *System.ServiceModel.dll* sklop, kao što je prikazano na slici 2.2.



Slika2.2

U kodu treba specificirati da se koristi *System.ServiceModel* namespace. U ovom trenutku C# fajl ima izgled kao na slici 2.3:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

//Ključni WCF namespase
using System.ServiceModel;

namespace MagicEightBallServiceLib.cs
{
    public class MagicEightBallService
    {
    }
}
```

Slika 2.3

Klasa *MagicEightBallService* će implementirati jedan WCF servis ugovor predstavljen tipiziranim CLR interfejsom *IEightBall*. Interfejs *IEightBall* će definisati jedan metod koji omogućava postavljanje pitanja radi dobijanja slučajnog odgovora.

WCF servis interfejsi su snabdevene *[ServiceContract]* atributom, dok je svaki član interfejsa snabdeven atributom *[OperationContract]*:

```
[ServiceContract]
public interface IEightBall
{
    //Postavite pitanje, da biste dobili odgovor
    [OperationContract]
    string ObtainAnswerToQuestion(string userQuestion);
}
```

Slika 2.4

Dozvoljeno je da se definiše ugovorni interfejs servisa tako da on ima metode koje nisu snabdevene atributom *[OperationContract]*, ali takva metoda neće biti dostupna preko WCF-a.

Upotreba interfejsa nema svrhu ako nisu implementirani od strane klase ili strukture. Kao i prava magična loptica, implementacija *MagicEightBallService*-a će vraćati slučajno izabran odgovor iz niza stringova. Poruke će se prikazivati na konzoli:

```
public class MagicEightBallService : IEightBall
{
    //Metoda za obaveštenje korisniku
    public MagicEightBallService()
    {
        Console.WriteLine("Loptica čeka Vase pitanje...");
    }

    public string ObtainAnswerToQuestion(string userQuestion)
    {
        string[] answers = {"Nesigurno u budućnosti", "Da", "Ne",
           "Nesigurno", "Pitajte kasnije", "Sigurno" };

        //Vraćanje slučajnog odgovora
        Random r = new Random();
        return string.Format("{0}? {1}.",
            userQuestion, answers[r.Next(answers.Length)]);
    }
}
```

Slika 2.5

2.1. [ServiceContract] atribut

Kako bi interfejs učestovao u WCF servisu, mora biti snabdeven *[ServiceContract]* atributom. Dve osobine ovog atributa, *Name* i *Namespace*, se mogu postaviti kako bi se kontrolisao tip servisa i namespace XML-a koji definiše tip servisa. Ako se koristi web-servis tip povezivanja, ove vrednosti se koriste da definišu <portType> elemente odgovarajućeg WSDL dokumenta.

U našem primeru nećemo dati vrednost osobini Name iz razloga što je pod-razumevano ime servisa bazirano na imenu C# klase. Međutim, podrazumevano ime za XML oblast važenja je http://tempuri.org što je poželjno promeniti za svaki WCF servis. Kada se gradi WCF servis koji će da prima i šalje korisničke tipove podataka (što se na ovom primeru neće raditi), važno je postaviti smisleno ime za oblast važenja XML-a, jer će ovo osigurati jedinstvenost korisničkog tipa.

Izmenimo definiciju interfejsa iz našeg primera na sledeći način:

```
[ServiceContract(Namespace = "http://Intertech.com")]
public interface IEightBall
```

Slika 2.1.1

Pored osobina *Namespace* i *Name*, *[ServiceContract]* atribut se može dopuniti i sledećim osobinama:

Osobina	**Značenje**
CallbackContract	Postavlja se vrednost ako ugovor zahteva povratnu reakciju kod dvosmerne razmene poruka
ConfigurationName	Ime koje se koristi da locira element servisa u konfiguracionom fajlu aplikacije. Podrazumevano ime je ime implementacione klase servisa
ProtectionLevel	Omogućava zadavanje stepena sigurnosti, kroz enkriptovanje, digitalni potpis ili oba istovremeno
SessionMode	Koristi se za utvrđivanje da li su sesije dozvoljene ili zahtevane od strane ugovora servisa

2.2. [OperationContract] atribut

Kako je već pomenuto, metode koje želimo da koristimo u okviru WCF okruženja, moraju sadržati atribut *[OperationContract]*, koji takođe može biti konfigurisan različitim osobinama. Neke od tih osobina su date u sledećoj tabeli:

Osobina	**Značenje**
Action	Uzima ili postavlja web servis akciju adresiranja zatražene poruke
AsyncPattern	Pokazuje da li se operacija implementira asinhrono korišćenjem Begin/End parom. Ovo nije u vezi sa klijentovim asinhronim pozivom metode.
IsInitiating	Specificira ako je tekuća operacija inicijalna operacija u sesiji
IsOneWay	Pokazuje da li operacija sadrži samo input poruke, bez pridruženog output-a
IsTerminating	Specificira da li se trenutna sesija treba prekinuti nakon što se operacija završi

2.3. Hostovanje WCF servisa

Host u primeru *MagicEightBallServiceLib* će biti novi projekat tipa *Console Application*, pod nazivom *MagicEightBallServiceHost*. Potrebno je dodati referencu na *System.ServiceModel.dll* i *MagicEightBallServiceLib.dll* sklopove i izmeniti inicijalni fajl uvozom *System.ServiceModel* i *MagicEightBallServiceLib* namespace-eva:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.ServiceModel;
using MagicEightBallServiceLib;

namespace MagicEightBallServiceHost
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("WCF Host");
            Console.ReadLine();
        }
    }
}
```

Slika 2.3.1

Prvi korak u pravljenju host-a za WCF servis je odlučivanje da li želimo da kompletna logika host-ovanja bude u kodu ili želimo da neki detalji budu u konfiguracionom fajlu aplikacije. U našem primeru ćemo koristiti konfiguracioni fajl, tako da je potrebno dodati novi fajl tipa *Application Configuration File* u projekat, pod nazivom App.config.

Pri izgradnji host-a, prate se predvidivi koraci - vezani za konfiguraciju i kod:

- Definisati endpoint za WCF servis koji hostujemo u hostovom konfiguracionom fajlu.
- Programski iskoristiti ServiceHost tip da bi servisni tipovi postali dostupni od strane ovog endpointa
- Pobrinuti se da host radi, kako bi odgovarao na zahteve klijenta.

U rečniku WCF-a, termin *endpoint* predstavlja adresu, povezivanje i ugovor (ABC). U XML-u, *endpoint* se predstavlja korišćenjem taga <*endpoint*> i *address*, *binding* i *contract* elementima. Izmenimo App.config fajl tako da koristi jedan *endpoint* (dostupan sa porta 8080):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="MagicEightBallServiceLib.MagicEightBallService">
        <endpoint address="http://localhost:8080/MagicEightBallService"
                  binding ="basicHttpBinding"
                  contract ="MagicEightBallServiceLib.IEightBall"/>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Slika 2.3.2

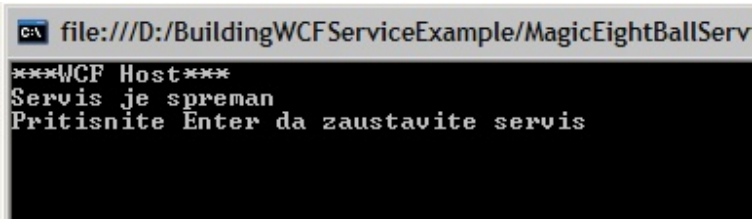
Sa trenutnim konfiguracionim fajlom, programska logika potrebna da bi se kompletirao host je vrlo jednostavna. Kada se naš izvršni fajl startuje, kreiraćemo instancu *ServiceHost* tipa. Ovaj objekat će automatski čitati podatke u okviru namespace-a <*system.serviceModel*> elementa iz konfiguracionog fajla kako bi se utvrdila tačna adresa, povezivanje i ugovor:

```
static void Main(string[] args)
{
    Console.WriteLine("WCF Host");
    using (ServiceHost serviceHost = new ServiceHost(typeof(MagicEightBallService)))
    {
        //Otvara se host i počinje slušanje dolazećih poruka
        serviceHost.Open();

        //Drži se servis u stanju izvršenja dok se ne pritisne Enter
        Console.WriteLine("Servis je spreman");
        Console.WriteLine("Pritisnite Enter da zaustavite servis");
        Console.ReadLine();
    }
}
```

Slika 2.3.3

Ako pokrenemo aplikaciju, na slici 2.3.4 vidimo da host živi u memoriji i čeka da primi zahteve od udaljenih klijenata:



Slika 2.3.4

Pre pravljenja klijentske aplikacije koja će komunicirati sa servisom, upoznaćemo se sa detaljima *ServiceHost* klase i elementa <*service.serviceModel*>.

ServiceHost klasa se koristi za konfiguraciju i otkrivanje WCF servisa. Međutim, direktno korišćenje ove klase će biti moguće samo ako gradimo poseban izvršni fajl da hostuje servis. Ako se koristi IIS da bi se otkrio servis, *ServiceHost* objekat se kreira automatski.

Kao što se vidi, ovaj tip zahteva kompletan opis servisa i iako se ovo dešava automatski pri kreiranju objekta, moguće je ručno konfigurisati stanje *ServiceHost* objekta. Pored *Open*() i *Close*(), u narednoj tabeli ilustrovani su još neki članovi od značaja:

Član	**Značenje**
Authorization	Ova osobina karakteriše nivo autorizacije
AddServiceEndpoint()	Omogućava da se programski registruje endpoint hosta
BaseAddresses	Sadrži listu registrovanih baznih adresa
BeginOpen() BeginClose()	Ove metode omogućavaju asinhrono otvaranje i zatvaranje ServiceHost objekta
CloseTimeout	Osobina koja omogućava zadavanje vremena za koje će servis biti zatvoren
OpenTimeout	Osobina koja omugaćava da se postavi i pročita vreme predviđeno za startovanje servera

Kao i svaki XML element, <*system.serviceModel*> može definisati skup podelemenata. U sledećoj tabeli dati su podelementi <*service.serviceModel*>:

Podelement	**Značenje**
behaviors	WCF podržava više endpoint-eva i ponašanja servisa. Behavior omogućava dalju kvalifikaciju funkcionalnosti domaćina ili klijenta
bindings	Element koji omogućava fina podešavanja svakog od poveziva-nja (basicHttpBinding, netMsmqBinding,...)
client	Element koji sadrži listu endpoint-eva koje klijent koristi za povezivanje sa servisom
comContracts	Element koji definiše COM ugovore omogućene za WCF i COM interoperabilnost
commonBehaviors	Element kome se može postaviti vrednost u machine.config fajlu. Može se koristiti kako bi se definisala sva ponašanja svakog WCF servisa na datoj mašini
diagnostics	Element koji sadrži podešavanja za dijagnostičke elemente WCF-a. Korisnik može (ne)omogućiti praćenje, brojač performansi, dodati proizvoljne filtere za poruke
serviceHostingEnviroment	Element koji sadrži kolekcije otkrivenih WCF servisa

3. Omogućavanje razmene meta-podataka

WCF klijentska aplikacija komunicira sa WCF servisom preko proxy tipa. Iako korisnik može sam napisati ceo proxy kod, to bi bio dugotrajan proces podložan greškama. .NET Framework 3.5 SDK sadrži alat komandne linije, svcutil.exe, za generisanje konfiguracionog fajla klijentske strane aplikacije i generisanje velike količine potrebnog koda. Da bi se omogućilo generisanje konfiguracionog fajla proxy-a klijentske strane pomoću svcutil.exe ili Visual Studia 2008, potrebno je prvo omogućiti MEX (Metadata EXchange) (koji je podrazumevano onemogućen).

Omogućavanje MEX-a je stvar izmene konfiguracionog fajla hosta, na sledeći način:

- Treba dodati novi <*endpoint*> samo za MEX
- Zatim treba definisati *behavior* WCF-a tako da dozvoljava HTTP GET pristup
- Treba povezati *behavior* preko imena sa servisom preko atributa *behaviorConfiguration* na početku <*service*> elementa.
- Na kraju treba dodati <*host*> element bazne adrese ovog servisa

Deo izmenjenog hostovog konfiguracionog fajla će izgledati kao na slici 3.1.

```
    <services>
      <service name="MagicEightBallServiceLib.MagicEightBallService"
               behaviorConfiguration ="EightBallServiceMEXBehavior">
        <endpoint address=""
                  binding ="basicHttpBinding"
                  contract ="MagicEightBallServiceLib.IEightBall"/>
        <!--Omogućavanje MEX endpointa-->
        <endpoint address="mex"
                  binding="mexHttpBinding"
                  contract="IMetadataExchange"/>
        <!--Potrebno je dodati ovaj deo da bi MEX znao adresu servisa-->
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:8080/MagicEightBallService"/>
          </baseAddresses>
        </host>
      </service>
    </services>

    <!--Definisanje ponašanja MEX-a-->
    <behaviors>
      <serviceBehaviors>
        <behavior name="EightBallServiceMEXBehavior">
          <serviceMetadata httpGetEnabled="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Slika 3.1

Sada se može restartovati servis i pogledati meta podatke korišćenjem internet pretraživača. Dok je host i dalje pokrenut, potrebno je jednostavno ukucati u pretraživač sledeći URL: *http://localhost:8080/MagicEightBallService:*

Slika 3.2

Na početnoj strani WCF servisa, nalaze se osnovni detalji o načinu interakcije sa servisom programski, kao i načinu pregleda WSDL ugovora klikom na uokvireni link na vrhu strane. Slika 3.3 ilustruje WSDL ugovor.

```
- <wsdl:definitions name="MagicEightBallService" targetNamespace="http://tempuri.org/">
    <wsdl:import namespace="http://Intertech.com" location="http://localhost:8080/MagicEightBallService?wsdl=wsdl0"/>
    <wsdl:types/>
  - <wsdl:binding name="BasicHttpBinding_IEightBall" type="i0:IEightBall">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    - <wsdl:operation name="ObtainAnswerToQuestion">
        <soap:operation soapAction="http://Intertech.com/IEightBall/ObtainAnswerToQuestion" style="document"/>
      - <wsdl:input>
          <soap:body use="literal"/>
        </wsdl:input>
      - <wsdl:output>
          <soap:body use="literal"/>
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
  - <wsdl:service name="MagicEightBallService">
    - <wsdl:port name="BasicHttpBinding_IEightBall" binding="tns:BasicHttpBinding_IEightBall">
        <soap:address location="http://localhost:8080/MagicEightBallService"/>
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

Slika 3.3

4. Izgradnja WCF klijentske aplikacije

Sada, kada je host postavljen, konačan zadatak je napraviti softver koji će komunicirati sa WCF-om. Iako je moguće izgraditi potrebnu infrastrukturu „ručno“, .NET okruženje 3.5 podržava nekoliko pristupa za brzo generisanje proxy koda i konfiguracionog fajla klijentske strane aplikacije. Dodajmo nov projekat tipa *Consol Application* pod nazivom *MagicEightBallServiceClient*.

4.1. Generisanje proxy koda korišćenjem svcutil.exe alata

Korišćenjem svcutil.exe, može se generisati novi C# fajl koji predstavlja proxy kod i konfiguracioni fajl klijentske strane. Da bi se ovo uradilo, potrebno je jednostavno definisati *endpoint* kao prvi parametar. /*out*: je opcija koja se koristi za zadavanje imena C# fajla koji će sadržati proxy kod, dok /*config*: opcija specificira ime konfiguracionog fajla klijentske strane.

Potrebno je pokrenuti command prompt Visual Studio 2008 (Start -> Programs -> Microsoft Visual Studio 2008 -> Visual Studio Tools -> Visual Studio 2008 Command Prompt) i komandama ući u direktorijum *MagicEightBallServiceClient*. Pod pretpostavkom da je servis pokrenut, sledeća linija komandi će generisati dva nova fajla u radnom direktorijumu:

svcutil http://localhost:8080/MagicEightBallService /out:myProxy.cs /config:app.config

U myProxy.cs fajlu, nalazi se klijentska strana reprezentacije IEightBall interfejsa, kao i nova klasa pod nazivom EightBallClient, koja se i sama proxy klasa. Ova klasa je izvedena iz generičke klase, *System.ServiceModel.ClientBase<T>*, gde je *T* interfejs servisa. Na slici 4.1.1 je dat jedan deo proxy klase:

```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public partial class EightBallClient : System.ServiceModel.ClientBase<IEightBall>, IEightBall
{
    ...

    public string ObtainAnswerToQuestion(string userQuestion)
    {
        return base.Channel.ObtainAnswerToQuestion(userQuestion);
    }
}
```

Slika 4.1.1

Pri kreiranju instance za proxy tip, osnovna klasa će uspostaviti konekciju sa *endpoint*-om korišćenjem podešavanja specificiranih u konfiguracionom fajlu klijentske strane aplikacije. Kao i konfiguracioni fajl na serverskoj strani, i klijentski konfiguracioni fajl ima element <*endpoint*>. Element <*client*> postavlja ABC sa klijentske perspektive:

```
<client>
    <endpoint address="http://localhost:8080/MagicEightBallService"
        binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IEightBall"
        contract="IEightBall" name="BasicHttpBinding_IEightBall" />
</client>
```

Slika 4.1.2

Sada se ova dva fajla mogu uključiti u *MagicEightBallServiceClient* projekat (kome se dodaje referenca ka *System.ServiceModel.dll* sklopu) i može se koristiti proxy tip za komunikaciju sa udaljenim WCF servisom.

4.2. Generisanje proxy koda korišćenjem VS 2008

Kao i svaki alat komandne linije, svcutil.exe obezbeđuje veliki broj opcija za kontrolisanje načina generisanja klijentskog proxy-a. Ukoliko nisu potrebne ove naprednije opcije, mogu se ista dva fajla iz prethodne priče generisati korišćenjem Visual Studio 2008 IDE-a. Jednostavno se izabere Add Service Reference opcija iz menija Project. Zatim se unese URI servisa klikne Go dugme, kako bi se video opis servisa. Ovaj alat je dovoljan da bi referencirao WCF sklop automatski. Sada će kompletiran klijentski kod izgledati kao na slici 4.2.1

```
//Lokacija proxy-a
using MagicEightBallServiceClient.ServiceReference;

namespace MagicEightBallServiceClient
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("***Postavite pitanje***\n");

            using (EightBallClient ball = new EightBallClient())
            {
                Console.Write("Vase pitanje: ");
                string question = Console.ReadLine();
                string answer =
                    ball.ObtainAnswerToQuestion(question);
                Console.WriteLine("Odogovor je: {}", answer);
            }
            Console.ReadLine();
        }
    }
}
```

Slika 4.2.1

5. Dizajniranje WCF ugovora podataka

Prethodno objašnjeni WCF servisi definisali su vrlo jednostavne metode koje operišu sa primitivnim tipovima. Kada se koristi neki od servis orijentisanih tipova povezivanja (basicHttpBinding, wsHttpBinding,...), dolazeći i odlazeći podaci se automatski formatiraju u XML elemente korišćenjem *System.Runtime.Serialization.XmlFormatter* tipa definisanog u sklopu *System.Runtime.Serialization.dll*. Ako se koristi TCP-based povezivanje (npr. netTcpBinding), parametri i povratne vrednosti prostih tipova podataka se prenose u binarnom formatu.

Međutim, kada se definiše ugovor servisa koji koristi korisničke tipove kao parametre ili povratne vrednosti, ovi tipovi se moraju definisati korišćenjem ugovora podataka. Ugovor podataka je tip snabdeven atributom *[DataContract]*. Svako polje koje će se koristiti u ugovoru se označava atributom *[DataMember]*. Ako polje nije označeno atributom *[DataMember]*, neće biti serijalizovano. Na slici 5.1 dat je primer ugovora podataka:

```
[DataContract]
public class Korisnik
{
    [DataMember]
    public int ID;
    [DataMember]
    public string Ime;
    [DataMember]
    public string Adresa;
}
```

Slika 5.1

Aplikacija za razmenu obaveštenja korišćenjem netMsmqBinding povezivanja

6. Ideja i cilj korišćenja NetMsmqBinding-a

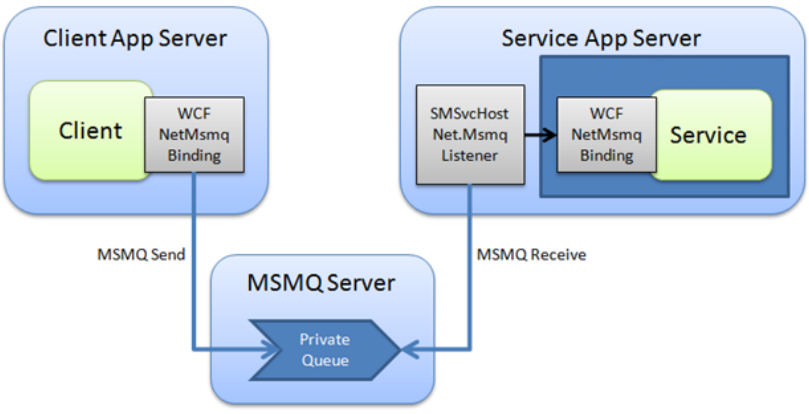
Pouzdana razmena poruka je ključni preduslov za izgradnju moćnih softverskih arhitektura koje omogućavaju konkurentno procesiranje podataka i ona je osnova za razvijanje aplikacija u realnom sistemu gde su mogućnosti za mrežno povezivanje sve veće, ali i gde je kvalitet povezivanja često nezadovoljavajući. U ovakvim uslovima potrebno je naći način da informacije dođu do korisnika nezavisno od kvaliteta mreže, kako važne informacije ne bi bile izgubljenje u transportu.

Redovi za poruke (Message Queues) omogućavaju asinhronu razmenu poruka, odnosno mogućnost da pošiljalac i primalac ne moraju da komuniciraju sa redom (queue-om) istovremeno. Poruke su skaladištene u redu dok ih primalac ne zatraži. Korišćenjem redova za poruke izbegava se situacija da zbog pada sistema dođe do gubitka informacija koje su poslate korisniku ili zahteva koje je korisnik poslao sistemu.

Neke prednosti uvođenja redova za poruke u aplikaciju su:

- Povećavanje robustnosti aplikacije, jer klijent može da pošalje poruku iako servis nije u funkciji
- Povećavanje skalabilnosti aplikacije, jer se može koristiti više instanci servisa kako bi se procesirale poruke preko samo jednog reda za poruke
- Eliminisanje potrebe klijenta da čeka odgovor sa servisa
- Smanjuje zavisnost klijenta i servisa, jer se sva komunikacija izvršava preko redova
- Omogućava se trajnost poruka, jer one mogu da prežive pad servisa.

Jedan od načina na koji WCF radi sa redovima za poruke je povezivanje korišćenjem netMsmqBinding - a. Na ovaj način se poruke automatski serijalizuju/deserijalizuju, stavljaju na red i skidaju sa reda, za razliku od korišćenja npr. http povezivanja, gde se ove akcije sprovode „ručno“ korišćenjem metoda za rad sa redovima. Na slici 6.1. prikazana je opšta arhitektura aplikacije koja koristi NetMsmqBinding.



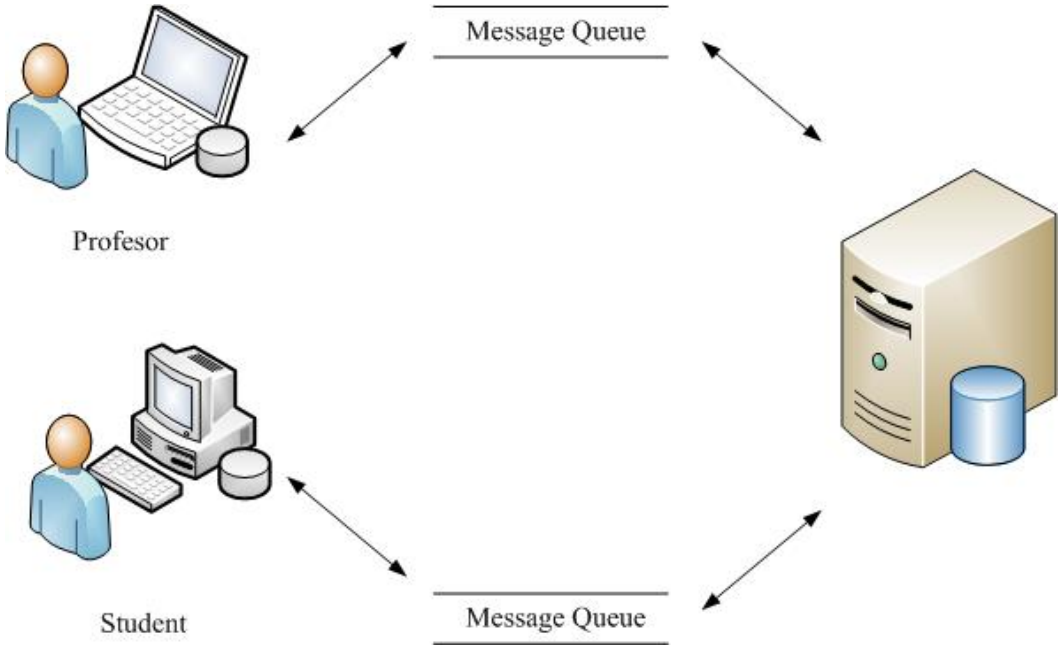
Slika 6.1

7. Arhitektura aplikacije za razmenu obaveštenja korišćenjem NetMsmqBinding-a

Problemi koji se odnose na pouzdano slanje i primanje poruka zastupljeni su i na univerzitetima. Sigurna razmena informacija između profesora i studenata u velikoj meri utiče na efikasnost izvršavanja obaveza obe strane. Na primer, korišćenjem redova za poruke, profesor prilikom slanja obaveštenja neće zavisiti od stanja servera, već može samom akcijom slanja biti siguran da će se obaveštenje naći u bazi podataka.

7.1. Network diagram

Na slici 7.1.1 prikazana je arhitektura aplikacije za slanje i primanje obaveštenja korišćenjem redova za poruke



Slika 7.1.1

Ideja aplikacije je da profesori i student mogu asinhrono da šalju, odnosno primaju obaveštenja iz baze podataka preko servera. Na ovaj način ni profesori ni student ne zavise od toga da li je server u funkciji.

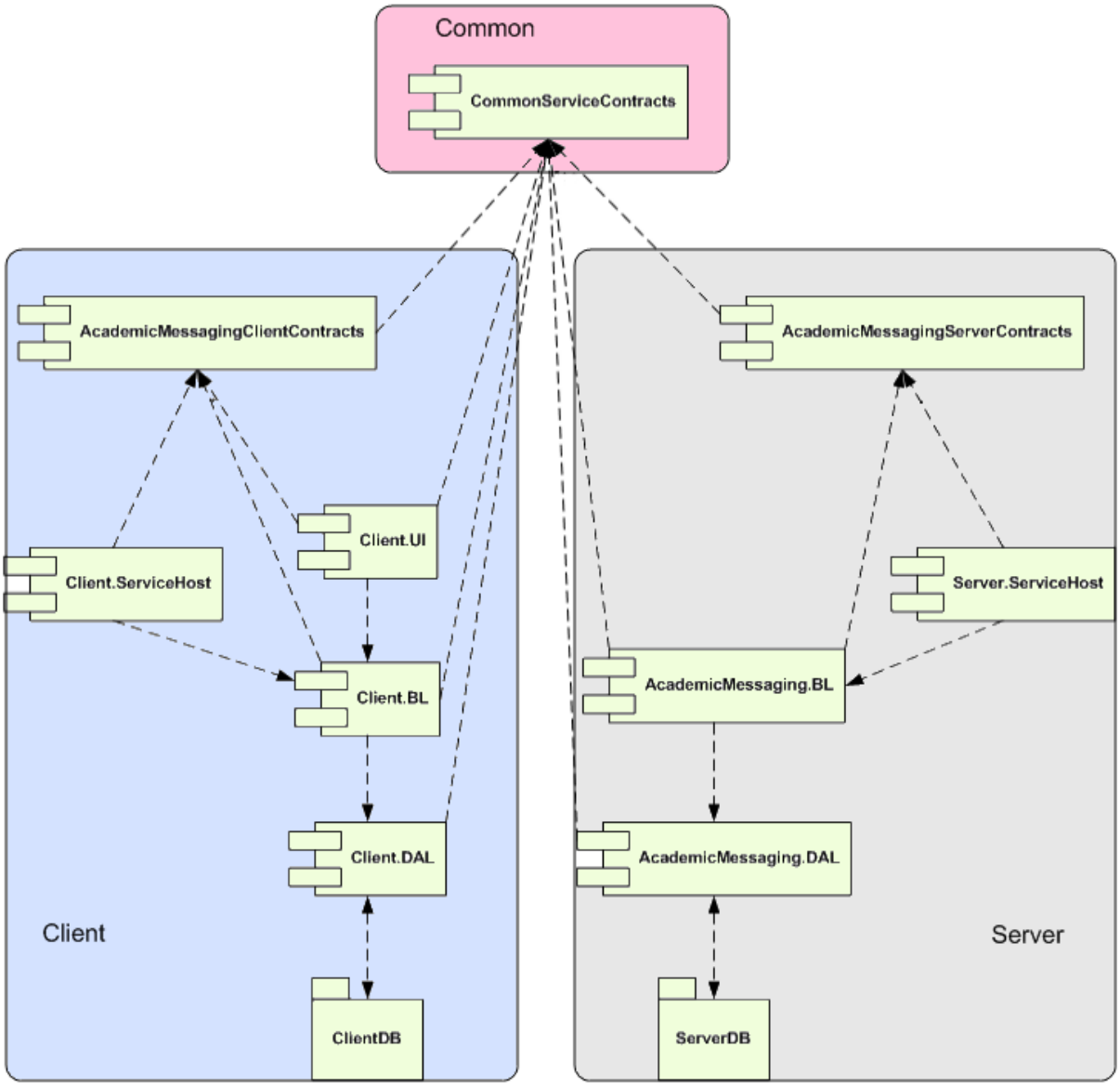
Klijent (nezavisno od tipa klijenta - profesor ili student) ima lokalnu bazu podataka u kojoj se nalaze obaveštenja koja su već primljena u nekom od prethodnih obraćanja globalnoj bazi. Na ovaj način postiže se veća brzina u dobijanju informacija, jer broj obaveštenja u bazi može biti jako veliki, što bi prouzrokovalo sporo dobijanje informacija pri svakom obraćanju servisu. Korišćenjem lokalne baze, traže se samo obaveštenja koja su svežija od poslednjeg primljenog obaveštenja za datog klijenta. U lokalnoj bazi klijenta nalaze se informacije i obaveštenja svih korisnika koji su se logovali i koristili tu klijentsku aplikaciju. Ako se korisnik ne nalazi u

lokalnoj bazi, iz globalne baze će se dobiti informacije o tom korisniku kao i sva obaveštenja za koje je korisnik prijavljen.

Ukoliko se klijent obrati servisu (radi slanja ili primanja obaveštenja), taj zahtev se skladišti na redu za poruke i ostaće na redu sve dok se zahtev ne kompletira.

7.2. Component diagram

Komponente sistema prikazane su na slici 7.2.1



Slika 7.2.1

Glavna karakteristika NetMsmqBinding-a je jednosmernost slanja poruka. To znači da klijent ne može da pozove metodu servisa koja ima povratne vrednosti. Već se u slučaju poziva metoda sa namerom da se dobiju neke povratne vrednosti, moraju napraviti dva jednosmerna puta (do baze i od baze). Iz tog razloga klijentski deo aplikacije se ponaša kao server u odnosu na serverski deo aplikacije. Odnosno, klijentski deo pruža servis serveru, a serverski deo pruža servis klijentu. Baza sa kojom komunicira serverski deo je globalna baza obaveštenja, a baza sa kojom komunicira klijentski deo aplikacije je lokalna baza aplikacije. Na taj način se pomoću dva jednosmerna poziva dva servisa simulira jedan poziv metode servisa sa povratnim vrednostima.

Aplikacija sadrži sledeće slojeve:

- **Prezentacioni sloj** (sa komponentom Client.UI) pomoću koga je predstavljen korisnički interfejs, implementiran u vidu windows formi
- **Servis klijenta** (sa komponentom Client.ServiceHost) koji omogućava vidljivost metoda koje se nalaze na klijentu
- **Poslovnu logiku klijenta** (sa komponentom Client.BL) koja sadrži implementaciju metoda definisanih u interfejsu klijenta. Interfejs klijenta je definisan u komponenti AcademicMessagingClientContracts
- **Sloj za pristup podacima DAL (Data Access Layer) klijenta** (sa komponentom Client.DAL) koji direktno komunicira sa lokalnom bazom klijenta
- **Servis servera** (sa komponentom Server.ServiceHost) koji omogućava vidljivost metoda koje se nalaze na serveru.
- **Poslovnu logiku servera** (sa komponentom AcademicMessaging.BL) koja sadrži implementaciju metoda definisanih u interfejsu servera. Interfejs servera je definisan u komponenti AcademicMessagingServerContracts
- **Sloj za pristup podacima DAL (Data Access Layer) servera** (sa komponentom AcademicMessaging.DAL) koji direktno komunicira sa glavnom bazom

8. Razvojno okruženje

Pri izradi aplikacije za razmenu obaveštenja korišćenjem NetMsmqBinding-a korišćene su sledeće tehnologije i razvojni alati:

- **.NET framework 3.5** - okruženje za razvoj aplikacija namenjenih za rad na Windows operativnim sistemima
- **Microsoft Visual Studio 2008** - razvojno okruženje za izradu desktop i web aplikacija, kao i windows i web servisa
- **Microsoft SQL Server 2005** - sistem za upravljanje relacionom bazom podataka koji je omogućio skladištenje obaveštenja i njihovo odpremanje iz baze podataka
- **Microsoft SQL Server 2005 Management Studio** - alat za razvoj baze podataka (kreiranje tabela)
- **Telerik WinForms kontrole** - kontrole za kreiranje formi za interakciju korisnika sa sistemom
- **Windows Communication Foundation** - servisno-orijentisan sistem razmene poruka koji je omogućio komunikaciju između klijentskog i serverskog dela aplikacije.
- **Message Queueing** - Windows komponenta koja je omogućila korišćenje redova za poruke

Rešenje je razvijeno korišćenjem programskog jezika C# i kombinacije jezika T SQL i LINQ za pisanje upita nad bazom podataka.

9. Implementacija rešenja

Rešenje je implementirano kao desktop aplikacija koja preko korisničkog interefjsa omogućava korisnicima asinhronu komunikaciju sa skladištem obaveštenja. Aplikacija radi sa transakcionim redovima.

Transakcioni redovi su redovi koji sadrže transakcione poruke, a transakcione poruke su poruke koje su prosleđene u okviru jedne transakcije.

Podrazumevano, MSMQ čuva poruke u memoriji dok se ne steknu uslovi da te poruke nastave predviđeni put. Ali ako MSMQ servis padne ili ako padne sistem koji drži poruke, velika je verovatnoća da će poruke biti izgubljene. Kao rešenje ovog problema, može se vrednost osobine *Recoverable* postaviti na „True“:

```
MessageQueue msgQueue = new MessageQueue(queueName);
msgQueue.DefaultPropertiesToSend.Recoverable = true;
```

Slika 9.1

Na ovaj način smo se pobrinuli da poruka ne bude izgubljena. Ali to nam ne garantuje da poruke neće biti duplirane. Dupliranje poruka je za pouzdanu aplikaciju nedopustivo isto koliko i gubitak poruka. Oba navedena problema uspešno rešava korišćenje transakcionih poruka.

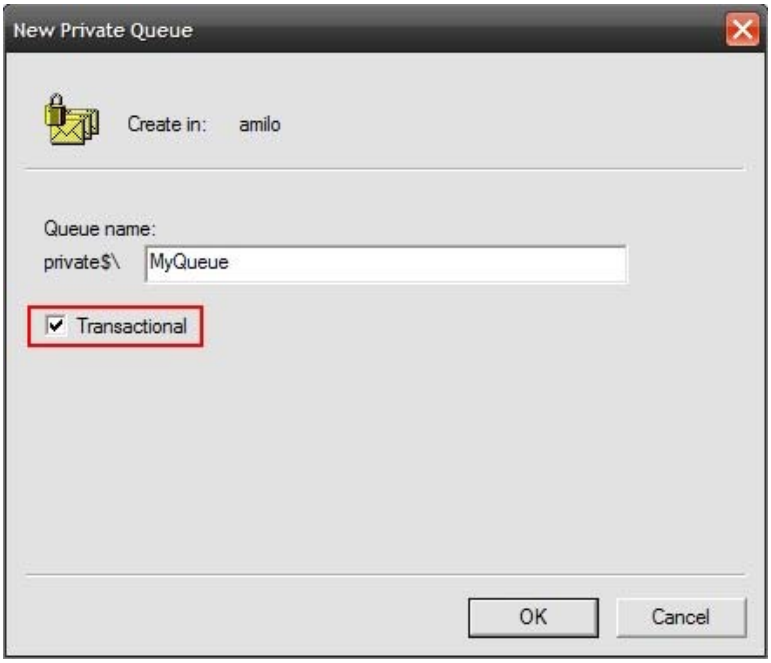
Prednosti transakcionih poruka su sledeće:

- Poruke će se proslediti tačno jednom u pravilnom redosledu
- Poruke su na disku, tako da neće biti izgubljene ako server padne
- Slanje i primanje poruka se mogu izvršavati u okviru transakcije. Ako se kreira jedna transakcija u okviru koje se izvršava primanje i procesiranje poruka, i dođe do pada servisa u toku procesiranja, cela transakcija će biti vraćena na početak (*roll back*). To znači da će se poruka vratiti na queue, pre nego da će biti izgubljena.

Mana transakcionih poruka su slabe performanse. Korišćenje transakcionih poruka je mnogo puta sporije od korišćenja netransakcionih.

Transakcione poruke mogu se kreirati na dva načina:

- U prozoru Computer Management (Start → My Computer → desni klik → Manage), u tree view kontroli doći do direktorijuma Private Queues (Services and Applications → Message Queues → Private Queues) i desnim klikom doći do opcije New → Private Queue. U prozoru za kreiranje reda, čekirati Transactional (Slika 9.2)



Slika 9.2

- U aplikaciji je kreiranje redova programsko, pa se samim tim i postavljanje osobine *Transactional* obavlja kroz program, kao na slici 9.3

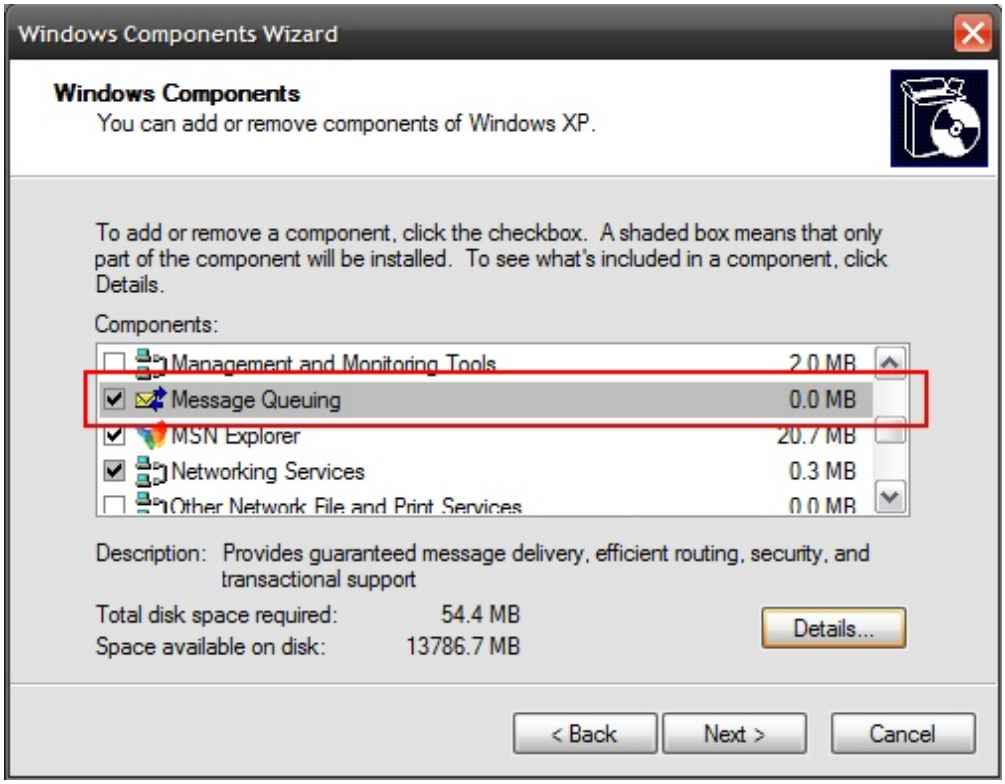
```
string queueName = @".\private$\Notifications";

if (!MessageQueue.Exists(queueName))
    MessageQueue.Create(queueName, true);
```

Slika 9.3

9.1. Instaliranje Message Queuing komponente

Kako bi aplikacija mogla da komunicira sa redovima za poruke, na sistemu mora biti instalirana Windows komponenta Message Queuing. Proces instalacije na Windows XP operativnom sistemu podrazumeva otvaranje prozora za dodavanje/uklanjanje Windows komponenti (Start → Control Panel → Add or Remove Programs → Add/Remove Windows Components (ili Alt + W)) i čekiranje komponente Message Queuing (slika 9.1.2)



Slika 9.1.2

9.2. Kreiranje solution-a i projekata

Prvi korak u razvijanju aplikacije je kreiranje novog solution-a u Visual Studiju i dodavanje projekata prikazanih na Component diagramu (slika 7.2.1). Projekat Client.UI je tipa Windows Forms Application, projekti Client.ServiceHost i Server.ServiceHost su tipa Console Application, dok su svi ostali projekti tipa Class Library. Projekti se međusobno referenciraju kako je opisano na Component diagramu.

9.3. Kreiranje ugovora

U projektu CommonServiceContracts definisani su svi DataContract-i koji se prenose preko servisa između baze i korisnika. U projektima AcademicMessagingClientContracts i AcademicMessagingServerContracts definisani su interfejsi sa potpisima metoda vidljivih serveru i klijentu aplikacije, tim redom.

Na slici 9.3.1 prikazan je jedan DataContract čiji objekat nosi informacije o obaveštenjima

```
namespace CommonServiceContracts.DataContracts
{
    [DataContract]
    public class Notification
    {
        [DataMember]
        public long IDNotification { get; set; }

        [DataMember]
        public string NotificationText { get; set; }

        [DataMember]
        public DateTime NotificationDate { get; set; }

        [DataMember]
        public string NotificationSubject { get; set; }

        [DataMember]
        public int IssueId { get; set; }
    }
}
```

Slika 9.3.1

Na slici 9.3.2 prikazan je ServiceContract klijentskog dela

```
namespace ClientAcademicMessagingContracts.ServiceContract
{
    [ServiceContract]
    public interface INotificationClient
    {
        [OperationContract(IsOneWay = true)]
        void ReceiveNotification(List<Notification> lstNotif);

        [OperationContract(IsOneWay = true)]
        void GetUserInformation(User user, List<int> lstIssueIds);
    }
}
```

Slika 9.3.2

9.4. Konfigurisanje WCF-a (klijent → server)

Centralni deo koda koji opisuje kako će aplikacija koristiti redove za poruke nalazi se u konfiguracionom fajlu ServiceHost projekata (sa klijentske i serverske strane). U tu svrhu su u projekte Client.ServiceHost i Server.ServiceHost dodati fajlovi tipa Application Configuration File u kojima su definisani endpoint-evi i bitna podešavanja za rad sa redovima za poruke. Sledeći kod prikazuje deo .config fajla u Server.ServiceHost projektu:

```
<services>
  <service name="AcademicMessaging.BL.NotificationServerBLImpl"
           behaviorConfiguration="ServiceBehavior">
    <endpoint address="net.msmq://localhost/private/Notifications"
              binding="netMsmqBinding"
              contract="ServerAcademicMessagingContracts.ServiceContract.INotificationService"
              bindingConfiguration="NotificationNetMsmqBinding"/>
    <endpoint address="mex"
              binding="mexHttpBinding"
              contract="IMetadataExchange"
              name="ServiceBehavior"
              listenUriMode="Explicit"/>
    <host>
      <baseAddresses>
        <add baseAddress="http://localhost:8080/NotificationServerBLImpl/" />
      </baseAddresses>
    </host>
  </service>
</services>
<bindings>
  <netMsmqBinding>
    <binding name="NotificationNetMsmqBinding"
             exactlyOnce="true"
             receiveErrorHandling="Fault">
      <security mode="None"/>
    </binding>
  </netMsmqBinding>
```

Slika 9.4.1

Naziv servisa koji se definiše u osobini *name* je naziv klase u kojoj su implementirane metode interfejsa i navodi se u obliku *Namespace.NazivKlase.*

Adresa endpoint-a se navodi u obliku net.msmq://localhost/... Bitno je naglasiti da se pri definisanju adrese ne navodi broj porta iza localhost. *private/Notifications* se odnosi na red za poruke pod nazivom notifications, koji će se naći među privatnim redovima za poruke. Kao način povezivanja navodi netMsmqBinding se, a pod ugovorom se podrazumeva interfejs sa potpisima metoda servera. Ime klase se navodi uz obavezno ime namespace-a.

bindingConfiguration je osobina čija vrednost je postavljena na ime *binding*-a u okviru *netMsmqBinding* taga ovog fajla. U okviru *netMsmqBinding* taga postavljaju se vrednosti osobinama koje definišu način rada sa redovima.

Postavljanjem vrednosti atributu *exactlyOnce* na „true“ govorimo WCF-u da radimo sa transakcionim redovima.

Osobina *receiveErrorHandling* se koristi za utvrđivanje kako će se upravljati „*lošim porukama*“. Loše poruke su bitan koncept vezan za transakcione poruke. Kao što je već rečeno, ako do greške dođe u toku procesiranja poruke, transakcija će se vratiti na početno stanje i poruka će se vratiti na red. Ako je do greške došlo zbog nekog trenutnog problema, poruka će uspešno biti procesirana pri sledećem pokušaju. Ali, ako je do greške došlo zbog npr. lošeg formata poruke, svaki sledeći put će biti problema sa njenim slanjem. WCF i MSMQ 4.0 pružaju podršku za detektovanje i upravljanje lošim porukama. Ako ista poruka ne može da se pošalje određeni broj puta (podrazumevano je 3 puta) smatraće se lošom porukom. Šta se dalje dešava sa porukom zavisi od vrednosti atributa *receiveErrorHandling.*

Vrednosti atributa *receiveErrorHandling* mogu biti:

- **„Move“**- automatski će se poruka staviti u podred „*poison*“, odakle se dalje sa njom može rukovati.
- **„Fault“** - poruka ostaje u redu i mora se ukloniti ručno.
- **„Drop“** - poruka se uklanja sa reda
- **„Reject“** - poruka se uklanja sa podreda „*poison*“

U primeru aplikacije, kako se i vidi na slici 9.4.1, vrednost ovog atributa je „*Fault*“.

Atribut security može imati šest vrednosti za *mode*: „None“, „Transport“, „Message“, „Both“, „TransportWithMessageCredentials“, „TransportCredentialOnly“. U aplikaciji se isključuje sigurnost, postavljanjem vrednosti na „None“.

Da bi klijent i server mogli uspešno da komuniciraju, potrebno je da atributi konfiguracionog fajla na klijentskoj strani imaju odgovarajuće vrednosti koje se podudaraju sa vrednostima atributa na serverskoj strani.

Kreiranjem proxy fajla pomoću svcutil alata komandne linije, kreiran je i konfiguracioni fajl klijenta u kome su postavljene vrednosti atributa vezanih za redove. Ove vrednosti se mogu proizvoljno menjati, ali treba obratiti pažnju da ključni atributi, koji su navedeni u konfiguracionom fajlu servera, moraju imati iste vrednosti kao odgovarajući atributi u konfiguracionom fajlu klijenta.

Na slici 9.5, prikazan je deo koda konfiguracionog fajla klijenta

```
<bindings>
    <netMsmqBinding>
        <binding name="NetMsmqBinding_INotificationService" closeTimeout="00:01:00"
            openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
            deadLetterQueue="System" durable="true" exactlyOnce="true"
            maxReceivedMessageSize="65536" maxRetryCycles="2" receiveErrorHandling="Fault"
            receiveRetryCount="5" retryCycleDelay="00:30:00" timeToLive="1.00:00:00"
            useSourceJournal="false" useMsmqTracing="false" queueTransferProtocol="Native"
            maxBufferPoolSize="524288" useActiveDirectory="false">
            <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
                maxBytesPerRead="4096" maxNameTableCharCount="16384" />
            <security mode="None">
                <transport msmqAuthenticationMode="WindowsDomain" msmqEncryptionAlgorithm="RC4Stream"
                    msmqProtectionLevel="Sign" msmqSecureHashAlgorithm="Sha1" />
                <message clientCredentialType="Windows" />
            </security>
        </binding>
    </netMsmqBinding>
</bindings>
<client>
    <endpoint address="net.msmq://localhost/private/Notifications"
        binding="netMsmqBinding" bindingConfiguration="NetMsmqBinding_INotificationService"
        contract="INotificationService" name="NetMsmqBinding_INotificationService" />
</client>
```

Slika 9.5

Pored već navedenih atributa, značajni su i *deadLetterQueue* i *durable* atributi.

Dead letter queue je red za poruke u koga se stavljaju poruke koje nisu poslate i čiji je životni vek istekao. Vrednosti koje može imati ovaj atribut su:

- **„None“** - ne upotrebljavaju se ovakvi redovi. Ako ne uspe slanje poruke, nikakav zapis o tome neće postojati. Ovo je podrazumevana vrednost ako je vrednost osobine *ExactlyOnce* postavljena na „false“
- **„System“** - korisiti se sistemski red za poruke za skladištenje ovakvih poruka. Ovo je podrazumevana vrednost ako je vrednost osobine *ExactlyOnce* postavljena na „true“
- **„Custom“** - red je specificiran u CustomDeadLetterQueue osobini koja se koristi za praćenje poruka sa nemogućnošću slanja.

Durable je atribut čije vrednosti pokazuju da li je poruka dugotrajna (ako je vrednost postavljena na „true“) ili kratkotrajna („false“) u redu za poruke.

9.5. Konfigurisanje WCF-a (server → klijent)

Kako je već rečeno, netMsmqBinding se može koristiti samo jednosmerno i u svrhu korišćenja dvosmernog poziva metoda servisa, klijent mora da igra ulogu servera kada se obaveštenja vraćaju iz baze. Zato se analogni postupak konfigurisanja WCF-a sprovodi u suprotnom smeru. Sada ulogu klijenta igra projekat AcademicMessaging.DAL, jer se sa njega vrši poziv servisa klijenta. Bilo je potrebno kreirati i ServiceHost klijenta sa analognim podešavanjima u njegovom konfiguracionom fajlu, kao i proxy fajl pomoću koga se pristupa metodama servisa klijenta.

9.6. Prikaz obaveštenja

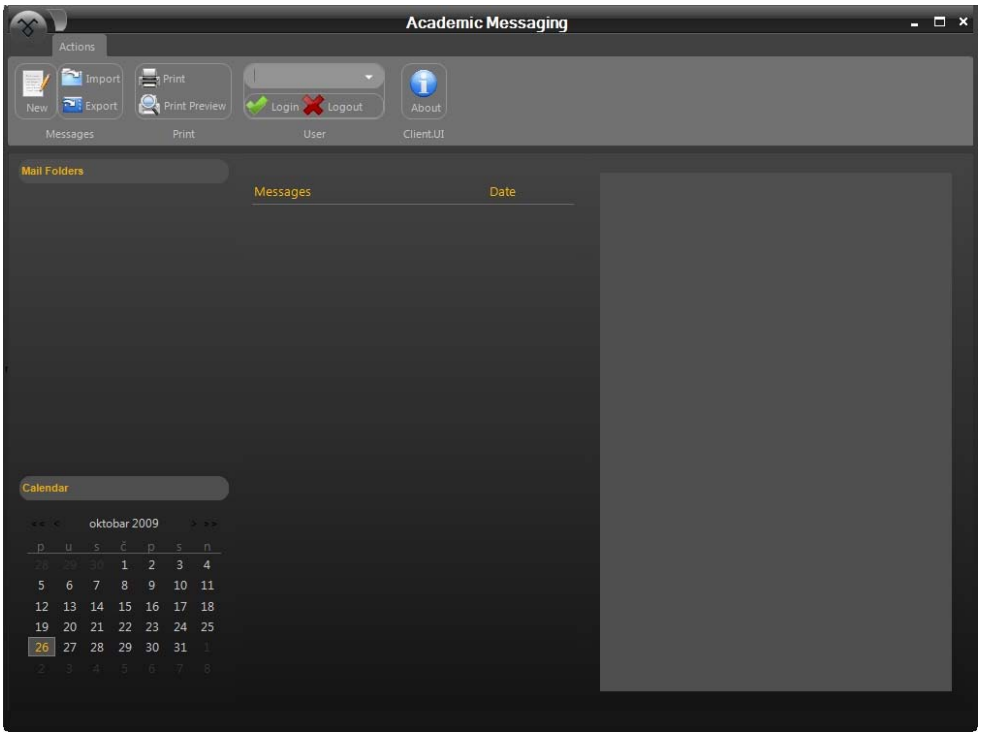
Kada se vrše asinhroni jednosmerni pozivi servisa, prekida se lanac koji čuva instancu forme koja je pozvala servis. Iz tog razloga ne postoji način da se obaveštenja koja se prosleđuju iz baze, prikažu direktno na formi klijenta. Zbog toga lokalna baza na klijentu ima još jednu svrhu, a to je skladištenje novodobijenih obaveštenja. Kako bi forma mogla da uzme obaveštenja iz lokalne baze, na formu je postavljen časovnik na čiji će svaki otkucaj da se proveri da li forma čeka na obaveštenje i ako čeka pročitaće se obaveštenja iz lokalne baze. Ova akcija ne utiče značajno na performanse sistema, jer je lokalna baza u stvari .mdf fajl koji se nalazi u samom solution-u.

9.7. Korisnički interfejs

Za izradu korisničkog interfejsa korišćene su Telerik kontrole kako bi izgled formi išao u korak sa dolazećim Office-om 2010.

9.7.1. Forme za prikaz

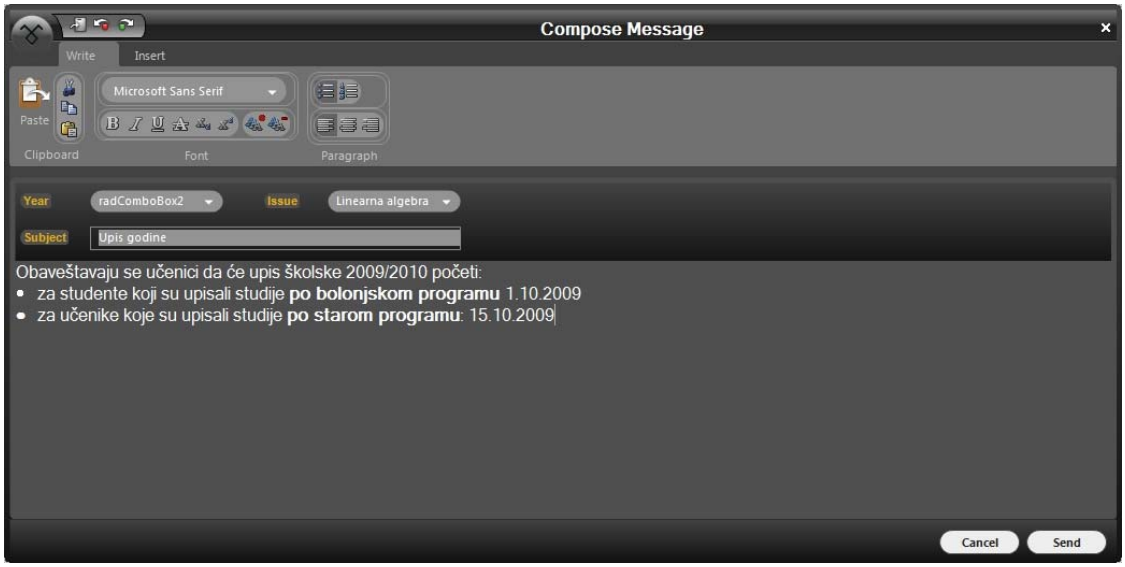
Na glavnoj formi nalaze se osnovne kontrole za lakše manipulisanje aplikacijom za razmenu obaveštenja. Na slici 9.7.1.1 prikazana je glavna forma



Slika 9.7.1.1

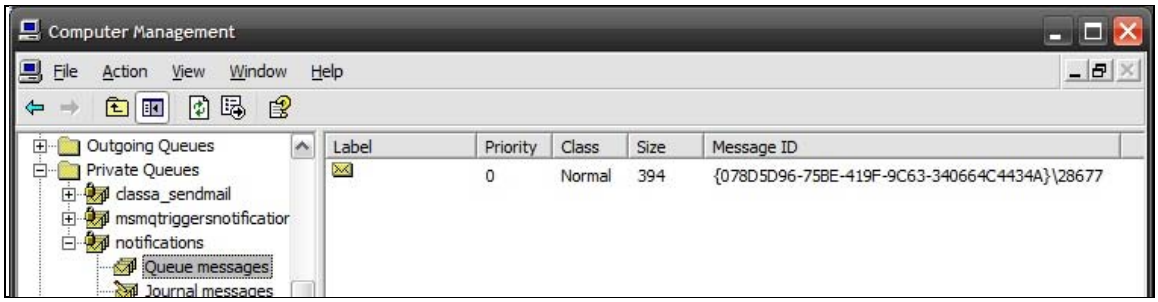
U drop down listi nalaze se imena svih korisnika čije su informacije u lokalnoj bazi. Ukoliko novi korisnik želi da se uloguje, dovoljno je da u drop down listu (u kojoj je omogućeno upisivanje), upiše svoje korisničko ime i klikne na login. Tada će se proveriti prava korisnika i u zavisnosti od toga da li je korisnik profesor ili student, omogućiće se slanje obaveštenja.

Ukoliko prijavljeni korisnik ima veća prava, on može i da šalje obaveštenja, klikom na dugme New i popunjavanjem forme za slanje obaveštenja (slika 9.7.1.3)



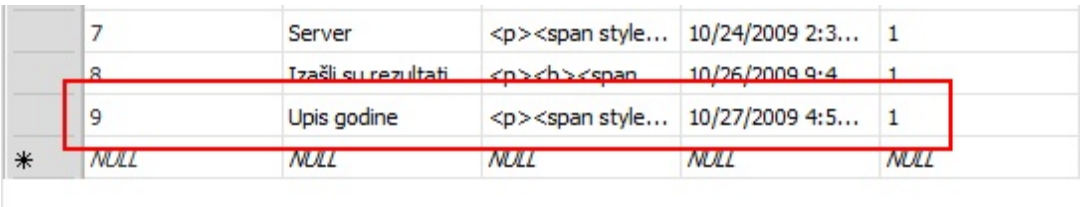
Slika 9.7.1.3

Ukoliko je servis isključen, a profesor klikne na *Send*, tada se može videti da u redu za poruke pod nazivom *notifications*, postoji poruka koja čeka da bude preuzeta (slika 9.7.1.4)



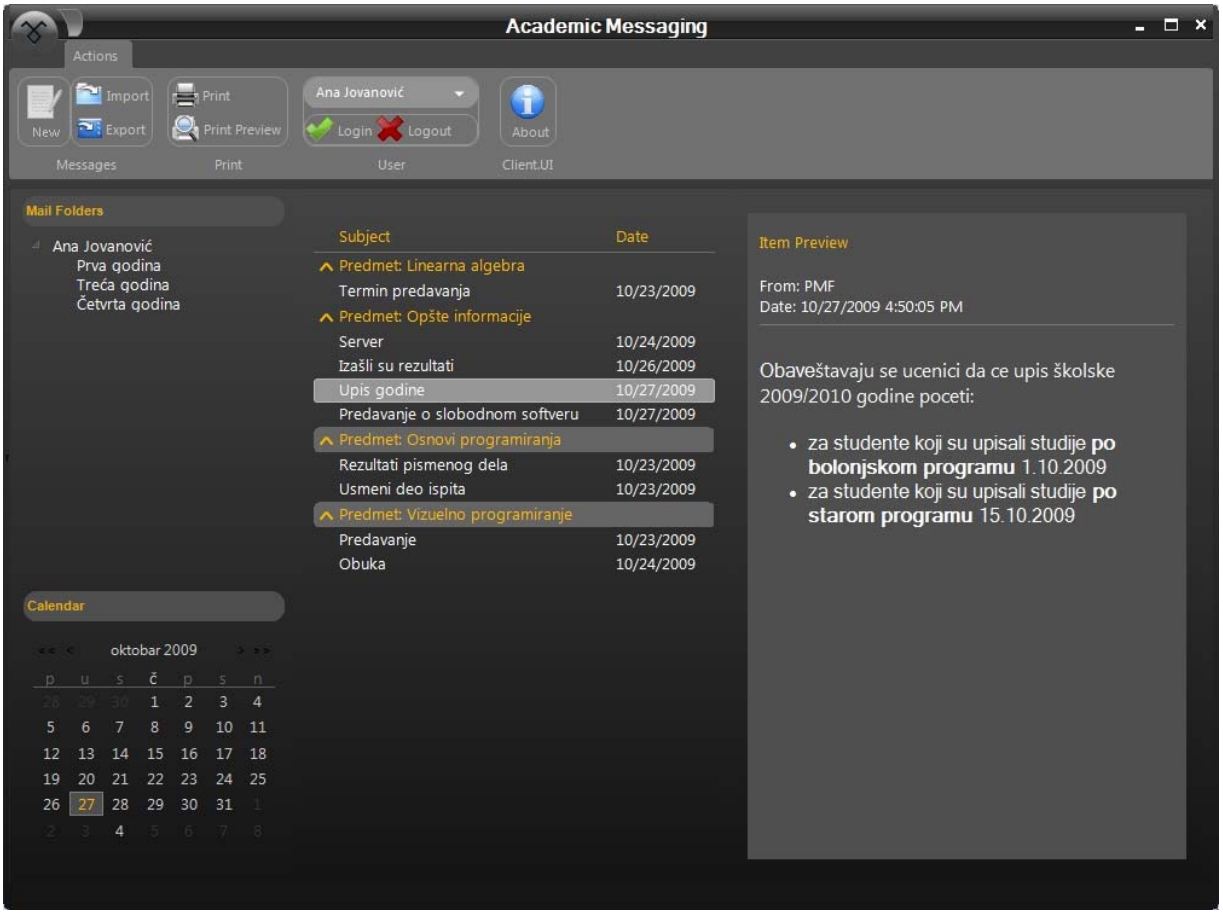
Slika 9.7.1.4

Kada se servis osposobi može se videti da poruke više nema na redu za poruke i da se nalazi u bazi podataka (Slika 9.7.1.5)



Slika 9.7.1.5

Ukoliko korisnik ima prava samo za čitanje obaveštenja, klikom na login, prikazaće sve sva obaveštenja iz predmeta za koje je student prijavljen da prima vesti.



Slika 9.7.1.2

U tree view kontroli ispod korisnikovog imena i prezimena nalaze se sve godine koje sadrže predmete na koje je korisnik prijavljen. Klikom na bilo koju od godina u tree view kontroli, poruke se filtriraju tako da se prikazuju samo one koje su iz predmeta koji pripadaju datoj godini. Klikom na bilo koje obaveštenje iz liste, sadržaj obaveštenja se prikazuje u formatu u kome je poslato. Prostor za prikaz obaveštenja je WebBrowser kontrola koja prepoznaje HTML tagove i na taj način se omogućava prikaz obaveštenja sa svim formatiranjima sa kojima je i poslato u bazu.

Zaključak

Windows Communication Foundation je veoma bogato tehnološko okruženje za izgradnju distribuiranih sistema korišćenjem univerzalnih zapisa poruka i kombinovanjem i proširivanjem mogućnosti distribuiranih API-a koji su mu prethodili, gradeći na taj način pouzdane univerzalne sisteme. WCF pojednostavljuje izradu servis-orijentisanih aplikacija kombinovanjem različitih tehnologija, dovodeći do najviših razvojnih standarda. Aplikacije razvijene korišćenjem WCF-a prevazilaze organizacijske i platformske granice.

Pored ovih prednosti, zadivljujuće je koliko je jednostavan, direktan i čist programski model i koliko fleksibilno se mogu sistemi konfigurisati. Zbog svega toga, WCF je jedan od najmoćnijih API-a za izgradnju distribuiranih sistema.

Literatura

- Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition - Andrew Troelsen
- Web service kurs - Spinnaker NT e-learning portal
- http://blogs.msdn.com/tomholl/archive/2008/07/12/msmq-wcf-and-iis-getting-them-to-play-nice-part-1.aspx
- http://blogs.msdn.com/tomholl/archive/2008/07/14/msmq-wcf-and-iis-getting-them-to-play-nice-part-3.aspx
- http://msdn.microsoft.com/en-us/library/ms752264.aspx
- www.google.com