VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

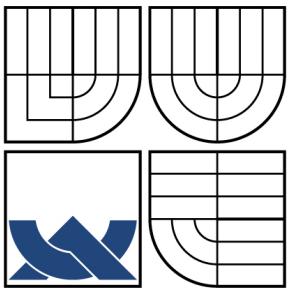
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

INTERNETOVÉ APLIKACE V .NET FRAMEWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

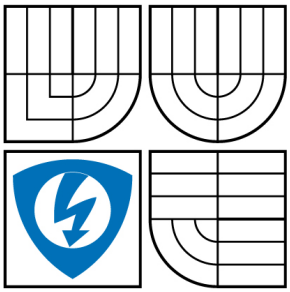
AUTOR PRÁCE MIROSLAV VOLOVEC
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

INTERNETOVÉ APLIKACE V .NET FRAMEWORK

INTERNET APPLICATIONS IN .NET FRAMEWORK

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

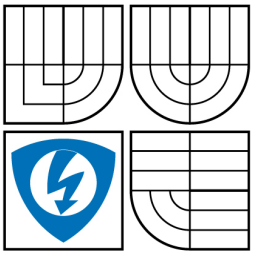
AUTOR PRÁCE — MIROSLAV VOLOVEC

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. IVO LATTENBERG, Ph.D.

SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Miroslav Volovec	***ID:***	97989
Ročník:	3	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Internetové aplikace v .NET Framework

POKYNY PRO VYPRACOVÁNÍ:

V jazyce C# pomocí vývojového nástroje Microsoft Visual Studio 2008 navrhněte a odlaďte projekt realizující pokročilý Instant Messenger s doprovodnými službami. Projekt koncipujte po dílčích částech, tak aby byl po přidání každé části funkční a bylo možné jej použít ve výuce jako vzorový projekt. Zdrojové kódy řádně okomentujte.

DOPORUČENÁ LITERATURA:

[1] PROSISE, J. Programování v Microsoft .NET, Nakladatelství Computer Press, a.s. 2003, 736 s., ISBN 8072268791
[2] BELLINASO M. Webové programování v ASP.NET 2.0 problém, návrh, řešení, Computer Press 2008, 648 s., ISBN 978-
80-251-1893-1
[3] LACKO L. ASP.NET a ADO.NET 2.0 Hotová řešení, Computer Press 2006, 385 s., ISBN 80-251-1028-1

Termín zadání:	29.1.2010	***Termín odevzdání:***	2.6.2010

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce ukazuje široké možnosti použití .NET Frameworku. Obsahuje popis tvorby systému odesílání rychlých zpráv. Popis tohoto systému zahrnuje tvorbu servisně orientované aplikace, webového klienta a nakonec i Windows aplikaci vytvořenou pomocí WPF. Cílem je ukázat možnosti tvorby softwaru pomocí inovativních postupů a nástrojů, které platforma Microsoft .NET nabízí.

KLÍČOVÁ SLOVA

.NET, Rychlé zprávy, databáze, aps.net, wpf, wcf, mssql

ABSTRACT

This work shows wide possibilities of .NET Framework usage. Describes development of instant messaging system. Description of this system includes production of service oriented application, web client and finally Windows application made with WPF. It's target is to show possibilities of software development using innovative methods and tools, that the Microsoft .NET platrofm offers

KEYWORDS

.NET, Instant Messaging, database, asp.net, wpf, wcf, mssql

VOLOVEC, Miroslav *Internetová aplikace v .NET*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2010. 45 s. Vedoucí práce byl doc. Ing. Ivo Lattenberg, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Internetová aplikace v .NET“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Tato práce se zabývá tvorbou systému pro posílání rychlých zpráv, tzv. Instant Messaging (IM). V práci je kladen důraz na použitelnost ve výuce, proto se zde zabývám širší paletou technologií postavených na platformě .NET Framework. Postupně procházím základy běhového prostředí, základ webových služeb, návrh databázového schématu v pokročilých modelovacích nástrojích, důležitost správného rozvrstvení systému a objektový návrh. Nedílnou součástí práce je představení nového trendu vývoje uživatelských rozhraní v technologii Windows Presentation Fundation, kde vysvětluji základní principy tvorby bohatých uživatelských prvků a celých aplikací v jazyce XAML.

1 ŘEŠENÍ STUDENTSKÉ PRÁCE

1.1 Úvod

Platforma .NET byla oficiálně představená firmou Microsoft v roce 2000. Microsoft .NET znamená novou generaci systému vývoje aplikací pro operační systémy Windows, a v dnešní době i Linux, viz projekt MONO (aktuálně ve verzi 2.6.3), založený na řízeném běhovém prostředí, obohaceném o neskromnou sadu základních tříd nesoucích jméno .NET Framework. Hlavní důvody vedoucí k vývoji technologie .NET jsou:

- Nekompatibilita jednotlivých programovacích jazyků, a s tím související obtížná spolupráce mezi programy/knihovnami napsanými v odlišných jazycích (např. C++ nebo Visual Basic atd. . .).
- Vysoká chybovost aplikací (chyby v práci s pamětí, neplatné konverze datových typů).
- Problémy s verzemi knihoven.
- Zastaralý a nepřehledný způsob vývoje dosavadních webových aplikací.

Všechny tyto problémy efektivně řeší platforma .NET – a to použitím zmíněného běhového prostředí, systémem assemblies, což jsou základní stavební prvky aplikací, a novou technologií ASP.NET pro vývoj webových aplikací.

1.2 Platforma .NET

1.2.1 Princip běhového prostředí

Většina dnešních aplikací, vytvořených například v jazyce C++, je zkompilována přímo pro danou platformu. To znamená, že zdrojový kód je kompilací přeložen do strojového kódu počítače. To ve výsledku přináší velmi dobrou rychlost běhu výsledné aplikace. Avšak na druhou stranu z toho plynou i některé nevýhody – nepřenositelnost mezi jednotlivými platformami, popřípadě verzemi operačních systémů a nezřídka jsou k vidění chyby v přístupech do operační paměti. Princip řízených běhových prostředí použitý právě u platformy .NET, ale i u velmi známe platformy Java firmy Sun Microsystems na to jde trochu jinak, a přidává k převodu zdrojového kódu do kódu strojového ještě jednu vrstvu. Tuto vrstvu představuje mezikód, do kterého jsou zdrojové kódy zkompilovány a tento mezikód je běhovým prostředím na cílové platformě (Windows, Linux) převeden do strojového kódu. Tento převod je na cílové platformě realizován vždy při spouštění konkrétní aplikace. Mínusem tohoto překladu je vyšší náročnost na výkon uživatelského počítače, a z tohoto důvodu se

tento způsob nepoužívá pro vývoj výpočetně náročných aplikací (např. počítačové hry). S jeho častým použitím se naopak můžete setkat spíše u obchodních aplikací, které přece jen nejsou tak náročné na výpočetní výkon, a rychlost běhu daných aplikací je naprosto vyhovující.

Poznámka: U spousty úloh (přístup k databázi, souborům atd.) uživatel snížení rychlosti aplikace ani nepocítí. Navíc je dobré dodat, že u těchto běhových prostředí při spuštění aplikace nedochází k překladu celé aplikace najednou, ale používá se JIT (Just-in-Time) kompilace. JIT kompilace znamená, že do strojového kódu je převedena pouze potřebná část mezikódu, a při opětovném použití této (již přeložené) části se spouští její zkompilovaná forma, což se příznivě projeví na rychlosti, která si již nezadá s během neřízeného programu.

1.2.2 Klíčové vlastnosti

Mezikód se ve světě platformy .NET nazývá MSIL, tedy Microsoft Intermediate Language. Tento jazyk relativních adres je spouštěn klíčovou součástí .NET frameworku pojmenovanou CLR (Common Language Runtime neboli společné běhové prostředí), jenž je standardizován organizací ECMA. V prostředí CLR existuje věc, která programátorům velmi usnadňuje práci s operační pamětí – Garbage Collector. Jedná se o sadu složitých algoritmů pro uvolňování nepotřebných programových objektů z paměti. Díky Garbage Collectoru se již vývojáři nemusí starat o přiřazování nebo uvolňování operační paměti, a odpadá tak riziko již zmíněné nekorektní práce s ní, která ve většině situací končí pádem aplikace.

Velmi důležitou vlastností, kterou dal Microsoft své platformě do vínku, je CLS – Common Language Specification (Společná jazyková specifikace). S ní souvisí CTS neboli Common Type System (společný typový systém). Výsledkem použití CLS a CTS je rovnocennost programovacích jazyků. Jinými slovy – pro vývoj .NET aplikací je možné použít jeden z několika programovacích jazyků vyšší úrovně. Může se jednat například o tyto :

- C#, jazyk vyvinutý přimo pro .NET
- Visual Basic .NET
- J#, jazyk se syntaxí rozšířeného jazyka Java
- Managed C++, managed označuje možnost psát řízený kód pro .NET dokonce i v původně „neřízeném“ jazyce jakým C++ od svého počátku je

1.2.3 Typy aplikací

Platforma Microsoft .NET vývojářům nabízí široké možnosti. Začít můžete u klasických konzolových aplikací, které pro vstup a výstup používají příkazový řádek.

Daleko zajímavější jsou aplikace s využitím knihoven Windows.Forms, interně využívající Microsoft Win32 API. Výsledkem jejich použití jsou známé formulářové aplikace pro Windows. Možné je také vytvořit aplikaci běžící jako proces na pozadí systému – službu Windows.

Dalším odvětvím jsou webové aplikace nahrazující zastaralé ASP 2.0; a to jejich nová generace označovaná jako ASP.NET, za kterou si dle mého názoru Microsoft zaslouží velkou pochvalu, protože tím posunul tvorbu dynamických webů o pořádný kus dál.

Neméně zajímavým typem aplikací jsou takzvané Webové služby, které umožňují pomocí všudypřítomného http protokolu na vzdáleném serveru volat metody.

1.3 Aktuální stav platformy .NET a součásti použité při řešení úlohy

Před zhruba dvěmi lety byla uvedena v mnoha ohledech revoluční verze .NET Framework s číslem 3.5. Tato verze přinesla mnoho nových vylepšení, které programátorům opět více zefektivní a zjednoduší práci. Zmíním zde zejména ty, které jsem použil při realizaci projektu.

Asi největším lákadlem je bezesporu technologie LINQ (Language INtegrated Query), která přišla s uvedením C# verze 3.0. LINQ přináší podporu přímého dotazování nad kolekcemi objektů podobně jako v databázích. Kolem LINQ vzniklo hned několik dalších projektů a rozšíření vyvíjených, ať už přímo společností Microsoft, tak i komunitou. Z dílny Microsoft bych zmínil objektově relační mapovací nástroj zvaný Linq2Sql. Díky Linq2Sql je možné z MSSQL databáze vygenerovat sadu objektů odpovídajících 1:1 tabulkám v databázi. Pak je možné dotazovat se databáze v jazyce C#(nebo Visual Basic 9) a pracovat s daty čistě objektově, včetně dotazů.

Další neméně důležitou technologii je Windows Communication Fundation (dále jen WCF). Tuto technologii sebou .NET Framework nese již z verze 3.0. WCF je jednotný framework pro vytváření servisně orientovaných aplikací. Základním stavebním kamenem WCF jsou služby. Službu si lze představit jako samostatný systém, který komunikuje s okolím pomocí tzv. endpoints, neboli koncových bodů.

Ani vývoj uživatelských rozhraní nezůstal pozadu, a tak Microsoft uvedl technologii Windows Presentation Fundation (dále jen WPF). Tato technologie umožňuje psát vzhledově bohaté formuláře a rozhraní pomocí definice v jazyce XAML (jeden z jazyků z rodiny XML). Tento jazyk je vzdáleně podobný html respektive ASP.NET.

Vývoj aplikací pro v .NET by nebyl z daleka tak efektivní nebýt integrovaného vývojového prostředí Microsoft Visual Studio, nyní ve verzi 2010, které bylo oficiálně vypuštěno před několika týdny. Toto prostředí jsem samozřejmě použil k řešení mé práce.

1.4 Jednoduchý ASP.NET http handler pro zpracování základních matematických operací

Cílem této úlohy je vytvořit jednoduchý webový handler* zpracovávající data předávané pomocí URL. V našem případě půjde o handler realizující základní matematické operace (plus – sčítání, minus – odčítání, nasobeni – násobení).

**Http handler v ASP.NET je autonomní funkční modul pro zpracovaní požadavků.*

1.4.1 Vytvoření nového projektu

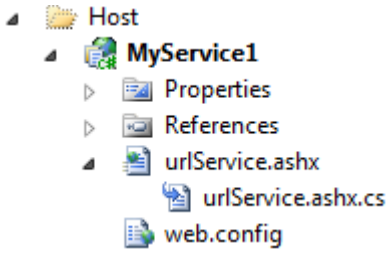
Jak již bylo v úvodu výše, zmíněno pro vývoj použijeme Microsoft Visual Studio 2008. Nejprve si vytvoříme novou *solution* (typ projektu VS2008, který sdružuje ostatní projekty) *File →New Project*. Toto bude výhodné pro organizaci projektů zejména v druhé úloze. Zde zvolíme část stromu *Other Projects Type*, v niž se nachází naše hledaná Visual Studio Solutions.

V nové *solution* si vytvoříme nový *solution directory*. Toto provedeme stiskem pravého tlačítka myši nad ikonkou naší *solution* a následným *add →New Solution Directory*, a nazveme ho HOST. Obdobným způsobem lze vytvářet adresáře i v jednotlivých projektech a pomocí „add“ lze vkládat i nové projekty do *solution* nebo soubory do projektů.

Do složky HOST vložíme nový projekt, který najdeme v *Visual C# →Web →WCF Service Aplication*. Momentálně by ani nebylo nutné vytvářet WCF Service Aplication, ale s ohledem na náplň následujících úloh je vhodné mít projekt již vytvořený.

1.4.2 Vytvoření handleru

Do vytvořeného projektu nyní vložte nový soubor *Web →Generic Handler*. Měl by to být soubor s příponou **.ashx*. Toto je náš *http handler*, který bude realizovat matematické funkce. Ostatní soubory nacházející se v projektu budeme v rámci této úlohy ignorovat. Po vytvoření souboru by měl obsah *Solution exploreru* (výchozí pozice vpravo nahoře) vypadat jako na obrázku 1.1.



Obr. 1.1: Strom projektu

Nyní si otevřeme soubor *http handleru*. VS2008 automaticky vygenerovalo jakousi základní kostru souboru. Tato kostra po dotazu na handler vypíše text „*hello World*“ jako výstup do webového prohlížeče. My ale chceme, aby náš handler uměl počítat. Webový handler je vlastně jakýsi chudý příbuzný ASP.NET webové stránky. Přijímá data pomocí protokolu http a vrací výsledek. Vstupní data mu můžeme poslat pomocí metody POST http protokolu nebo, co je náš případ, předáme vstupní hodnoty pomocí url. Zdrojový kód handleru upravíme a výsledek bude tento:

```
namespace MyService1
{
    public class urlService : IHttpHandler
    {

        public void ProcessRequest(HttpContext context)
        {
            string action = context.Request["action"];
            int a, b;
            switch (action)
            {
                case "plus":
                    {
                        if (int.TryParse(context.Request["a"], out a)
                                && int.TryParse(context.Request["b"], out b))
                        {
                            context.Response.Write("Vysledek=" + (a + b));
                        }
                    } break;
                case "minus":
                    {
                        if (int.TryParse(context.Request["a"], out a)
                                && int.TryParse(context.Request["b"], out b))
                        {
                            context.Response.Write("Vysledek=" + (a - b));
                        }
                    } break;
                case "nasobit":
                    {
                        if (int.TryParse(context.Request["a"], out a)
                                && int.TryParse(context.Request["b"], out b))
                        {
                            context.Response.Write("Vysledek=" + (a * b));
                        }
```

```
                } break;
            default:
                break;
        }
    }

    public bool IsReusable
    {
        get
        {
            return false;
        }
    }
}
```

Nyní zkusíme náš projekt spustit. Provedeme to kliknutím pravého tlačítka myši na ikonu projektu. V nabídce vybereme *Debug →Start New Instance*. S největší pravděpodobností vyskočí dialog, který se zeptá na vytvoření souboru `web.config`, toto odsouhlasíme. Poté by se měl automaticky spustit lokální, do VS2008 integrovaný, webový server, na kterém bude umístěn náš handler. Měl by mít adresu např. `http://localhost:1202` . Automaticky by se měl také otevřít webový prohlížeč, ve kterém uvidíme obsah adresáře, kde se nachází náš handler.

Nyní je vše připraveno k otestování handleru. Do prohlížeče zadáme adresu a cestu k souboru handleru, např. `http://localhost:1202/urlService.ashx`. Handler můžeme ovládat pomocí parametrů předávaných v url. Parametry k ovládání jsou následující:

- Action
  - Plus – sčítání dvou čísel
  - Minus – odčítání dvou čísel
  - Nasobeni – násobení dvou čísel
- A - parametr v němž bude hodnota prvního čísla
- B – hodnota druhého čísla

Kdo se už setkal s webovými aplikacemi, bude jistě vědět jak se hodnoty do url zapisují, pro ty ostatní je tato url:

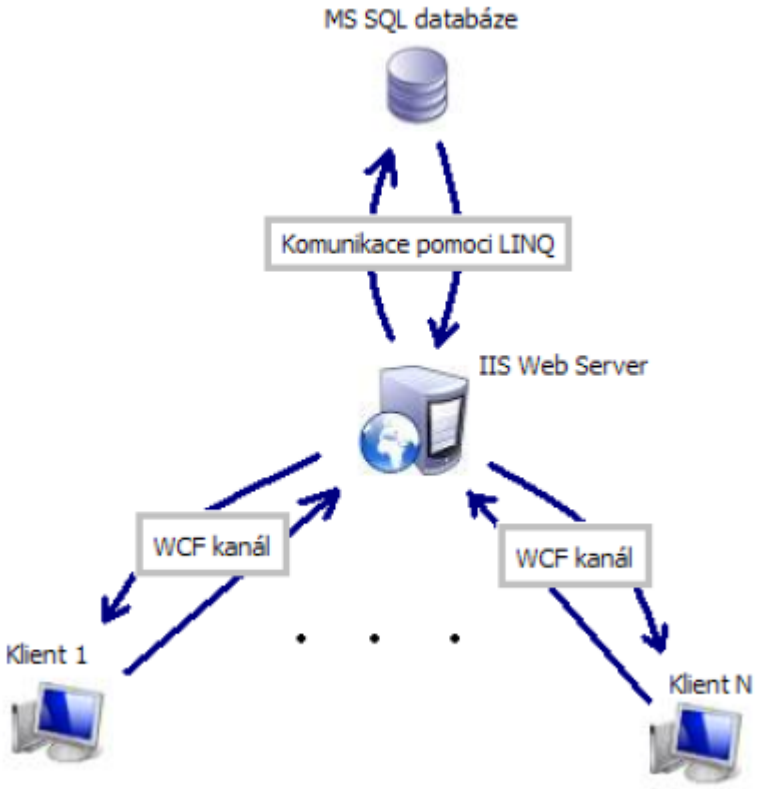
```
http://localhost:1202/urlService.ashx?action=plus&a=5&b=6
```

Modifikací této url můžete měnit akci, kterou má handler s čísly provést, nebo měnit hodnoty čísel samozřejmě.

Cílem úlohy byla ukázka velmi jednoduché „služby“ (nejednalo se o službu v pravém slova smyslu, ale *http handler* se do jisté míry službě podobá), která provádí matematické operace. Dále také seznámení s prostředím Microsoft Visual Studia a základní seznámení s tvorbou nových projektů.

1.5 Instant Messaging systém

Instant Messaging systém (dále jen IMS) je z angličtiny převzatý název pro systém poskytující službu rychlého odesílání zpráv v rámci internetu. IMS známe několik např. ICQ, MSN, Yahoo, Jabber atd. Náš systém bude navenek fungovat velmi podobně, ale v některých zásadních věcech se bude lišit. Tím nejzásadnějším rozdílem bude komunikace pomocí protokolu HTTP. Jádrem celého systému bude databáze, jenž bude sloužit jako prostředník předávání zpráv, a zároveň bude uchovávat všechny potřebné informace o uživatelích. Další součástí systému bude webová služba komunikující pomocí HTTP s klientskou aplikací a na serveru s cílovou databází. Vizuální pohled na systém je na obrázku 1.2.



Obr. 1.2: Struktura systému

1.5.1 Databáze

Základem každého informačního systému je databáze. Náš IMS je v podstatě také informační systém jako každý jiný. Slouží pro správu a šíření informací nimiž IM zprávy jistě jsou.

Jelikož pracujeme s technologiemi .NET budeme používat databází MSSQL. Toto řešení má několik zásadních výhod. Technologie MS na sebe dobře navazují. Můžeme tak použít integrovaný designer LinqToSql jako základ pro datovou vrstvu nad databází.

Pro náš systém potřebujeme v základu jen 3 nejdůležitější tabulky. Tabulka pro uživatele, tabulka pro zprávy a vazební tabulka 1:N. Použitá databáze, je ale o něco složitější. Není totiž nic horšího než zanedbání návrhu databáze. Systém se pak rozvíjí velmi obtížně. Proto zvolíme komplexnější databázový model, který počítá s pozdější implementací rozšiřujících funkcí jakými jsou tvorba uživatelských skupin, seznamy kontaktů atd. Modelování jsem prováděl v Toad Data Modeleru. Tabulky modelované v tomto nástroji jsou na obrázku 1.3. Následně se pak z modelu vygeneruje T-SQL výstup, který spustíme v cílové databázi, a tím se nám vytvoří vše potřebné.

Jak už bylo zmíněno, tak základní datovou vrstvu nad databází vytvoříme pomoci designeru LinqToSql ve VS2010. Nad tímto objektovým modelem je ale vhodné napsat si statickou třídu pro práci s daty.

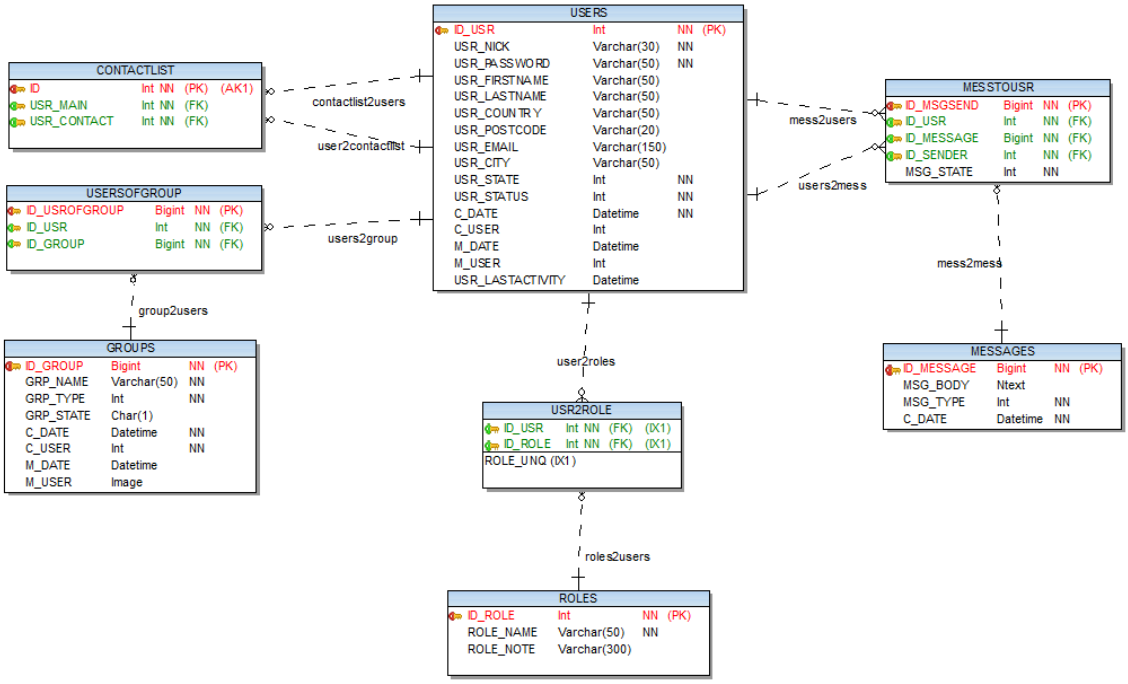
Vytvoříme tedy dva další projekty. Jeden bude obsahovat LinqToSql objekty a *.dbml soubor (database markup language), což je vlastně XML popis tabulek a vazeb, ze kterého VS2010 vygeneruje třídy. Druhý projekt bude už obsahovat naší datovou vrstvu, kterou bude využívat WCF aplikace. Tím budeme mít ideálně rozvrstvenou architekturu. Použití takto rozvrstvené architektury má výhodu v tom, že pokud bychom měnili cílovou databázi nebo chtěli použít místo LinqToSql například Entity Framework, tak promítneme změny pouze do nejnižších vrstev, a tím pádem nemusíme už přepisovat WCF aplikaci.

Za zmínku určitě stojí samotná práce s databází pomocí Linq. V ukázce 1.1 vidíme vytvoření data kontextu a následné „vytažení“ dat z databáze. Vše probíhá objektově, a v každém okamžiku jsou kontrolovány typy proměnných. LinqToSql následně tyto programové dotazy překládá do SQL jazyka. Tento přístup má několik nesporných výhod. Jako první bych uvedl větší transparentnost díky kontrole typů. Nemůže se nám tedy stát, že bychom omylem do dotazu přiřadili nesprávnou proměnnou. Kompiler totiž zahlásí ihned chybu a program nezkompiluje. Další výhodou je nemožnost tzv. „SQL Injection“ útoků. Princip takových útoků spočívá v nalezení vstupu aplikace, který se promítne do konečného dotazu. To se nám ovšem nestane, protože neskládáme dotaz do řetězce, ale máme jej kompilovaný v kódu.

```
MessDataContext db = new MessDataContext();

var messages = from m in db.MESSAGEs
               where m.C_DATE == DateTime.Today
```

```
select m;
```

Kód 1.1: Vytvoření data kontextu a dotaz nad databází



Obr. 1.3: Databázový model

1.5.2 Webová služba

Jak už bylo dříve zmíněno, jako komunikační bod systému budeme používat webovou službu, konkrétně WCF service. V dnešní době je stále větší poptávka po servisně orientovaných aplikacích. Pod pojmem „servisně orientovaná aplikace“ chápejme aplikaci, která volá metody implementované na serveru. Toto rozdělení je v mnohých případech velmi výhodné, někdy zcela nutné. Nutností je i v našem případě.

Základní myšlenka při vzniku WCF je SOA. SOA je zkráceně Servisně orientovaná architektura. WCF je framework, kterým vytváříme SOA aplikace. Sjednocuje hned několik starších technologii pro tvorbu těchto aplikací. Jde vlastně o komunikaci aplikace s aplikací, přesněji služby s klientem. Konkrétně je WCF vlastně distribuovaná aplikace.

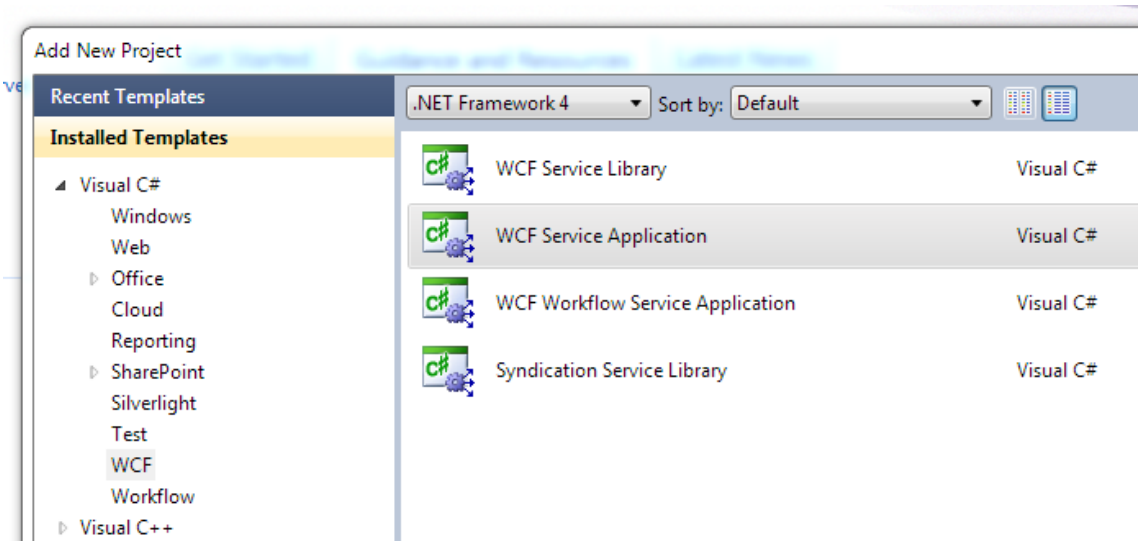
V minulosti jsme pro vytváření SOA aplikací měli několik možností. Malý výčet: ASP.NET Web Services, Web Services Enhancements(WSE), .NET Remoting atd. Programátor použil pro specifickou aplikaci vždy tu možnost, která byla pro daný problém nejlepší. Tyto technologie se hodily každá na něco jiného. Vývoj aplikací

byl závislý na dané technologii což je do jisté míry nevýhoda. Další nevýhodou je vzájemná nekompatibilita těchto technologií.

A co je to vlastně WCF? WCF sjednocuje dřívější technologie. Programátor nemusí znát do puntíku použitou komunikační technologii. Stačí znát a vědět jak používat právě WCF.

Jeho použití v praxi znamená obrovské úspory času a zjednodušení práce vývojářům.

Vytvoření WCF služby je vcelku jednoduché. Do projektu stačí vložit přednastavený typ projektu z nabídky VS2010. Konkrétně to bude „WCF Service Application“ obrázek 1.4. Následně pak definovat rozhraní (interface) a metody tohoto rozhraní následně implementovat. Metody rozhraní musí mít navíc nastavené speciální parametry, viz kód 1.2 aby bylo jasné, že se jedná o WCF. Kromě speciálních parametrů WCF v rozhraní musíme také definovat takzvané data kontrakty (DataContract). DataContract je jednoduchá třída, která obsahuje soubor dat, které chceme pomocí WCF přenášet. Jejich definice viz kód 1.3.



Obr. 1.4: Dialog vložení nového projektu

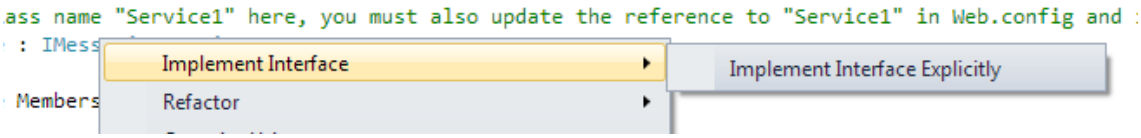
```
[ServiceContract]
public interface IMessagingService
{
    [OperationContract]
    int User_RegisterNewContact(string nick, string password);
    ...
}
```

Kód 1.2: Ukázka speciálních parametrů při definici rozhraní

```
[DataContract]
public class ImMessage
{
    [DataMember]
    public int ID_SENDER { get; set; }
    [DataMember]
    public int ID_USR { get; set; }
    ...
}
```

Kód 1.3: Definice data kontraktu

Implementace rozhraní pak už není nic složitého. Když otevřeme soubor naší WCF aplikace (*.svc), tak by nám VS2010 mělo rovnou otevřít soubor se samotným kódem. Třída naší WCF aplikace je zděděná z rozhraní. Takže stačí pouze kliknout pravým tlačítkem na identifikátor rozhraní v kódu, a zvolit „Implement interface“. Vývojové prostředí nám vygeneruje prázdné metody definované rozhraním, do nichž pouze doplníme požadovanou funkčnost. Tato funkčnost spočívá hlavně v manipulaci s daty.



Obr. 1.5: Funkce implementace metod z rozhraní

1.6 Klientská aplikace

Služba kterou jsme vytvořili je, co se komunikace s ní týče, univerzální napříč všemi aplikacemi, které lze v .NET Framework vytvořit. Základem pro komunikaci s ní je vygenerování proxy tříd na základě metadat, která služba poskytuje. Prostředí Visual Studia 2010 poskytuje nevídaný komfort při používání technologií jakou je mimo jiné právě WCF. Proxy třídy nemusí programátor psát ručně, ale s pomocí VS2008 si je může automaticky vygenerovat. Pro efektivní práci je toto automatické generování proxy tříd víc nutností než možností.

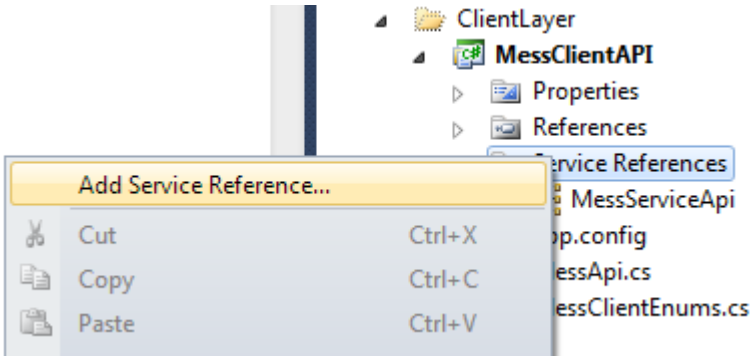
1.6.1 Klientská vrstva pro komunikaci s webovou službou

Než začneme tvořit konkrétní klientskou aplikaci, bude dobré si vytvořit projekt, který bude obsahovat statickou třídu pro práci s metodami webové služby a také referenci na webovou službu, a tím i proxy třídy.

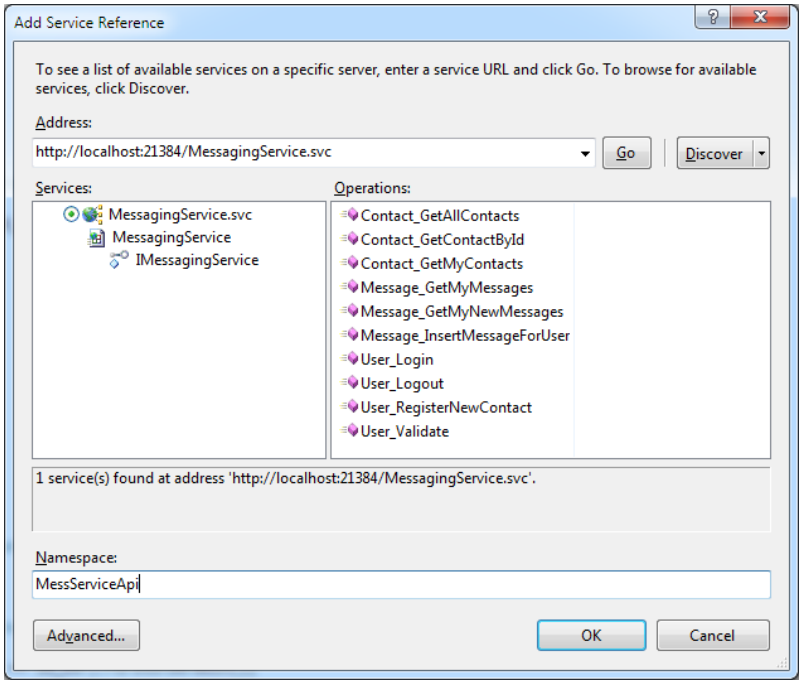
Tato další vrstva je výhodná, protože při práci s WCF kanálem je dobré tento kanál „uzavírat“.

Do prázdného projektu přidáme referenci na webovou službu. To uděláme jako na obrázku 1.6. Následně pak v dialogu (obrázek 1.7) zadáme adresu naší webové služby a vyplníme název namespace. Je možné využít i možnosti Discover, která prohledá dostupné webové služby na lokálním počítači. Po stisknutí tlačítka „OK“ proběhne generování již naprosto automaticky.

Pokud generování proběhlo v pořádku, stačí projekt přeložit a můžeme tuto vrstvu používat v libovolné klientské aplikaci.



Obr. 1.6: Přidání service reference - kontextové menu



Obr. 1.7: Přidání service reference - nalezení serveru a vytvoření proxy tříd

1.7 Webová klientská aplikace v ASP.NET

Jak už jsem výše zmínil k WCF službě může přistupovat téměř libovolná aplikace. Dobrým demonstračním příkladem je ASP.NET aplikace.

1.7.1 Co je to ASP.NET

ASP.NET je součást .NET Frameworku a nástupce starší technologie ASP (Active Server Pages) vyvinutá jako přímá konkurence pro JSP (Java Server Pages). V ASP.NET můžou programátoři realizovat své projekty v kterémkoli jazyce spadajícím do .NET Frameworku. Díky předkompilování jsou také ASP.NET stránky rychlejší. Toto však neplatí vždy. Rychlejší běh se projeví až při složitějších operacích.

Koncept ASP.NET WebForms je v mnoha ohledech velmi podobný Windows-Forms. Tato podobnost je záměrná. To proto aby vývojáři aplikací pro Windows měli snazší přechod k programování pro web. Stránky se skládají z ovládacích prvků (Controls), které jsou odvozeny od ovládacích prvků pro Windows.

Jednou z vymožeností ASP.NET je také stavové prostředí. Ačkoli se to může zdát Windows programátorům jako samozřejmost, tak ve skutečnosti je to jinak. Jak jistě víme, tak protokol HTTP je bezstavový. ASP.NET dovoluje událostmi řízené programování díky uchování stavu (aktuálním stavu stránky a jejich ovládacích prvků) mezi jednotlivými požadavky pomocí dvou způsobů:

- **ViewState** uchovává informace mezi opakovaným odesíláním formuláře na server zašifrovaně ve skrytých polích na HTML stránce. Výhoda této techniky je, že není třeba nastavovat žádnou podporu na straně serveru ani klienta. Naopak nevýhodou je větší datový tok mezi klientem a serverem. ViewState může docela nabobtnat pokud jej používáme nesprávně. Například explicitní ukládání datových objektů.
- **Session** uchovává data na straně serveru. Posílá se pak pouze identifikátor dané Session jako cookie (soubor uložený v rámci privátních dat prohlížeče) nebo v URL. ASP.NET umožňuje ukládat session do samostatného procesu nebo taká na SQL server (pro tento případ musí být v cílové databázi připravený specifické tabulky a uložené procedury).

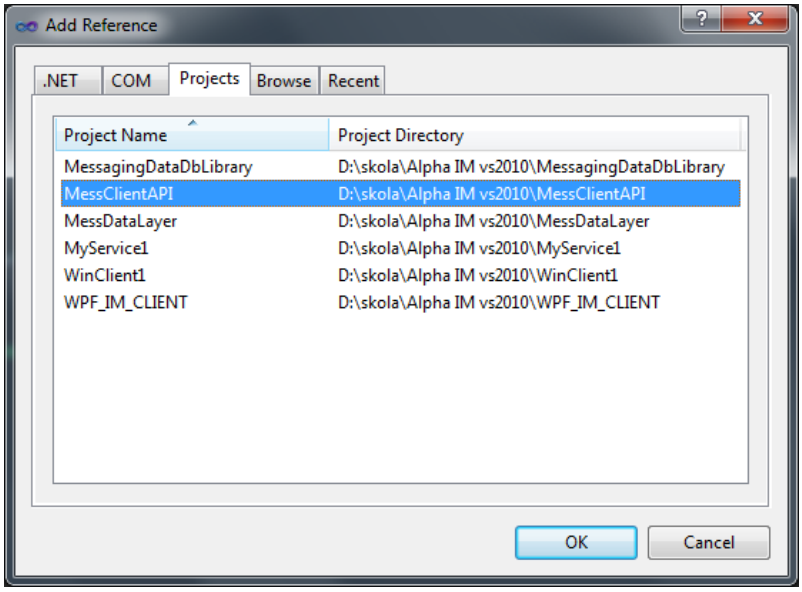
V aktuální verzi má navíc ASP.NET podporu AJAX[1], což zase o kus sbližuje desktopové a webové programování.

[1] **AJAX** (Asynchronous JavaScript and XML) je obecné označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich znovunačítání. Na rozdíl od klasických webových aplikací poskytují uživatelsky příjemnější prostředí, ale vyžadují použití moderních webových prohlížečů.

1.7.2 Nastavení projektu

V *solution* vytvoříme nový projekt „ASP.NET Web Application“. Projekt obsahuje jednu prázdnou stránku a webový konfigurační soubor.

Aby bylo možné používat klientskou vrstvu, která byla zmíněna v předchozí kapitole, tak musíme do projektu webové aplikace přiřadit referenci na projekt s klientskou vrstvou (obrázek 1.8), a také přidat do projektu referenci na `System.Runtime.Serialization`.



Obr. 1.8: Přidání reference na klientskou vrstvu

Toto ale není jediné nastavení, které musíme provést, aby komunikace fungovala. Je třeba do konfiguračního souboru `web.config` vložit speciální konfigurační sekci:

```
<client>
  <endpoint address="<adresa wcf služby>"
      binding="basicHttpBinding"
      bindingConfiguration="BasicHttpBinding_IMessagingService"
      contract="TestMessApi.IMessagingService"
      name="BasicHttpBinding_IMessagingService">
      <identity>
          <dns value="localhost" />
      </identity>
  </endpoint>
</client>
```

Kód 1.4: Potřebné nastavení v souboru web.config

1.7.3 Postup

Demonstrační webová aplikace bude poměrně jednoduchá. Vystačíme si jednou ASP.NET stránkou a několika málo metodami obsluhujícími události stránky.

Využijeme tedy následující ASP.NET komponenty:

- **MultiView** je kontrolka, která v jeden moment zobrazuje pouze jeden z kolekce obsahových kontainerů zvaných **View**. Využijeme jej pro přepínání mezi jednotlivými módy aplikace (přihlášení, registrace, messenger jako takový).
- **UpdatePanel** tímto prvkem definujeme oblast stránky, která má být zpracovávána asynchronně pomocí AJAX. Členské kontrolky tak nebudou muset pro svou činnost používat obnovení celé stránky.
- **Timer** pomocí javascriptu odpočítává nastavený interval, a po jeho uběhnutí spustí automaticky svou událost „Thick“. Tuto kontrolku využijeme v kombinaci s UpdatePanelem k automatickému obnovování messengeru, abychom měli v pravidelných intervalech nově přichozí zprávy k dispozici.

Ostatní použité kontrolky jsou už tak triviální, že jsem si dovolil je nekomentovat.

Do stránky vložíme MultiView a v něm vytvoříme tři nová View. Názvy (ID) jednotlivých View zvolíme tak, abychom věděli k čemu dané View slouží a jakou část programu bude obsahovat. View budou následující:

- **Přihlášení** bude obsahovat dvě pole pro vložení ID účtu a hesla. A nesmí chybět tlačítko pro odeslání požadavku na přihlášení.
- **Registrace** bude obsahovat pole pro NickName a dvě pole pro zadání hesla a jeho ověření. A tlačítko pro odeslání.
- **Okno konverzace a seznam uživatelů** zde bude prvek Literal, který do požadovaného místa na stránce vypisuje HTML obsah. Do něho budeme vkládat zprávy. Pak bude v této části DropDownList, ve kterém budou všechny kontakty systému. Bude sloužit pro výběr kontaktu, kterému chceme odeslat zprávu. Samozřejmě nesmí chybět pole pro vložení textu a odesílací tlačítko.

Ještě je vhodné do programu umístit nějaký prvek Label, do kterého bude možné vypisovat stav prováděné akce.

Pokud máme výše zmíněné kontrolky připravené, můžeme přiřadit tlačítkům metody k událostem Click a Timeru k události Thick. Funkčnost tlačítek je vcelku jasná. Stačí jen využít metody klientské vrstvy jako na příkladu 1.5

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    if (MessApi.User_Login(int.Parse(tbId.Text), tbPasswordLogin.Text))
    {
        mvMessanger.SetActiveView(viewMessenger);
        UserID = int.Parse(tbId.Text);
        UserPassword = tbPasswordLogin.Text;

        foreach (var item in MessApi.Contact_GetAllContacts())
        {
            ddlUsers.Items.Add(new ListItem(item.USR_NICK, item.ID_USR.ToString()))
        }
        tmrCheckMessages.Enabled = true;
        lbNickName.Text = MessApi.Contact_GetContactById(int.Parse(tbId.Text)).USR_
    }
    else
        lbInfo.Text = "Prihlášení se nepodarilo";
}
```

Kód 1.5: Použití klientské vrstvy pro přihlášení

Asi nejsložitější akcí v tomto případě je výpis zpráv. Ten se provede tak, že události Thick Timeru přiřadíme funkčnost, která bude v určitých intervalech asynchronně kontrolovat nové zprávy na serveru. Metoda nám vrátí kolekci zpráv, kde v cyklu každou naformátujeme pomocí objektu `StringBuilder` do HTML textu a přiřadíme na počátek v dříve zmíněném prvku Literal.

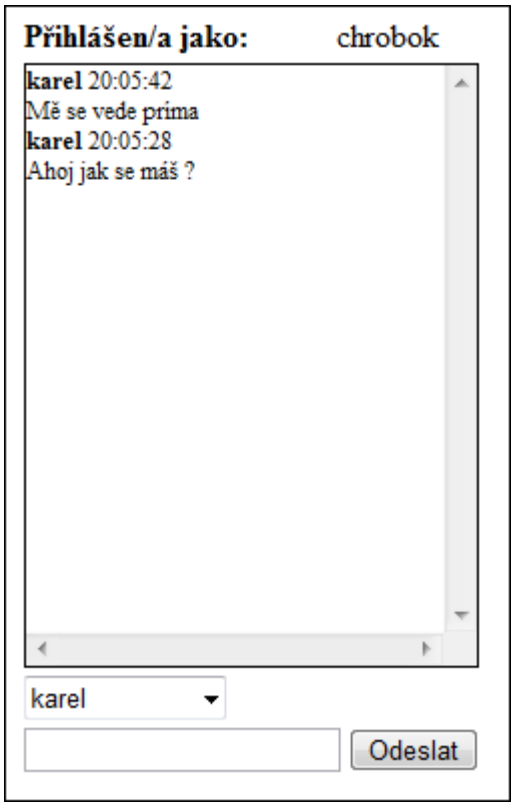
Výsledná aplikace by mohla vypadat asi jako na obrázcích 1.9,1.10 a 1.11.

Obr. 1.9: Příhlašovací část webové aplikace

1.7.4 Závěr

Na této ukázce jsme se letmo seznámili s možnostmi ASP.NET. Tato demonstrace moderního vývoje webu, který v dnešní době stojí na kvalitních frameworcích[2] nám

[2]**Framework** je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovnu API, návrhové vzory nebo doporučené postupy při vývoji.



Obr. 1.10: Komunikační webové aplikace

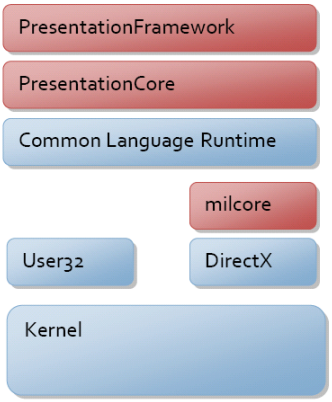
Obr. 1.11: Registrační část webové aplikace

pomocí několika málo kontrolek a bez znalosti ukázala jak snadné je vytvoření zdánlivě složité aplikace. JavaScriptu[3]

V další části se seznámíme s vývojem aplikací pro Windows. Ukážeme si možnosti programování aplikací ve WPF jenž má nahradit dnes už ne zcela vyhovující Windows Forms.

1.8 Desktopová aplikace ve WPF

Windows Presentation Foundation (WPF) je grafický subsystém sloužící k renderování grafických rozhraní ve Windows aplikacích. Pod názvem „Avalon“ byl představen jako součást .NET Frameworku 3.0. navržen, aby odstranil závislost na GDI[4] subsystému, je WPF postavena na technologii DirectX, která umožňuje hardwarovou akceleraci a zpřístupňuje moderní vymoženosti grafických rozhraní, jakými jsou průhlednost, přechody a různé další transformace. WPF dobře odděluje vývoj grafického rozhraní a samotnou logiku aplikace.



Obr. 1.12: WPF aplikace - Struktura WPF Framework

WPF nabízí nový jazyk z rodiny XML, známý jako XAML, jimž můžeme definovat jednotlivé elementy uživatelského rozhraní a vazby mezi nimi. WPF aplikaci můžeme spustit pod systémem Windows nebo také ve webovém prohlížeči (ten musí

[3]**JavaScript** je multiplatformní, objektově orientovaný skriptovací jazyk, jehož autorem je Brendan Eich z tehdejší společnosti Netscape. Nyní se zpravidla používá jako interpretovaný programovací jazyk pro WWW stránky, často vkládaný přímo do HTML kódu stránky. Jsou jím obvykle ovládány různé interaktivní prvky GUI (tlačítka, textová políčka) nebo tvořeny animace a efekty obrázků.

[4]**Graphics Device Interface** (zkráceně GDI či někdy nazýváno jako Graphical Device Interface) je spolu s kernelem a uživatelským rozhraním jednou ze tří hlavních součástí operačního systému Microsoft Windows. Graphics Device Interface slouží k reprezentaci grafických objektů a jejich transformací do výstupních zařízení jako jsou monitory či tiskárny.

mít nainstalovaný doplněk Silverlight). Sjednocuje hned několik služeb: uživatelské rozhraní, 2D a 3D kreslení, vektorovou grafiku, animace, audi, video atd.

Microsoft Silverlight je webová podmnožina WPF umožňující psát aplikace graficky stejně bohaté jako je například Flash pomocí stejného programovacího modelu jako .NET aplikace. Silverlight nemá plnou podporu všech systémových knihoven. 3D není v Silverlight podporováno.

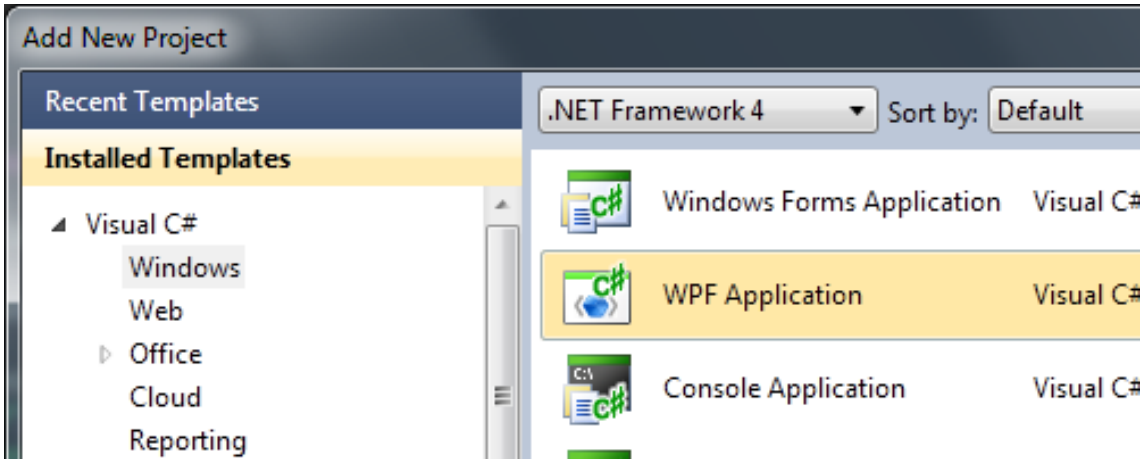
1.8.1 První WPF aplikace

Nejprve bude dobré si ukázat jak vlastně WPF aplikaci vytvořit a také se seznámit se strukturou projektu i základními soubory v projektu.

Zabudovaná podpora WPF je ve Visual Studiu od verze 2008.

Přidáme si tedy do solution nový projekt. Nazveme jej tak aby bylo jasné k čemu slouží. Například „WpfImClient“.

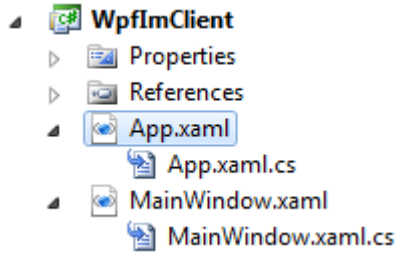
V okně pro přidání projektu vybereme projekt viz obrázek 1.13.



Obr. 1.13: WPF aplikace - Přidání projektu s WPF aplikací

Projekt bude obsahovat dva XAML soubory. Základním souborem každé WPF aplikace je `app.xaml`. Když ho otevřeme můžeme si všimnout, pro nás zajímavého parametru `StartupUri="MainWindow.xaml"`. Tento parametr nastavuje výchozí okno aplikace, které se má zobrazit po spuštění programu. Soubor `app.xaml` má také svůj CodeBehind[5], ve kterém můžeme dle potřeby implementovat vlastní funkce nebo vlastnosti. Toto se však běžně v tomto souboru nedělá.

[5]**CodeBehind** se označuje kód v pozadí implementující funkčni logiku daného okna, webové stránky nebo jiných objektů používaných v .NET Framework.

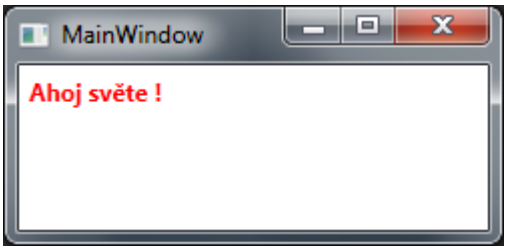


Obr. 1.14: WPF aplikace - Nový WPF projekt

Soustředíme se na soubor `MainWindow.xaml`. Po jeho otevření se nám ve výchozím zobrazení ukáže v horní části WISYWYG[6] editor a ve spodní editor xaml kódu.

Výchozím elementem ve WPF okně je `Grid`(tabulka). Je to jeden z Layout Controls, o kterých si povíme později. Tento `Grid` smažeme a vložíme mezi tagy `Window` element `Label`. Tento element slouží k usmístění nějakého textu.

```
<Label FontWeight="Bold" Foreground="Red">Ahoj svete !</Label>
```



Obr. 1.15: WPF aplikace - První WPF okno Ahoj Světe!

1.8.2 Layout Controls

Většina kontrolek ve WPF umožňuje, aby do nich šlo vkládat další kontrolky. Můžeme například do `Button` vložit obrázek a `TextBox`. Pokud, ale vložíme do `Button` dvě nebo více kontrolek bez toho aniž by se použil nějaký LayoutControl, tak vývojové prostředí zahlásí chybu. Proč ? No protože v tento moment WPF neví jak kontrolky uvnitř `Button` uspořádat. Proto existují právě LayoutControls. Tato sada kontrolek má jasně definované pravidla jak kontrolky rozmístit. Proto je možné do nich vkládat teoreticky neomezené množství dalších kontrolek aniž by nastala nějaká chyba.

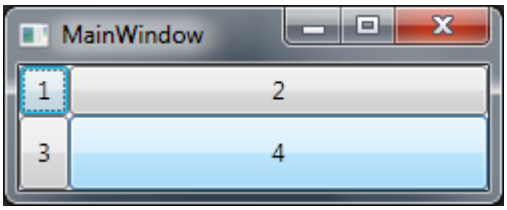
Existují následující LayoutControls:

[6] **WYSIWYG** je akronym anglické věty „What you see is what you get“, česky „co vidíš, to dostaneš“. Tato zkratka označuje způsob editace dokumentů v počítači, při kterém je verze zobrazená na obrazovce vzhledově totožná s výslednou verzí dokumentu.

- **Grid** je klasická tabulka. Chová se podobně jako tabulky například v HTML. Mezi tagy `Grid` je třeba pouze definovat počet sloupců a řádku případně jim nastavit jakou mají mít velikost. Umístění vložených kontrolek se pak provádí nastavením přímo v dané kontrolce. Takový jednoduchý `Grid` by mohl vypadat asi takto:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="25" />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="25" />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Button Grid.Row="0" Grid.Column="0">1</Button>
    <Button Grid.Row="0" Grid.Column="1">2</Button>
    <Button Grid.Row="1" Grid.Column="0">3</Button>
    <Button Grid.Row="1" Grid.Column="1">4</Button>
</Grid>
```

Kód 1.6: Jednoduchý Grid

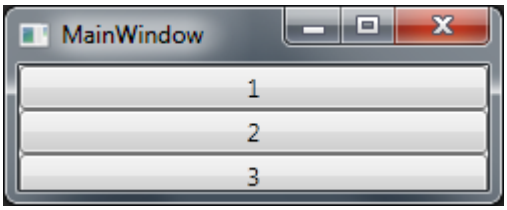


Obr. 1.16: WPF aplikace - Gridu

- **StackPanel** uspořádává kontrolky za sebou. Lze nastavit vertikální nebo horizontální orientaci

```
<StackPanel Orientation="Vertical">
    <Button>1</Button>
    <Button>2</Button>
    <Button>3</Button>
</StackPanel>
```

Kód 1.7: StackPanel

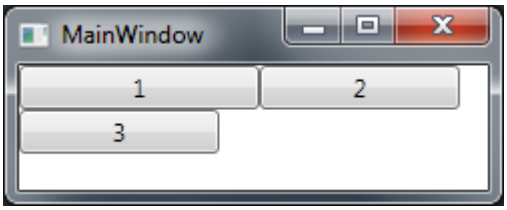


Obr. 1.17: WPF aplikace - StackPanel

- **WrapPanel** funguje stejně jako textový dokument. Nezalamuje však jen text ale i všechny kontrolky, které ukládá horizontálně vedle sebe a pokud se nějaká nevejde do dané šířky tak jí umístí na „nový řádek“.

```
<WrapPanel>
    <Button Width="120">1</Button>
    <Button Width="100">2</Button>
    <Button Width="100">3</Button>
</WrapPanel>
```

Kód 1.8: WrapPanel

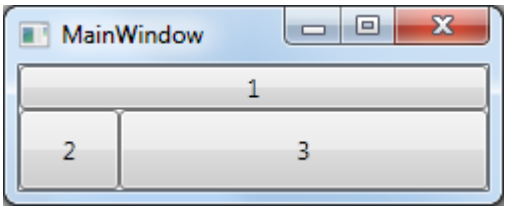


Obr. 1.18: WPF aplikace - WrapPanel

- **DockPanel** funguje tak, že přilepí kontrolky k jednomu ze 4 okrajů a ostatní element vyplní zbylé místo. Kontrolkám uvnitř musíme stejně jako třeba u kontrolky `Grid` nastavit, kde se mají nacházet v tomto případě, kde se mají „přilepit“.

```
<WrapPanel>
    <Button Width="120">1</Button>
    <Button Width="100">2</Button>
    <Button Width="100">3</Button>
</WrapPanel>
```

Kód 1.9: WrapPanel

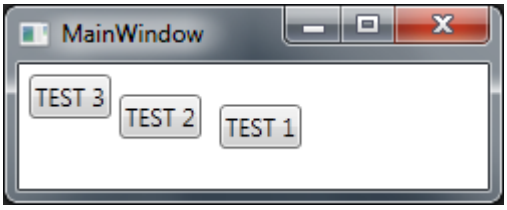


Obr. 1.19: WPF aplikace - DockPanel

- **Canvas** slouží k absolutnímu určení pozice. Tento LayoutControl není příliš vhodný pro tvorbu formulářů. Využití však najde pokud v některých graficky náročnějších kombinacích.

```
<Canvas>
    <Button Canvas.Top="20" Canvas.Left="100">TEST 1</Button>
    <Button Canvas.Top="15" Canvas.Left="50">TEST 2</Button>
    <Button Canvas.Top="5" Canvas.Left="5">TEST 3</Button>
</Canvas>
```

Kód 1.10: Canvas



Obr. 1.20: WPF aplikace - Canvas

Se znalostí těchto prvků WPF se můžeme pustit do designování formulářů. Další ovládací prvky jsou podobné jako v ASP.NET nebo WindowsForms.

1.8.3 Styly

Ve WPF máme široké možnosti jak udělat z obyčejných kontrolek kontrolky neobyčejné. Možnosti úpravy vzhledu jsou opravdu široké. Lze nastavovat různé parametry jako barva pozadí, textu atd. Lze však kontrolkám vtisknout i zcela vlastní šablonu jak by měly vypadat. Funkce kontrolky zůstává stejná, ale její vzhled lze upravovat zcela k nepoznání.

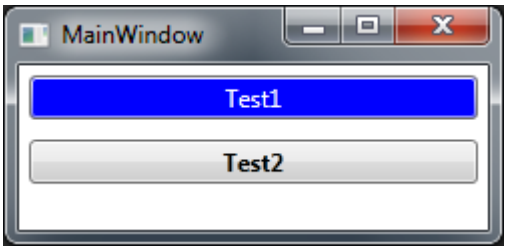
Většinou je však hlavním požadavkem jednotný vzhled aplikace. Bylo by velmi náročné psát například pro každé jedno tlačítko jeho vzhled. Navíc pokud bychom se rozhodli změnit vzhled aplikace museli bychom pracně přepisovat každé tlačítko. Proto ve WPF existují styly, což jsou vlastně množiny určitých vlastností. Můžou

obsahovat jednoduché vlastnosti jako velikost a barvu písma atd. Ale je možné nimi nastavovat celé šablony ovládacích prvků, ale to si ukážeme později.

Základní použití stylu vypadá asi takto jako v kódu 1.11 a výsledek pak na obrázku 1.21.

```
<StackPanel Orientation="Vertical">
    <StackPanel.Resources>
        <Style TargetType="Button">
            <Setter Property="Button.Background" Value="Blue" />
            <Setter Property="Button.Foreground" Value="White" />
            <Setter Property="Button.Margin" Value="5" />
        </Style>
        <Style x:Key="BlueButton">
            <Setter Property="Button.FontWeight" Value="Bold" />
            <Setter Property="Button.Margin" Value="5" />
        </Style>
    </StackPanel.Resources>
    <Button>Test1</Button>
    <Button Style="{StaticResource BlueButton}">Test2</Button>
</StackPanel>
```

Kód 1.11: WPF - Ukázka stylů



Obr. 1.21: WPF - Ukázka jednoduchých stylů

Styly lze používat podobným způsobem jako například CSS[7] při psaní HTML formulářů a webových stránek. V příkladu jsme si ukázali styl pojmenovaný a obecný. Rozdíl je v atributu `TargetType`. Ten určuje na jaký typ kontrolky má být styl aplikován obecně. Takže pokud tento nepovinný parametr vyplníme bude se styl aplikovat automaticky na všechny kontrolky v elementu, kde byl styl definován. U Pojmenovaných stylů musíme styl v kontrolce explicitně přiřadit, aby se aplikoval.

Stejně jako u CSS lze styly ve WPF dědit. Nefunguje to však automaticky, ale musí se nastavit parametr `BasedOn`, který určuje rodičovský styl zděděného stylu.

[7] **Cascading Style Sheets** (CSS) je jazyk pro popis způsobu zobrazení stránek napsaných v jazycích HTML, XHTML nebo XML

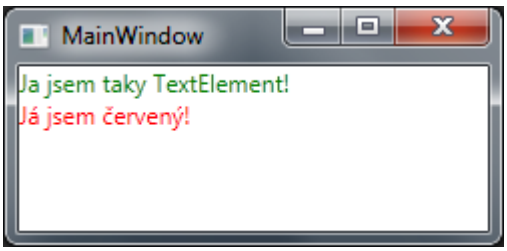
Dependency Properties

Než přejdeme k dalším vlastnostem WPF bylo by dobré říct si něco o tzv. „Dependency Properties“. Jedná se o speciální vlastnosti kontrolek. Jejich zvláštnost spočívá ve schopnosti přebírat hodnoty od nadřazených prvků v nichž se kontrolka nachází. Kontrolka hledá postupně ve stromu elementů až narazí na požadovanou Dependency vlastnost, kterou pak použije. Zápis může pak vypadat následujícím způsobem:

```
<StackPanel Orientation="Vertical" TextElement.Foreground="Green">
    <TextBlock>Ja jsem taky TextElement!</TextBlock>
    <StackPanel TextElement.Foreground="Red">
        <TextBlock>Já jsem cervený!</TextBlock>
    </StackPanel>
</StackPanel>
```

Kód 1.12: WPF - Dependency property

Na obrázku 1.22 vidíme, že kontrolky použily tu Dependency vlastnost, která je jim ve stromu elementů nejblíž.



Obr. 1.22: WPF - Použití Dependency properties

1.8.4 Trigry

Trigr je označení pro podmínku spouštějící nějakou událost v událostmi řízeném programování.

V jazyce XAML nám umožňují trigry reagovat na určitou změnu vlastností daného prvku, datovou proměnnou nebo nějakou jinou událost. Rozdělujeme je tedy na:

- **Property triggers** Tento typ trigrů reaguje na změnu proměnných. Pokud se nějaká proměnná změní tak aby odpovídala podmínce můžeme vyvolat změnu vlastností pomocí elementu `Setter`. Jak může jednoduchý triger vypadat vidíme zde:

```
<StackPanel.Resources>
  <Style TargetType="Button">
      <Style.Triggers>
          <Trigger Property="Button.IsMouseOver" Value="true">
              <Setter Property="Button.Foreground" Value="Red" />
              <Setter Property="Button.FontWeight" Value="Bold" />
          </Trigger>
      </Style.Triggers>
      <Setter Property="Button.Margin" Value="5" />
  </Style>
</StackPanel.Resources>
```

Kód 1.13: WPF - Property trigr

- **Data triggers** Data trigry umí reagovat na změny v uživatelských proměnných.
- **Event triggers** Event trigry jsou omezené pouze na používaní s animacemi. Reagují na nějakou událost kontrolky. U tlačítka může být touto událostí například klik.

Hlavní využití nacházejí triggery v oživení uživatelského rozhraní. Na základě trigrů můžeme nastavovat různé vlastnosti kontrolek a také upravovat jejich styly.

1.8.5 Šablony

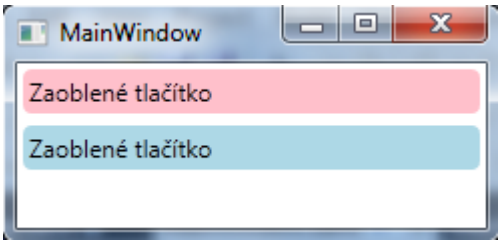
Prozatím jsme si ukázali ve stylech pouze jednoduchá nastavení vzhledu. Pomocí stylů však můžeme aplikovat na danou kontrolku šablony, které nám umožňují zcela zásadně proměnit vhled kontrolky. Použití vlastního vzhledu však neztrácíme původní funkčnost kontrolky.

Možnosti jak upravit kontrolku jsou takřka neomezené. Pro ukázku nám, ale stačí následující úprava tlačítka.

```
<Style TargetType="{x:Type Button}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate>
                <Border
                        BorderBrush="Red"
                        Padding="3"
                        Margin="3"
                        CornerRadius="4"
                        Background="{TemplateBinding Button.Background}">
                    <ContentPresenter
                        Content="{TemplateBinding Button.Content}" />
                </Border>
            </ControlTemplate>
```

```
        </Setter.Value>
    </Setter>
    <Setter Property="Button.Background" Value="LightBlue" />
    <Style.Triggers>
        <Trigger Property="IsMouseOver" Value="true">
            <Setter Property="Button.Background" Value="Pink" />
        </Trigger>
    </Style.Triggers>
</Style>
```

Kód 1.14: WPF - Ukázka šablony



Obr. 1.23: WPF - Použití šablony

1.8.6 Data Binding

V podstatě každá aplikace zpracovává a vypisuje nějaká data, proto je nutné si říci taky něco o vypisování dat ve WPF.

Asi nejpohodlnějším způsobem jak data ve WPF vypsat, je použít nějaký opakovací prvek jako například ListBox, ListView atd. Samozřejmě můžeme data i manuálně v kódu. Zajímavější však bude, ukázat si výpis například pomocí kontrolky ListBox. Tyto opakovací kontrolky mají již připravenou vlastnost `ItemSource`, které pouze přiřadíme kolekci datových objektů a kontrolka je schopna data ihned vypsat. Je nutné pouze definovat vzhled položky. To můžeme provést například pomocí šablony aplikované stylem jak se můžeme podívat na následujícím kódu.

```
<Style TargetType="{x:Type ListBoxItem}">
        <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type ListBoxItem}">
        <Grid Margin="3">
          <Grid.ColumnDefinitions>
              <ColumnDefinition Width="30" />
              <ColumnDefinition />
          </Grid.ColumnDefinitions>
          <Border Grid.Column="0"
```

```
                Width="20"
                Height="20"
                Background="{Binding Barva, Converter={StaticResource ColorConverter}}">
            </Border>
            <TextBlock Grid.Column="1" Foreground="Black"
                Text="{Binding Name}"></TextBlock>
          </Grid>
        </ControlTemplate>
      </Setter.Value>
        </Setter>
</Style>
```

Kód 1.15: WPF - Data Binding

Definice proměnné, kterou chceme „vytáhnout“ z datového objektu, se provádí pomocí složených závorek a klíčového slova `Binding`.

Takto jednoduchý výpis však na složitější aplikace nestačí. Někdy je potřeba základní datovou hodnotu převést na jinou hodnotu vhodnou pro zobrazení. Proto ve WPF existují tzv. `ValueConverter`.

1.8.7 Konvertory

Jak již bylo zmíněno, tak k převodu jednoduchých hodnot na „zobrazitelné“ hodnoty použijeme konvertor. Jedná se o třídu implementující rozhraní `IValueConverter`. V podstatě se jedná o třídu s metodou přijímající vstupní parametr, daný většinou právě konstrukcí s `Binding`, kterou převede na požadovaný výstupní objekt. Použití konvertoru můžeme vidět u předchozího úryvku kódu v nastavení barvy pozadí elementu `Border`.

Aby však bylo možné konvertor použít je nutné ho definovat v XAML kódu. `<local:ColorConverter x:Key="ColorConverter" />` a také přidat referenci jmenného prostoru, kde se konvertor nachází, do parametrů `Window` elementu.

Základní implementace třídy ColorConverter může vypadat asi takto:

```
public class ColorConverter : IValueConverter
{
  #region IValueConverter Members

  public object Convert(object value, Type targetType, object parameter,
        System.Globalization.CultureInfo culture)
  {
    string barva = (string)value;
    switch (barva)
    {
      case "Modrá": return Brushes.Blue;
      case "Zelená": return Brushes.Green;
```

```
        case "Cervená": return Brushes.Red;
        default: return Brushes.White;
      }
    }

    public object ConvertBack(object value, Type targetType, object parameter,
          System.Globalization.CultureInfo culture)
    {
      throw new NotImplementedException();
    }

    #endregion
}
```

Kód 1.16: WPF - Konvertor hodnot

Aby bylo vše jasné, je třeba si ukázat datový objekt, data a také výsledek dosažený použitím konvertoru.

```
public class MojeDatovaTrida
{
    public string Name { get; set; }
    public string Barva { get; set; }
}
```

Kód 1.17: WPF - Datový objekt

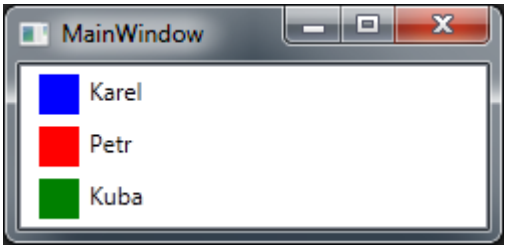
```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    List<MojeDatovaTrida> list = new List<MojeDatovaTrida>();
    list.Add(new MojeDatovaTrida() { Name = "Karel", Barva = "Modrá" });
    list.Add(new MojeDatovaTrida() { Name = "Petr", Barva = "Cervená" });
    list.Add(new MojeDatovaTrida() { Name = "Kuba", Barva = "Zelená" });

    MyListBox.ItemsSource = list;
}
```

Kód 1.18: WPF - Naplnění kolekce

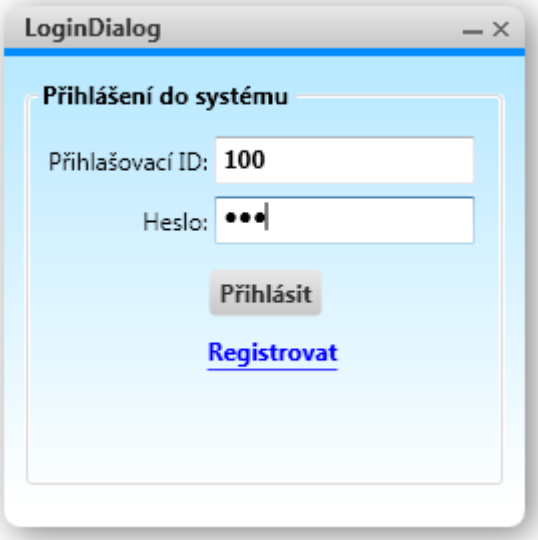
1.8.8 Koncová klientská aplikace ve WPF

V předchozích částech práce věnovaných základům WPF jsme se seznámili s dostatečnými informacemi, abychom mohli vytvořit opravdu profesionálně vypadající aplikaci.

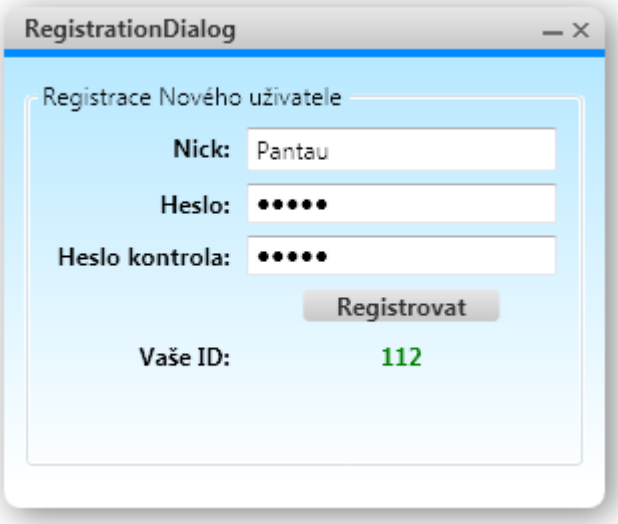
Jako demonstraci použiji klientskou aplikaci využívající naší klientskou vrstvu IM systému, kterou jsem vytvořil během studia technologie WPF.



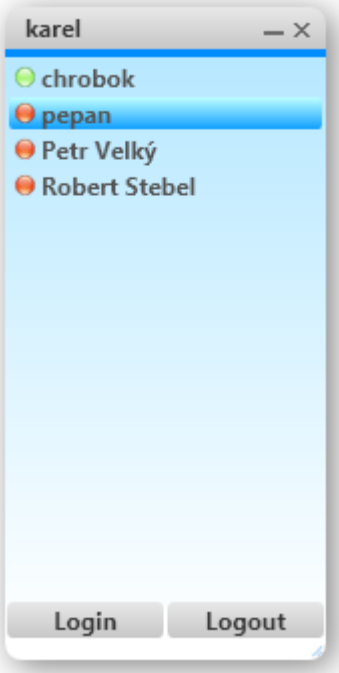
Obr. 1.24: WPF - Použití konvertoru



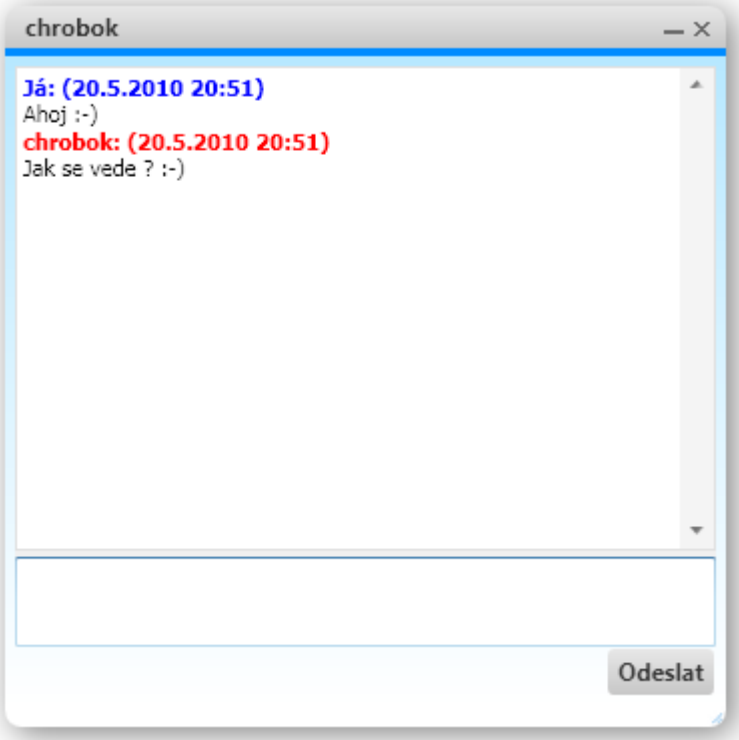
Obr. 1.25: WPF ukázka - Přihlašovací dialog



Obr. 1.26: WPF ukázka - Registrační dialog dialog



Obr. 1.27: WPF ukázka - Seznam kontaktů



Obr. 1.28: WPF ukázka - Okno diskuze

2 ZÁVĚR

Tato práce má za cíl ukázat možnosti tvorby internetových aplikací postavených na .NET Framework. Tato platforma poskytuje prostředí a nástroje umožňující efektivní tvorbu síťových aplikací s dostatečnou mírou abstrakce. Programátor má možnost soustředit se primárně na řešení problému a jeho pozornost tak není zbytečně odváděna k řešení síťového provozu na nižších vrstvách. Platforma .NET se snaží v mnoha ohledech sjednocovat webové, windows a serverové aplikace ať už pomocí WCF nebo i WPF. Popud ke vzniku této široké a velice univerzální platformy je zapříčiněn zejména masovým rozšířením internetu. Díky rostoucímu výkonu počítačů a použití vykreslování pomocí grafických akcelerátorů, je možné tvořit uživatelská rozhraní pomocí technologie WPF, která byla ještě v době svého uvedení (v roce 2005) příliš náročná na valnou část tehdejšího hardwaru. Rostoucí výkon hardwaru jde ruku v ruce se zvyšování rychlostí sítí a internetu. Dnešní rychlosti sítí dovolují bezproblémové použití distribuovaných aplikací a technologie WCF jejich tvorbu velmi zjednodušuje.

LITERATURA

[1] MacDonald, M. *Pro WPF in C# 2008.* Apress 2008, ISBN-13 (electronic): 978-1-59059-955-6

[2] *Základy Windows Presentation Foundation* [online].
2007, poslední aktualizace 13.3.2007 [cit. 2010-3-10]
Dostupné z URL: <http://www.vyvojar.cz/Seri es/3-zaciname-s-wpf.aspx>.

[3] *Úvod do WCF* [online].
2007, poslední aktualizace 7.5.2007 [cit. 2010-3-10]
Dostupné z URL: <http://www.vyvojar.cz/Series/2-uvod-do-wcf.aspx>.

[4] *Creating a Custom Window in WPF* [online].
2008, poslední aktualizace 17.8.2008 [cit. 2010-3-10]
Dostupné z URL: <http://www.kirupa.com/blend_wpf/custom_wpf_windows.htm>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

WPF Windows Presentation Fundation

WCF Windows Communication Fundation

CSS Kaskádové styly - Cascading Style Sheets

HTML Hypertextový značkovací jazyk - Hypertext Markup Language

IMS Systém rychlého odesílání zpráv - Instant Messaging System

SEZNAM PŘÍLOH

A ZDROJOVÉ KÓDY

Zdrojové kódy najdete na přiloženém CD. Ke spuštění programu ze zdrojových kódů je nutné mít Microsoft Visual Studio 2010 a Microsoft SQL Express 2005 nebo vyšší.