Zabezpečení Windows Communication Foundation

Windows Communication Foundation security

Bc. Miloš Janíček

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2009/2010

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Miloš JANÍČEK**
Osobní číslo:	**A08764**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Zabezpečení WCF služeb v prostředí .NET Framework v.3.5 a jejich využití v distribuovaných aplikacích.**

Zásady pro vypracování:

1. **Proveďte literární rešerši v oblasti zabezpečení WCF služeb v prostředí .NET Framework 3.5.**
2. **Analyzujte distribuované aplikace a jejich možnosti.**
3. **Navrhněte jednoduchou distribouvanou aplikaci a implementujte základní moduly serverové části aplikace a komunikační WCF služby jako Windows Service v prostředí .Net Framework (respektujte bezpečnost aplikace).**
4. **Realizujte instalace a konfigurace klientské i serverové aplikace na lokálním počítači.**
5. **Vyslovte závěry k realizovanému projektu.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **SHARP, John. Microsoft Windows Communication Foundation Step by Step. Praha : Microsoft Press, 2007. 448 s. ISBN 9780735623361.**
2. **KLEIN, Scott. Professional WCF programming: .NET development with the Windows Communication Foundation. Indianapolis : John Wiley & Sons, 2007. 430 s. ISBN 978-0-470-08984-2.**
3. **PEIRIS, Chris, MULDER, Dennis. Pro WCF: practical Microsoft SOA implementation. Berkeley : Apress, 2007. 475 s. ISBN 978-1-59059-702-6.**
4. **DOSTÁLEK, Libor, et al. Velký průvodce protokoly TCP/IP: Bezpečnost . 2. aktualiz. vyd. Brno : Computer Press, 2003. 592 s. ISBN 80-7226-849-X.**
5. **SHARP, John. Microsoft Visual C 2008 krok za krokem. Brno : Computer Press, 2008. 592 s. ISBN 978-80-251-2027-9.**
6. **SHARP, John, JAGGER, Jon. Microsoft Visual C .NET krok za krokem. Brno : Mobil Media a.s., 2002. 655 s. ISBN 80-86593-27-4.**
7. **PIPER, Fred, MURPHY, Sean. Kryptografie. Praha : Dokořán, 2006. 157 s. ISBN 80-7363-074-5.**
8. **KOPKA, Martin. Počítačové sítě. Olomouc : Univerzita Palackého v Olomouci, 1996. 98 s.**
9. **Certificate Creation Tool (Makecert.exe) . MSDN .NET Framework Developer Center [online]. 2009 [cit. 2009-11-16]. Dostupný z WWW: .**

Vedoucí diplomové práce: **doc. Mgr. Roman Jašek, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **19. února 2010**

Termín odevzdání diplomové práce: **8. června 2010**

Ve Zlíně dne 19. února 2010

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

prof. Ing. Vladimír Vašek, CSc.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Tato práce se zabývá zabezpečením komunikace mezi klientskou aplikací a serverem distribuovaného systému v prostředí .NET Framework. Klientská aplikace komunikuje se serverem prostřednictvím služby Windows Communication Foundation (WCF). Jelikož jsou klient a server od sebe mnohdy vzdálení a jejich komunikace či přenášení dat probíhá prostřednictvím internetu, je nutné tento způsob přenosu informací zabezpečit. V textu je popsán princip fungování distribuovaných aplikací, seznámení s technologií WCF, způsob zabezpečení WCF služby při hostování ve Windows službě a v Internetové Informační Službě (IIS). Dále je popsána konfigurace koncových bodů tzv. endpointů, které jsou vystaveny pro příjem a odesílání zpráv službou. Také je zde uvedena autorizace klienta a způsoby zabezpečení citlivých informací.

Klíčová slova: WCF, endpoint, IIS, SSL, HTTP, HTTPS, certifikát, Certifikační autorita

ABSTRACT

This work deals with the security of communication between the client application and server in the distributed system environment .NET Framework. Client application communicates with the server through Windows Communication Foundation (WCF). Since the client and server are often apart and their communication and data transfer takes place via the Internet, you need make safe this type of transmission of informations. The text describes the operating principle of distributed applications, identification with the WCF technology, the WCF service method of securing the hosting service in Windows and Internet Information Services (IIS). Next there is described the configuration of endpoints called endpoints that is exposed for receiving and sending messages by this service. There is also a set authorization client and methods for securing sensitive information.

Keywords: WCF, endpoint, IIS, SSL, HTTP, HTTPS, certificate, Certificate Authority

Chtěl bych poděkovat vedoucímu diplomové práce panu doc. Mgr. Romanu Jaškovi, Ph.D. za odborné vedení a konzultace při vypracování této diplomové práce a za poskytnutí odborných informací.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD

V dnešní době se využívají stále složitější informační systémy, něž tomu bylo dříve. Jsou kladeny stále větší nároky na výkonnost a dnešní systémy musí zpracovávat stále větší objemy dat. Také více zákazníků přichází s požadavky propojit svoji geograficky rozsáhlou strukturu firmy jedním systémem prostřednictvím internetu. Dnes je spojení pomocí internetu naprosto běžná věc a proto je logické, že jedním z nároků na software, který spousta společností v praxi využívá pro chod firmy, je podpora propojení jednotlivých poboček a pracovišť. Díky stále se rozšiřujícímu internetu a zvyšující se rychlosti připojení mohou spolupracovat subjekty, které jsou od sebe velice vzdáleny. Taková sdílená komunikace přináší i určitá rizika v oblasti bezpečnosti. Často si jednotlivé komunikující strany vyměňují citlivá obchodní nebo osobní data, která by měla zůstat utajená případnému útočníkovi. Proto je nutností provozovat takovou komunikaci bezpečně. O způsobu a technologii zabezpečení komunikace v distribuovaných aplikacích pojednávám v této diplomové práci a snažím se čtenáři vysvětlit i prakticky ukázat zabezpečení takovéto komunikace.

I. TEORETICKÁ ČÁST

1 DISTRIBUOVANÉ APLIKACE

Než budeme hovořit o principech distribuovaných aplikací je potřeba si objasnit pojem distribuovaný systém, na který se zaměříme v úvodu této kapitoly.

1.1 Distribuované systémy

Neustálé zvyšování hardwarových nároků a zpracovávání stále většího objemu informací si žádá neustálé zvyšování výpočetní kapacity procesorů ve výpočetních prostředcích. Ne každý si zejména z finančního hlediska může dovolit nějaký superpočítač, na kterém by realizoval složité výpočty, ale existuje i jiné řešení. Pokud nám řešená úloha umožňuje vhodně ji rozdělit mezi větší množství pomalejších procesorů, můžeme dosáhnout poměrně velké výpočetní kapacity. To nám umožňují právě distribuované systémy.

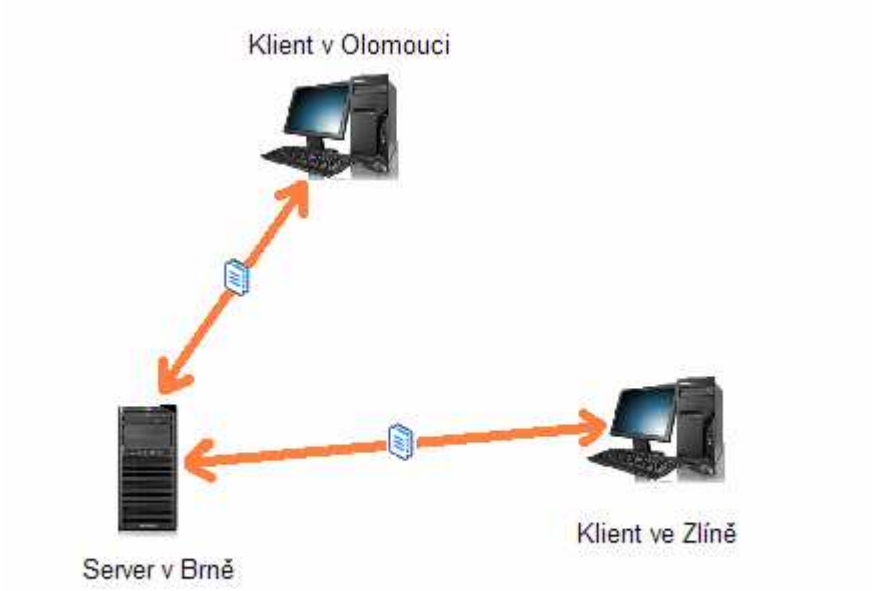
Aby jakýkoliv systém mohl být distribuovaný, musí splňovat několik základních charakteristik. Každý distribuovaný systém se musí skládat z více navzájem propojených autonomních uzlů (počítačů), které spolu komunikují a navenek se jeví jako jeden integrovaný systém. Dále by měl mít následující vlastnosti:

- sdílení systémových prostředků více aplikacím,
- souběžnost,
- jeho specifikace a všechna rozhraní musí být známá,
- uživatel by neměl poznat, zda jsou užívané prostředky sdílené nebo lokální,
- při detekování chyby by měl pokračovat v práci, i když je některá jeho součást nedostupná. [15]

1.1.1 Princip činnosti distribuovaných systémů

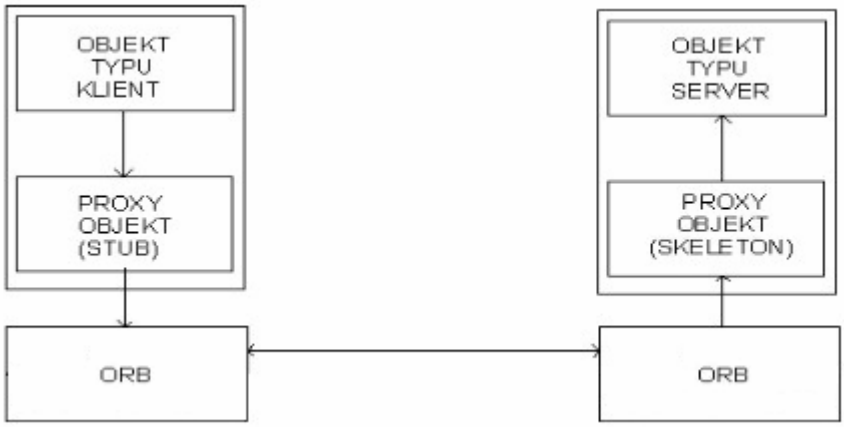
Základem fungování distribuovaných systémů je vzájemná komunikace jednotlivých výpočetních zdrojů (počítačů). Ke komunikaci využívají tyto systémy počítačovou síť a každá oblast, kde se používají, klade jiné nároky na výkonnost použité komunikační sítě, na rychlost odezvy, na množství přenášených dat. V praxi je běžné, že spolu komunikují od sebe velmi vzdálené komunikační uzly (Obr. 1), které si během komunikace vymění spoustu informací. V dnešní době, kdy programátoři v hojné míře využívají výhod

objektově orientovaného programování, jsou nejběžnějším řešením komunikace distribuovaných systémů právě objekty. Vzhledem k tomu, že komunikující objekty jsou od sebe často velice vzdáleny, používají se jejich zástupné (*proxy*) objekty. Častý problém této vzdálené komunikace, který se naskýtá, je identifikace objektu, kterému chce jiný objekt zaslat zprávu. Většina současných objektových systémů řeší identifikaci použitím adresy objektu (cílového) v operační paměti. Chceme-li tedy komunikovat s objektem ležícím v jiném adresním prostoru, vytvoříme v našem adresním prostoru zástupný objekt, který zajistí zakódování a přesun zprávy (dat) po síti. Na druhé straně však musí existovat podobný objekt, který provede přesný opak, tedy dekódování zprávy a její zaslání požadovanému cílovému objektu (Obr. 2). Zakódování a dekódování zprávy se v praxi nazývá serializace a deserializace. Zástupný objekt na straně klienta provede zakódování (serializaci) zprávy do vhodné formy pro přenos po síti. Naproti tomu zástupný objekt na straně serveru provede dekódování zprávy tzv. deserializaci a její zaslání určenému objektu a vrácení výsledků operace. Tyto zástupné objekty bývají vygenerované na základě rozhraní vzdálených objektů. [15]



Obr. 1: Komunikace vzdálených uzlů v distribuovaných systémech

Tímto jsem tedy vysvětlil pojem distribuované systémy, můžeme přistoupit k definování termínu distribuované aplikace. Jedná se o aplikace, které musí vykazovat stejné vlastnosti, jako jsou uvedeny u distribuovaných systémů. Jde tedy o aplikaci, jejíž náročnost výpočtů je rozdělena mezi více výpočetních prostředků, nejčastěji o rozdělení typu klient/server, kdy část výpočtů probíhá na serveru a část na klientské stanici. U těchto aplikací bývá velice důležitou oblastí implementování komunikace mezi jednotlivými částmi. Vzdálená komunikace je realizována prostřednictvím počítačové sítě a jednotlivé komunikující prvky se spolu dorozumívají vzájemným zasíláním zpráv.



Obr. 2: Způsob komunikace klient/server (převzato z [15])

1.2 Serializace a deserializace objektů

V mnoha případech aplikací, které vytváříme, nám stačí velmi krátká doba „života“ nějakého objektu, který použijeme a jakmile jej nepotřebujeme, odstraníme jeho instanci z paměti. U distribuovaných aplikací potřebujeme instanci objektů, které přenáší data na jiný komunikační uzel, zachovat delší dobu. To nám umožní serializace takového objektu, která nám je schopna onu instanci objektu převést na datový proud (*stream*) a uložit ji do nějakého perzistentního úložiště (relační databáze, soubor, atd.) nebo pomocí HTTP protokolu přenést na jiný počítač. V prostředí .NET Framework nám serializaci objektů zajišťuje třída `System.Serializable`. [23]

Pokud instanci objektu serializujeme a pošleme v podobě datového proudu na jiný počítač, musíme provést opačný proces tzv. deserializaci, abychom mohli s objektem pracovat, jako by byl vytvořen lokálně. Při deserializaci je z datového proudu vytvořena

instance očekávaného objektu. V prostředí .NET Framework tuto činnost zajišťuje opět třída `System.Serializable`. Serializaci a deserializaci objektů nám umožňují samozřejmě i další vývojové prostředí, které nám umožňují vytvářet distribuované aplikace např. Java, Visual Basic, a další.

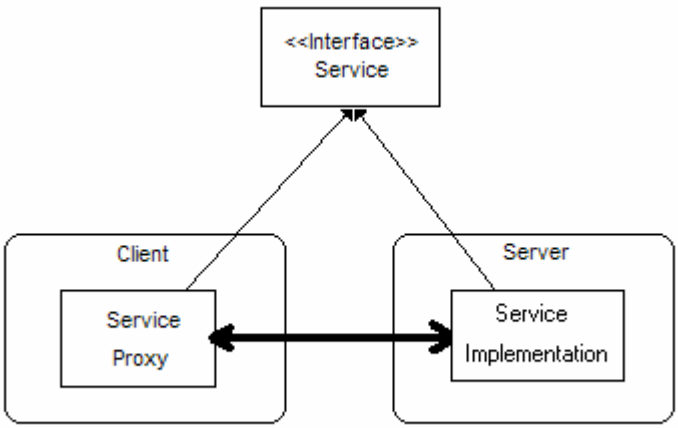
1.3 Způsoby komunikace

Jak už víme, distribuované aplikace jsou založeny na více vzájemně komunikujících uzlech a proto realizace komunikace je pro správné fungování distribuované aplikace klíčová. Stejně jako existuje několik technologií, které umožňují implementaci komerčních distribuovaných aplikací, je i několik způsobů realizace komunikace.

1.3.1 RMI

Implementace distribuovaných objektů RMI (*Remote Method Invocation*) je dostupná pouze pro jazyk Java. V technologii RMI existují vždy tři subjekty (Obr. 3):

- klientský program požadující nějakou službu,
- serverová aplikace nabízející nějakou službu,
- služba – implementace rozhraní (*interface*). [14]



Obr. 3: Subjekty technologie RMI (převzato z [14])

Rozhraní (*interface*) neobsahuje žádný kód, jedná se jen o předpis, jak má vypadat skutečný objekt. Na straně serveru je implementované rozhraní a na straně klienta je podle tohoto rozhraní implementován zástupný objekt (*proxy*). Zástupné objekty na straně klienta

jsou implementovány tak, aby přímo volali metody objektů na druhé straně. U technologie RMI jsou zástupné objekty na straně klienta reprezentovány tzv. objektem `Stub`, který provede zakódování zprávy (dat) do formy vhodné pro přenos po síti. Naproti tomu zástupné objekty na straně serveru jsou reprezentovány tzv. objektem `Skeleton`, který provádí dekódovaní zprávy, její zaslání určenému objektu a vrácení výsledků operace. Zástupné objekty klienta bývají automaticky generovány na základě rozhraní vzdálených objektů serveru. Samotný přenos dat zajišťuje programový modul ORB *(Object Request Broker)*, který implementuje všechny potřebné operace:

- navázání spojení se vzdáleným procesem,
- odesílání a příjem zpráv,
- předávání zpráv odpovídajícím zástupným objektům,
- ošetření výjimečných stavů a možných chyb při spojení,
- udržování databáze vzdálených objektů. [14]

Vždy, jakmile je zpráva doručena na server, je nutné provést přiřazení (binding) zprávy objektu, se kterým chceme komunikovat. Tento proces se skládá vždy ze dvou kroků:

- nalezení vzdáleného hostitelského uzlu,
- nalezení správného procesu na hostiteli. [14]

Důležitý u vzdálené komunikace je formát přenášených dat. Je nutné data zakódovat do takového formátu, aby se mohla zpět dekódovat s původním významem a to nezávisle na použité platformě. [14]

1.3.2 RPC

RPC (*Remote procedure call*) vzdálené volání procedur, tato technologie umožňuje programu vykonat proceduru, která je uložena na jiném místě, než běží samotný program, například na jiném počítači v síti. Tato metoda komunikace je známá ze 70. let minulého století a spadala do normy *RFC 707*. Jako první začala RPC používat společnost *Xerox* pod jménem „Kurýr“ (*Courier*) v roce 1981. Prvotní masové nasazení přišlo až se systémem Unix, konkrétně šlo o *ON RPC*. [16]

Metoda RPC funguje tak, že nejprve dojde k jednoduchému zabalení parametrů a identifikátorů procedury do podoby vhodné pro přenos mezi počítači (tzv. *marshalling*). Poté co jsou zabalená data odeslána vzdálenému počítači, dojde k jejich rozbalení a zjištění o jakou proceduru jde (tzv. *unmarshalling*). Následně dojde k zavolání procedury, zabalení a odeslání výsledku zpět. Klientský počítač výsledek rozbalí a přijatá hodnota se předá proceduře. [16]

1.3.3 Technologie COM/DCOM

COM (*Component Object Model*) je platforma firmy Microsoft, která zahrnuje jak specifikaci, tak i implementaci a je dostupná pro více programovacích jazyků např. C/C++, Visual Basic, Java. Přestože firma Microsoft tuto technologii vyvinula pro prostředí Windows, lze v něm teoreticky programovat i v jiných dostupných operačních systémech. V praxi se ovšem s touto technologií na jiných operačních systémech nesetkáme, protože pro ně jsou standardem jiné používané objektové modely. Tento standard nám určuje základní vlastnosti objektů, pravidla pro práci s nimi a pevně stanovuje protokol, dle kterého spolu objekty komunikují. Dále nám také umožňuje vytvářet komponenty přenositelné mezi různými aplikacemi. [13]

Každá taková komponenta má definované své rozhraní, které je označeno unikátním identifikátorem. Jedna komponenta může poskytovat více rozhraní, a proto je možné zajistit kompatibilitu verzí. Aplikace může prostřednictvím tohoto identifikátoru kontaktovat komponentu a vyžádat si popis rozhraní. Potom teprve může volat metody komponenty. [13]

DCOM (*Distributed extension of the Component Object Model*) je pouze rozšíření standardního modelu COM a možnost komunikace s objekty na vzdálených počítačích a tím je umožněno plnohodnotně vytvářet distribuované aplikace. [13]

Komponenty vytvořené technologií COM se vyznačují následujícími vlastnostmi:

Modularita – každý objekt je pro své okolí uzavřená „černá“ skříňka (*black box*) a ostatní objekty se nemusí zajímat o její interní funkcionalitu. S objekty mají určené metody, pomocí nichž je možné s nimi manipulovat. V průběhu vývoje aplikace se mohou objekty měnit, ale jeho rozhraní musí zůstat zachováno v původní podobě. [13]

Univerzálnost – Nezávislost na programovacím jazyku. Komponenty lze použít ve všech jazycích a kompilátorech, které umí vytvořit kód odpovídající COM standardu a podporují pointery a objektové programování. Deklarace objektů a jejich metod je řešena pomocí IDL (*Interface Definition Language*) univerzálního deklaračního jazyka, který jednotlivé kompilátory snadno přeloží do svého jazyka. [13]

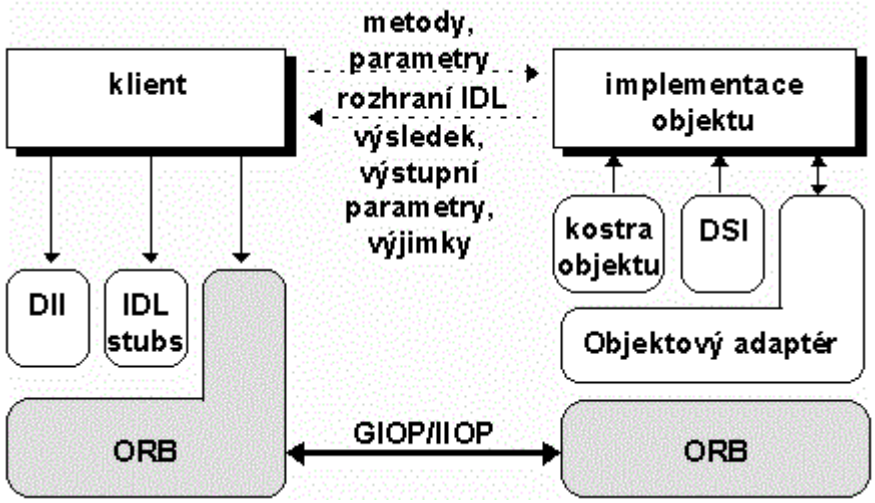
Správa paměti – Životní cyklus komponent se řídí tzv. *referencemi* - počtem pointerů, které na daný objekt ukazují. Klient pouze zvyšuje či snižuje počet referencí dle jednoduchých pravidel a klesne-li počet referencí na nulu, objekt se automaticky postará o vymazání sama sebe z paměti. [13]

Výhodou standardu DCOM je možnost používat tzv. *ActiveX* objekty, které jsou zaregistrovány na jiném počítači. Tyto objekty musí být zaregistrovány jak na počítači, na kterém běží klientská aplikace, tak na straně serveru. Na serveru navíc musí být samotný *ActiveX* objekt a povoleno jeho vytvoření z jiného počítače. [13]

1.3.4 Technologie CORBA

CORBA (*Common Object Request Brooker Architecture*) tvoří ucelené prostředí pro tvorbu objektově orientovaných distribuovaných aplikací. Tento objektový model je jazykově nezávislý a specifický pro vývoj distribuovaných aplikací. Standard CORBA je postupně vyvíjen od počátku 90. let společnostmi soustředěnými ve sdružení OMG *(Object Management Group).* Významným mezníkem ve vývoji byla dohoda na standardu CORBA verze 2.0 v roce 1995. Tento standard přinesl společně s dalšími novinkami definici protokolu komunikace mezi implementacemi různých výrobců, tím byla odstraněna další z překážek bránící šíření architektury CORBA. [12]

Základní funkcí této architektury je podpora jazykově neutrálního použití distribuovaných objektů. Klientská aplikace může využívat dostupné objekty bez ohledu na to, v jakém jazyce jsou implementovány, kde a na jakém počítači běží a jakým komunikačním protokolem jsou dostupné. Objektem se rozumí identifikovatelná, zapouzdřená entita, která poskytuje nějaké funkce a klient přistupuje k těmto funkcím zasíláním požadavků. Forma generovaných požadavků je závislá na programovacím jazyce, ve kterém je klientská část implementována. V objektově orientovaných jazycích to může být volání metod zástupného objektu, zatímco ve funkcionálních jazycích volání vygenerovaných funkcí. [12]



Obr. 4: Architektura objektového modelu CORBA (převzato z [12])

Celková implementace prostředí CORBA se skládá z klientského prostředí, prostředí na straně serveru a objektově orientovaných služeb. Hlavní komponenty architektury jsou znázorněny na obrázku (Obr. 4). Jednotlivé komponenty mají následující funkce:

- **Implementace objektu:** Kód objektu, který implementuje zveřejněné služby objektu. Implementace může být napsána v libovolném podporovaném jazyce (obvykle C, C++ nebo Java). Rozhraní služeb objektu je definováno v jazyce IDL (*Interface Definition Language*).
- **Klient:** Aplikace využívající vzdálené objekty. Pro použití vzdáleného objektu musí mít dostupnou jeho definici rozhraní v jazyce IDL v podobě překladu nebo při dynamickém volání za chodu programu a jednoznačnou adresu objektu (IOR).
- **IDL stubs (spojky):** Kód vygenerovaný kompilátorem jazyka IDL, který propojuje uživatelský kód s agentem ORB. V jazyce C++ má spojka formu zástupné třídy, jejíž metody může klientský kód přímo volat.
- **DII (Dynamic Invocation Interface):** Klient může používat také objekty, ke kterým získá definici rozhraní za běhu programu. Rozhraní pro dynamické volání metod dovoluje generovat dynamické požadavky.
- **ORB (Object Request Broker):** Zprostředkovatel objektových služeb zahrnuje veškeré vnitřní mechanismy pro vyhledání požadovaného objektu, generování a přenos požadavků, parametrů a výsledků na úrovni komunikace mezi systémy.

ORB může používat různé metody komunikace, včetně přímé aktivace objektů v rámci jednoho adresového prostoru.

- **Objektový adaptér:** Objektový adaptér propojuje implementaci objektu s agentem ORB, demultiplexuje přicházející požadavky, aktivuje objekty a předává jim požadavky prostřednictvím volání metod kostry objektu.
- **Kostra objektu:** Je vygenerována kompilátorem jazyka IDL, slouží jako bázová třída odpovídající definici objektu v jazyce IDL.
- **DSI (Dynamic Skeleton Interface):** Dynamicky vytvořená kostra objektu, obdoba DII na straně klienta. Typickým použitím je most pro transformaci požadavků z jednoho komunikačního protokolu do jiného nebo firewall.
- **GIOP (General Inter ORB Protocol):** Protokol komunikace mezi různými ORB. Je definován nad běžným spojovaným transportním protokolem. Konkrétní implementace nad protokolem TCP/IP je definována jako IIOP (Internet Inter-ORB Protocol). [12]

Při vývoji aplikací v prostředí CORBA je potřeba dodržet následující jednotlivé kroky:

1. Zápis rozhraní objektu v jazyce IDL.
2. Vygenerování spojek a kostry kompilátorem jazyka IDL.
3. Návrh klientské aplikace v implementačním jazyce.
4. Implementace objektu a hlavního programu serveru. [12]

Architektura OMG CORBA je poměrně složitá, ale na druhé straně otevřená a za mnoho let vývoje také koncepčně promyšlená. Bohužel otevřenost architektury je v některých případech na újmu přenositelnosti, protože standard CORBA 2.0 se některým implementačním detailům vyhýbá. Implementace různých výrobců tak sice mohou deklarovat dodržení standardu, ale jsou mezi nimi velké rozdíly, zvláště pak v rozhraní BOA a ORB. Přenositelnost implementací objektů a klientů na úrovni implementačního jazyka tím samozřejmě trpí. [12]

1.4 Výhody a nevýhody distribuovaných aplikací

Jednou z předností použití distribuované aplikace je možnost rozdělení zatížení zpracování výpočetních procesů mezi jednotlivé počítače. Díky tomu je možné i velmi složité úlohy rozdělit na několik relativně jednodušších úloh. Další výhodou je snadnější a přehlednější další rozšiřování aplikace pro programátory a v neposlední řadě nasazení nové verze aplikační části na serveru bez nutnosti instalace nové klientské aplikace na klientských stanicích. Ovšem použití distribuovaných aplikací přináší i některé problémy jako je např. vyšší náklady na provoz a údržbu. Některé problémy obnáší i samotný vývoj distribuovaných systémů.

1.4.1 Problém přiřazování

Prvním krokem při vzdálené komunikaci je samozřejmě vyhledání objektu, se kterým chceme komunikovat. Tomuto procesu se říká přiřazování (*binding*) a zahrnuje následující dva kroky:

- nalezení vzdáleného hostitelského uzlu,
- nalezení správného procesu na hostiteli.

Přirozeným řešením tohoto problému je vytvoření centrální databáze vzdálených objektů. Tedy objekt, který chce poskytovat ostatním své metody (vzdálené) se pomocí ORB zaregistruje v databázi vzdálených objektů pod smluveným jménem. Jiný objekt se pak (opět prostřednictvím ORB) může pokusit zadané jméno vyhledat. Je-li vzdálený objekt v databázi nalezen, ORB se poté postará o vytvoření zástupného objektu. [15]

1.4.2 Problém předávání dat

Jedním z hlavních problémů je formát přenášených dat, které je třeba zakódovat do takového formátu, aby je bylo možné zpět dekódovat s původním významem a to nezávisle na použité platformě. Číselné hodnoty se kódují jako řada bytů a problémem je vyřešit pořadí bitů závislé na architektuře. Ukazatele jsou problémem, neboť ve své podstatě nereprezentují nic jiného než adresu v paměti. Přímým předáním ukazatele do jiného adresního prostoru by došlo k nesmyslnému chování systému. Předání ukazatele je z tohoto důvodu třeba provést ve více krocích:

- ORB na straně odesílatele zjistí velikost dat, na které ukazatel míří,

- ORB na straně příjemce alokuje v adresním prostoru odpovídající blok,
- dojde k přenosu dat k příjemci a jejich uložení do alokovaného prostoru,
- předá se zpráva cílovému objektu s tím, že adresa uložená v ukazateli je nahrazena adresou alokovaného bloku,
- po vykonání operace (metody) se data přenesou zpět a uloží na místo původních dat,
- pomocný alokovaný prostor na straně příjemce je uvolněn. [15]

2 TECHNOLOGIE WINDOWS COMMUNICATION FOUNDATION

Windows Communication Foundation (WCF) je platforma pro distribuované programování moderních aplikací v prostředí .NET Framework a operačním systému Windows a obsahuje několik mechanizmů na ochranu přenášených informací a správu přístupu. WCF je založena na komunikaci pomocí zpráv, kde zpráva je skupina dat, která obsahuje záhlaví a tělo zprávy. Zprávou může být například HTTP požadavek nebo Microsoft Message Queuing (MSMQ).

WCF je technologie pro vývoj Service-Oriented Architecture (SOA) aplikací a základním prvkem je tedy služba. Klient je webová nebo desktopová aplikace implementovaná na platformě .NET a iniciuje komunikaci směrem ke službě. Služba je aplikace, která očekává požadavky klienta, aby na ně mohla zareagovat. Okolím je služba WCF vnímána jako jeden nebo kolekce koncových bodů tzv. endpointů, které vystavují rozhraní (*interface*) a definují strukturu posílaných zpráv klientem.

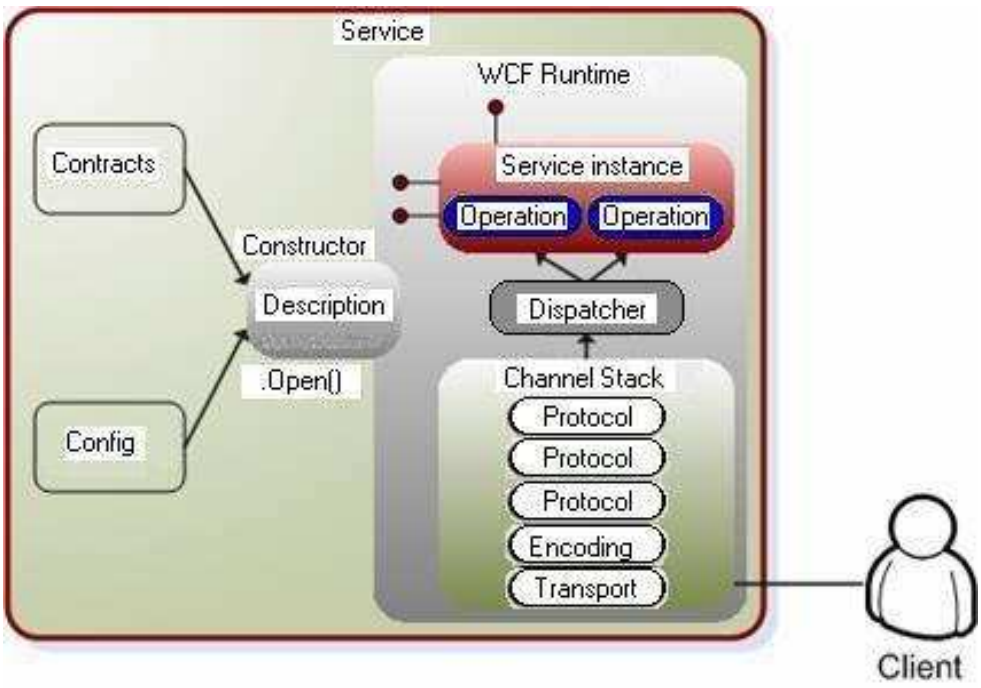
V minulosti jsme měli pro vytváření SOA aplikací několik možností. Zpravidla platilo, že program byl napsán podle toho, jakou technologii jsme pro vývoj použili. Těmito možnostmi je například ASP.NET Web Services, Web Services Enhancements (WSE), .NET Remoting, Enterprise Services, MSMQ a další. Nevýhodou těchto technologií je to, že vývoj aplikace je závislý na použité technologii a jednotlivé technologie nejsou vzájemně kompatibilní. Oproti starším technologiím, které se využívají pro vývoj SOA aplikací je implementace WCF z hlediska programátora mnohem snazší a z hlediska vývoje moderního softwaru rychlejší a lépe udržovatelná. WCF je technologie, která všechny dřívější technologie v prostředí .NET sjednocuje. Programátor tak nemusí znát model dané technologie, ale stačí mu pochopit koncepci WCF. A další velkou výhodou je jednoduchá možnost nastavit bezpečnost a další vlastnosti služby. ([1], part 1), ([3], part 1), [18]

2.1 WCF služba

Jde o systém, který poskytuje jeden nebo více vzdálených koncových bodů (tzv. endpointů), se kterými komunikují klienti prostřednictvím vzájemného posílání

tzv. SOAP[1] zpráv. Služba zveřejňuje metadata a tvoří ji tři základní části, její struktura je zobrazena na obrázku (Obr. 5):

- třída služby, což je její samotná implementace,
- prostředí, kde služba poběží,
- množina koncových bodů (endpointů).



Obr. 5: Struktura WCF služby (převzato z [24])

Koncové body (endpointy) jsou ve službě místem, které slouží ke komunikaci s klienty a zajišťují odesílání a přijímání zpráv. Koncové body obsahují následující tři části:

- *address* – adresa udává, kde služba běží, tedy kam budou zasílány zprávy.
- *binding* – říká nám, jakým způsobem bude služba komunikovat, tedy jaký komunikační protokol je zvolen, jaké kódování, ale také výběr bezpečnosti, transakcí atd.

[1] SOAP Simple Object Access Protocol je protokol, který umožňuje výměnu zpráv založených na XML prostřednictvím počítačové sítě zejména pomocí HTTP protokolu. ([5], s. 564), ([6], s. 623)

- *contract* – specifikuje rozhraní, které služba poskytuje, jeho metody a další. Je nezávislý na volbě adresy a bindingu. [18], ([2], chapt. 2)

SOAP zpráva je zasílána mezi klientem a serverem (službou) a je nezávislá na přenosovém protokolu. Je to vlastně požadavek a odpověď po nějakém komunikačním kanále. Přesnější model posílání zpráv (*messaging patterns*), které WCF podporuje, je následující:

- *one way* – klient odešle zprávu službě a neočekává odpověď,
- *request-response* – klient pošle požadavek a čeká na odpověď,
- *duplex* – obousměrná komunikace mezi klientem i službou probíhající asynchronně (služba může vynutit spuštění metody na straně klienta). ([5], s. 564), ([6], s. 623)

Metadata popisují službu a tím specifikují všechny údaje důležité k tomu, aby na jejich základě mohl být nakonfigurován klient. Díky tomu klient ví, na jakém protokolu a adrese služba běží. *Hostovací prostředí* je místo, kde služba poběží, kde bude tzv. hostována. WCF služba může být hostována například v Internetové Informační Službě (IIS), klasickém Windows procesu (služba) nebo také jako tzv. *self-hosting*, což může být prakticky jakákoliv aplikace (konzolová, WinForm, WPF). *Kanál (channel)* je prostředí, ve kterém se přenášejí zprávy. Toto prostředí je vytvořeno v okamžiku, kdy klient zašle nějakou zprávu koncovému bodu služby. *Proxy* je prostředník mezi klientem a serverem a používá se na straně klienta vždy, i když jde jen o komunikaci na jednom počítači v rámci paměti. Pokud chce klient komunikovat se serverem, musí si vždy vytvořit instanci *proxy* třídy. Metody, které služba zveřejňuje jako proveditelné, jsou pak součástí *proxy* třídy a klient je tak může volat. ([1], part1), ([2], chapt. 2) ([3], chapt. 4), [18]

2.1.1 Binding ve WCF službě

Jde o proces vyhledání správného objektu na serveru (službě), se kterým chceme vzdáleně komunikovat. Binding se skládá z následujících částí:

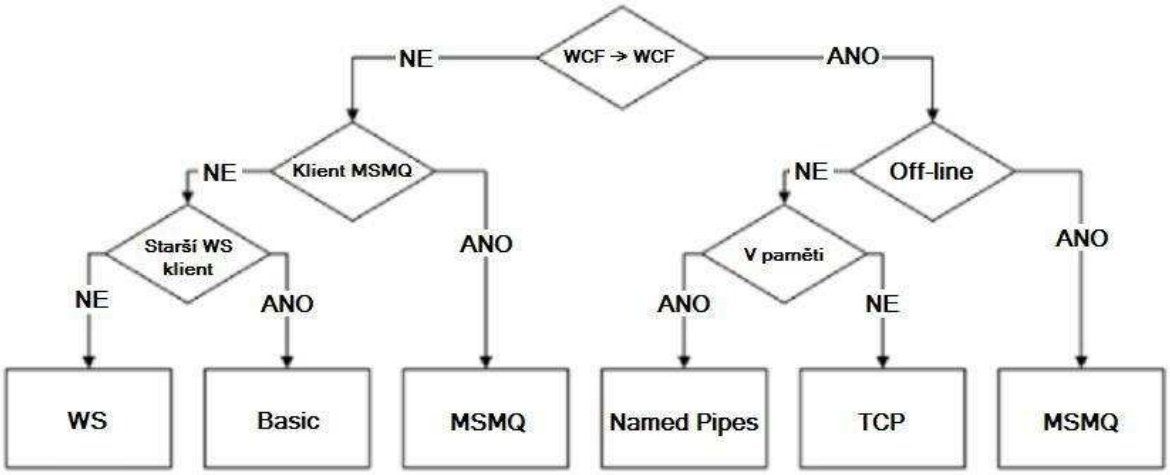
- *transport* (komunikační protokol),
- *encoding*,
- *security*,
- *reliable sessions*,

- *transakce*.

Transport je komunikační protokol, který se použije pro komunikaci služby s klientem. Při konfiguraci služby je nutné zvolit správný typ bindingu, kterých existuje ve WCF technologii několik:

- *BasicHttpBinding* – pro komunikaci webových služeb splňujících *WS-Basic Profile*,
- *WSHttpBinding* – zabezpečený a interoperabilní binding bez podpory duplex kontraktů,
- *WSDualHttpBinding* – s podporou duplex kontraktů,
- *WSFederationHttpBinding* – podpora protokolu *WS-Federation*,
- *NetTcpBinding* – zabezpečený a optimalizovaný binding pro komunikaci WCF aplikací (velmi využívané),
- *NetNamedPipeBinding* – komunikace WCF aplikací v rámci jednoho PC (velmi rychlé),
- *NetMsmqBinding* – komunikace pomocí MSMQ (message queue – velmi spolehlivé doručování zpráv, často používané),
- *NetPeerTcpBinding* – vícepočítačová komunikace,
- *MsmqIntegrationBinding* – komunikace mezi WCF aplikací a již existující MSMQ aplikací.

Pokud by Vašim požadavkům nevyhovoval žádný z nabízených bindingů, můžete si vytvořit i vlastní (*custom binding*). Důležité je vybrat správný binding, který použijeme pro komunikaci. Při výběru bindingu je důležité si uvědomit, jak bude služba komunikovat s klientem a zda jsou obě aplikace implementovány v technologii WCF. Blíže to vysvětluje obrázek (Obr. 6). [6], ([2], chapt. 2), [17]



Obr. 6: Schéma výběru bindingu ve WCF (upraveno z [17])

V dalším kroku je nutné provést nastavení chování služby (tzv. *service behavior*), což znamená nastavit následující parametry:

- *Instancing* – způsob vytváření instancí tříd služby,
- *Concurrency* – nastavení vztahu vláken a instancí,
- *Throttling* – nastavení limitu současně existujících instancí, *session* a volání,
- *Error handling* – zpracovávání chyb,
- *Metadata* – poskytování metadat,
- *Session* – nastavení vytváření *session*,
- *Security* – zabezpečení služby,
- *Transactions* – nastavení transakčního vykonávání operací služby. ([2], chapt. 2), [17]

Pro vytváření instance třídy služby (*Instancing*) nám technologie WCF nabízí tři módy:

- *Per call* – při každém požadavku klienta je vytvořena nová instance třídy služby. Jakmile je požadavek zpracován a služba odešle výsledek, její instance je zrušena. Tento způsob použijeme v případě, že nepotřebujeme uchovávat stavy mezi jednotlivým voláním operací.
- *Per session* – pro každé nové volání klienta je vytvořena nová instance služby (vznikne nový klientský komunikační kanál), stav je tedy uchován po dobu trvání

spojení klienta se službou. Tento způsob použijeme tehdy, pokud naopak chceme udržovat stavy po dobu trvání práce klienta se službou.

- *Single* – u tohoto módu existuje jen jedna instance služby, která obsluhuje požadavky od všech klientů. Tento způsob využijeme, pokud chceme, aby všichni klienti pracovali se společnými hodnotami. [17]

Nastavení vztahu vláken a instancí (*Concurrency*) realizujeme v případě, že k instanci služby bude přistupovat více vláken. Máme na výběr rovněž ze tří módů:

- *Single* – v daném okamžiku může k instanci služby přistoupit pouze jedno vlákno.
- *Reentrant* – stejně jako u single může v daném okamžiku k instanci služby přistoupit pouze jedno vlákno, ale jsou povolena i zpětná volání. Používá se v případech, kdy vlákno první služby volá druhou službu a vlákno druhé služby zpět volá první službu.
- *Multiple* – v daném okamžiku k instanci služby může přistupovat více vláken.

Nastavení limitu současně existujících instancí (*Throttling*) nám vymezuje, kolik systémových zdrojů služba bude využívat. [17]

2.2 Implementace WCF služby

Pro správné fungování služby je velmi důležité správně implementovat rozhraní služby, pomocí kterého s ní budou komunikovat klienti. Při vytváření WCF služby musí programátor definovat tzv. kontrakty (contracts), které jsou čtyři základní typy:

- *service contract* – kontrakt služby,
- *data contract* – kontrakt dat,
- *message contract* – kontrakt zpráv,
- *fault contract* – kontrakt chyb. [17], ([5], s. 567 - 569)

Service contract definuje rozhraní pro operace, které bude klient vzdáleně vyvolávat. Na začátku rozhraní je označen atributem `[ServiceContract]` a jednotlivé metody atributem

`[OperationContract]`. Pomocí tohoto označení klient pozná, které funkce může využívat. Příklad implementace jednoduchého rozhraní je znázorněn v následujícím kódu[2]. [17], ([5], s. 569)

```
[ServiceContract]
public interface IUser
{
      [OperationContract]
      string GetUsers();

      [OperationContract]
      string GetUserById(int id);

      [OperationContract]
      void Delete(int id);
}
```

Data contract určuje, jaké datové typy se budou používat při komunikaci s klientem, a popisuje strukturu přenášených dat. Pokud nám nebudou stačit běžné datové typy jazyka (`int, string`), musíme definovat tento kontrakt. V takovém případě musí být třída označena atributem `[DataContract]` a jednotlivé členy třídy atributem `[DataMember]` jak je znázorněno v následujícím kódu[3]. [17], ([5], s. 570)

```
[DataContract]
public class User
{
      private int id = 0;
      private string firstName = "Jan";
      private string lastName = "Novák";
      [DataMember]
      public int Id
      {
        get { return id; }
        set { id = value; }
      }
```

[2] Zdrojový kód je napsán v programovacím jazyce C#, který je jedním z jazyků prostředí .NET Framework
[3] Zdrojový kód je napsán v programovacím jazyce C#, který je jedním z jazyků prostředí .NET Framework

```
      [DataMember]
      public string FirstName
      {
        get { return firstName; }
        set { firstName = value; }
      }
      [DataMember]
      public string LastName
      {
        get { return lastName; }
        set { lastName = value; }
      }
}
```

Message contract definuje strukturu SOAP zprávy a může být typový nebo netypový. Umožňuje specifikovat, zda zpráva bude obsahem hlavičky nebo těla zprávy. Na rozdíl od *Service contract* a *Data contract*, které jsou určeny ke specifikaci operace služby, není určen pro znovupoužití a sdílení. Každá třída musí mít ve své deklaraci atribut `[MessageContract]` a jednotlivé části zprávy atributy `[MessageHeader]` jako hlavičku a `[MessageBody]` jako tělo zprávy jak ukazuje následující ukázka zdrojového kódu[4]. [17]

```
[MessageContract]
public sealed class Message
{
      private string nazev;
      private string popis;

      [MessageHeader]
      public string Nazev
      {
        get { return nazev; }
        set { nazev = value; }
      }
```

[4] Zdrojový kód je napsán v programovacím jazyce C#, který je jedním z jazyků prostředí .NET Framework

```
      [MessageBody]
      public string Popis
      {
        get { return popis; }
        set { popis = value; }
      }
}
```

Fault contract je poslední kontrakt, o kterém budeme hovořit, slouží ke zpracování chyb vyvolaných službou a rozlišení, zdali a jak budou zaslány klientovi. Ošetření výjimek na straně serveru a odeslání na klienta je velice důležitá činnost, protože klient by měl mít informace o tom, co se na serveru stalo, zejména v případě dojde-li ve službě k nějaké chybě. Pro ošetření výjimek ve službě máme dvě možnosti. V prvním případě se vytvoří objekt `FaultException`, kterému se v konstruktoru předá obsah zprávy a následně je chyba odeslána na klienta jak ukazuje následný kód[5]. [25]

```
if (value < 0)
{
      var faultCode = new FaultCode("Error");
      MessageFault messageFault = MessageFault.CreateFault(faultCode,
      "Value must be greater than zero.");
      throw new FaultException(messageFault);
}
```

Kde `faultCode` je proměnná s názvem chybového kódu třídy `FaultCode`. Pomocí nové instance `MessageFault` a její metody `CreateFault` vytvoříme z chybového kódu a chybové hlášky samotnou chybu, kterou budeme chtít odeslat klientské aplikaci. Nakonec vyvoláme výjimku `FaultException`, která se zašle klientské aplikaci. Tuto výjimku musíme šetřit na straně klientské aplikace, protože jinak bychom ji nemohli odchytit. To již probíhá jako klasické odchytávání chyb v .NET Framework viz. následující zdrojový kód[5].

```
try
{
      Console.Write("Write number: ");
      Console.WriteLine(proxy.GetNumber(Console.ReadLine()));
```

[5] Zdrojový kód je napsán v programovacím jazyce C#, který je jedním z jazyků prostředí .NET Framework

```
}
catch (FaultException ex)
{
      Console.WriteLine(ex.Message);
}
```

V případě vyvolání výjimky se nám do klientské aplikace vypíše chybové hlášení: „`Value must be greater than zero`“. [25]

Druhou možností je použití generického typu výjimky `FaultContract`. Vytvoříme vlastní objekt výjimky pomocí atributů `DataContract` a každé metodě, která bude vyvolávat naši vlastní výjimku, musíme v definici rozhraní přidat atribut `FaultContract` viz. zdrojový kód. [25]

```
[OperationContract]
      [FaultContract(typeof(MyException))]
      int GetNumber(int value);
```

Následuje implementace třídy vlastní výjimky.

```
[DataContract]
    public class CustomException
    {
        [DataMember]
        public string ExceptionCode;

        [DataMember]
        public string ExceptionMessage;
    }
```

Předchozí zdrojový kód, kde jsme v metodě služby vyvolali výjimku `FaultException` musíme upravit tak, aby vrácená výjimka byla typu `CustomException`, kterou jsme implementovali. [25]

```
if (value < 0)
{
      var ce = new CustomException
      {
            ExceptionCode = "Error",
            ExceptionMessage = "Value must be greater than zero."
      };
      throw new FaultException<CustomException>(ce, ce.ExceptionCode);
```

```
}
```

Proměnná `ce` je instance třídy naší vlastní výjimky a její vlastnosti `ExceptionCode` a `ExceptionMessage` jsme naplnili potřebnými daty. V našem případě pouze řetězci s chybovou hláškou a kódem chyby. Výjimku jsme nakonec vyvolali s generickým typem `CustomException`, kde nám jako parametry slouží samotná chyba (její text) a její kód. Upravíme i odchytávání výjimky na straně klienta viz. následující kód. [25]

```
try
{
      Console.Write("Write number: ");
      Console.WriteLine(proxy.GetNumber(Console.ReadLine()));
}
catch (FaultException<CustomException> ex)
{
      Console.WriteLine(ex.ExceptionMessage);
}
```

2.3 Možnosti provozování WCF služby

Použití technologie WCF má velkou výhodu ve variabilnosti provozování samotné služby. Zatímco např. webové služby je možné provozovat pouze na webovém serveru u WCF služby máme na výběr z několika následujících možností.

- **Internet Information Services (IIS):** Tento způsob je obdoba provozování webových aplikací nebo webových služeb na IIS. Při prvním požadavku klienta automaticky vytvoří proces. IIS kontroluje v jakém stavu je proces služby a podle toho reaguje na změny, které nastanou, např. když delší dobu neodpovídá tak jej recykluje. Jedna ze zásadních nevýhod je, že IIS podporuje pouze HTTP protokol.
- **Windows Service:** Jde o Windows proces, který vytvoří prostředí pro běh služby a je ji možné řídit jako všechny ostatní služby v systému (spustit manuálně nebo automaticky, pozastavit, restartovat). Také máme k dispozici výpis událostí (tzv. log soubor), které ve službě nastaly. Velkou výhodou u tohoto způsobu hostování WCF služby je podpora všech dostupných komunikačních protokolů.
- **Self hosting:** V tomto případě je zodpovědný za životní cyklus hostujícího procesu služby zdrojový kód, který implementuje samotný programátor. Hostování má

v plné režii jakýkoliv Windows proces, tedy jakákoliv konzolová, WinForm nebo WPF (*Windows Presentation Foundation*)[6] aplikace. ([2], chapt. 15), ([1], part 2)

Každý z uvedených způsobů hostování WCF služby má své výhody i nevýhody a je velice důležité zvolit správný způsob, který závisí na konkrétním použití technologie WCF a povaze systému.

2.4 Struktura WCF security

Při komunikaci mezi klientem a serverem (službou) jsou posílány *Simple Object Access Protocol* (SOAP) zprávy jak je znázorněno na obrázku (Obr. 7), takže zabezpečení této komunikace musí být na úrovni posílaných zpráv tzv. *message-level security* tak, jak je zobrazeno na obrázku (Obr. 8 a Obr. 10). Služba musí být autentifikována pomocí certifikátu ověřeného certifikační autoritou (zkráceně též CA) a klient se ověří na základě uživatelského jména a hesla, které musí být šifrované. Základní složky zabezpečení WCF tedy jsou:

- zabezpečení přenosu (*transfer security*),
- kontrola přístupu (*access control*),
- logování důležitých událostí (*auditing*). [28]

Pro zabezpečení přenosu nabízí WCF dvě základní možnosti (módy):

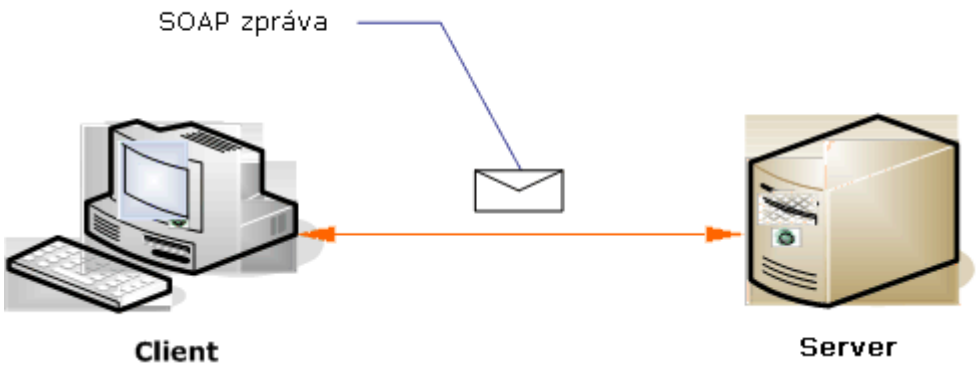
- *transport security mode* – zabezpečení je implementováno na úrovni přenosového protokolu např. HTTPS,
- *message security mode* – zabezpečení je implementováno na úrovni SOAP zpráv např. *WS-Security*. [28], [29], [31]

WCF podporuje tyto základní typy ověřování (*credential*):

- *Windows* – využívá ověřování systému Windows,

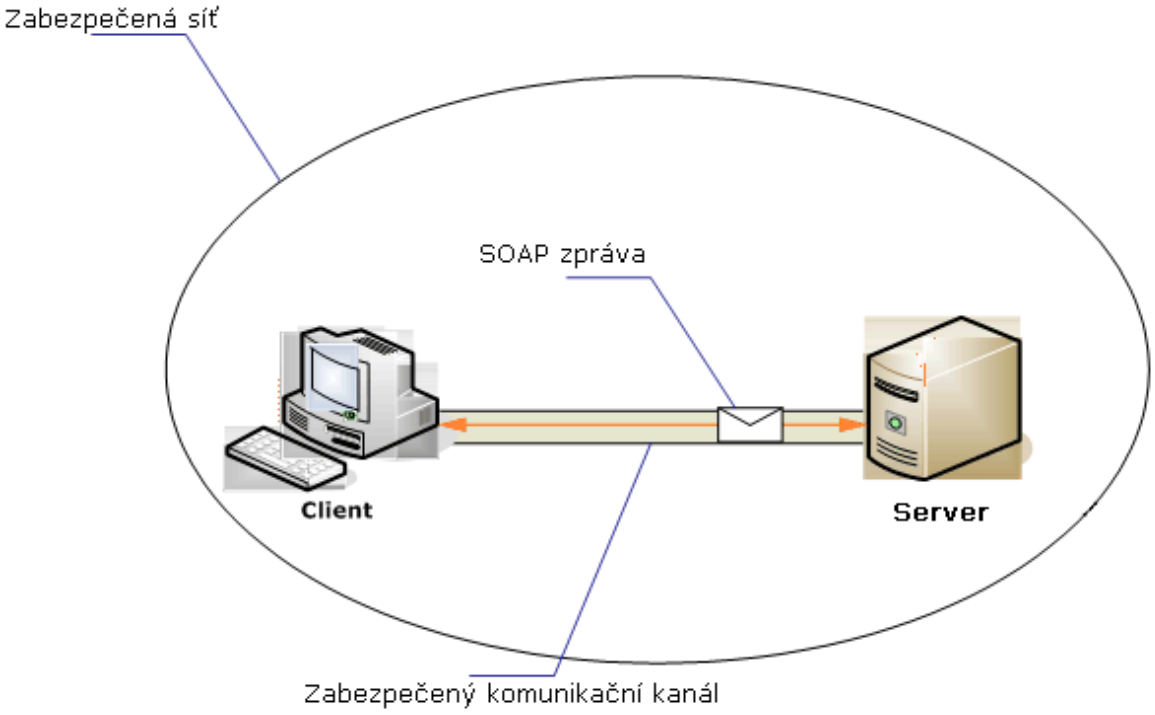
[6] WPF je technologie v prostředí .NET Framework pro vytváření moderního graficky pokročilého uživatelského rozhraní

- *Certificate* – provádí ověřování klienta pomocí certifikátu ve formátu *X.509*[7],
- *Digest* – způsob ověřování bez šifrování zprávy, pouze na úrovni ověřovací metody,
- *Basic* – určuje pouze základní ověření pro klienta, není příliš bezpečné,
- *Password* – při ověřování klienta je vyžadováno uživatelské jméno a heslo. Ověření se provádí pomocí ověřování systému Windows nebo jiného vlastního ověřovacího řešení,
- *NTLM* (zkratka *NT LAN Manager*) – autentizační síťový protokol pro ověření uživatele nebo spojení. [29], [31]



Obr. 7: Znázornění komunikace mezi klientem a serverem (převzato z [26])

[7] X.509 je standard pro systémy založené na veřejném klíči k jednoduchému podepisování.

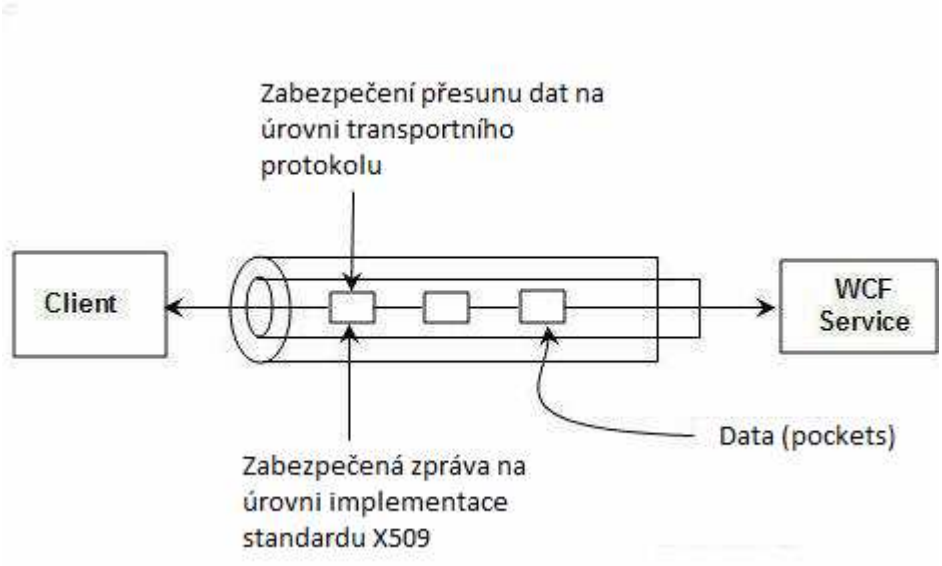


Obr. 8: Znázornění zabezpečené komunikace (převzato z [27])

Samotná komunikace mezi klientem a serverem probíhá ve většině případů po internetu a to přináší řadu hrozeb z hlediska bezpečnosti přenášených informací. Velice problematickou záležitostí bývá získání citlivých informací z internetového bankovnictví, kdy při zadání bankovního převodu útočník zachytí tuto zprávu a pokud není komunikační kanál dostatečně zabezpečen, může dojít ke zcizení a následnému zneužití těchto dat. Někdy útočníkovi nejde o odcizení přístupových údajů, ale jeho snahou je změnit obsah zprávy ještě než je doručena koncovému bodu např. změnit číslo účtu, na který chce uživatel převést peníze. Z těchto důvodů a i kvůli mnoho dalším, je důležité při vytváření služby pro vzdálenou komunikaci dodržet čtyři následující zásady (Obr. 9):

- Důvěrnost (*Confidentiality*) nám zajišťuje, že informace při cestě mezi klientem a serverem se nedostanou jinam,
- Integrita (*Integrity*) zaručuje, že příjemce obdrží data odesílatele v nezměněné podobě,
- Autentizace (*Authentication*) ověří odesílatele a příjemce,
- Autorizace (*Authorization*) prověřuje, zda je uživatel oprávněn provádět akce, které požaduje od aplikace. [30]

Obr. 9: Zásady zabezpečení služeb (převzato z [30])



Obr. 10: Schéma zabezpečení přenášených dat (upraveno z [30])

3 AUTENTIZACE KLIENTA

Jde o proces ověření uživatele (klienta), který žádá o přístup k nějakému systému, v případě WCF ke službě. Při autentizaci je cílem rozhodnout zda identita na druhé straně komunikačního kanálu, která žádá o spojení se službou je opravdu ta osoba, za kterou se vydává. Při ověřování uživatele jsou využívány jeho identifikační údaje, které jsou ověřeny interním mechanizmem systému. V případě, že identifikační údaje klienta neodpovídají, je žadateli přístup odepřen. Pokud autentizace proběhne úspěšně, je uživateli povoleno přihlášení do systému a je mu přidělena identita a většinou i role v systému.

Autentizace patří k nejdůležitějším činnostem při zabezpečení aplikací využívajících WCF. V prostředí .Net je možné autentizaci uživatele provést třemi základními způsoby, které jsou následně popsány a zmíněn je i způsob případné konfigurace nebo implementace.

3.1 Použití ASP.NET Membership Provider

Jedná se o technologické prostředky, které umožňují vytvářet a spravovat jednotlivé uživatelské účty. Technologie Membership Provider využívá pro práci s uživateli vlastní databázi, která může být integrována do databáze, kterou používá Vaše aplikace, což je i doporučováno. Prostředí Membership Provider nabízí velké množství funkcí, které umožňují snadno pracovat s uživatelskými informacemi zejména je důležitá funkce přidělování ověřovacích rolí a samotné ověřování uživatele. Při použití tohoto způsobu autentizace je nutné upravit konfiguraci Vaší klientské aplikace. V konfiguračním souboru najděte element **<**`system.web`**>** a přidejte do něj následující kód. [32]

```
<membership defaultProvider="SqlMembershipProvider"
userIsOnlineTimeWindow="15">
  <providers>
    <clear />
      <add
        name="SqlMembershipProvider"
        type="System.Web.Security.SqlMembershipProvider"
        connectionStringName="SqlConn"
        applicationName="MembershipAndRoleProviderSample"
        enablePasswordRetrieval="false"
        enablePasswordReset="false"
        requiresQuestionAndAnswer="false"
```

```
        requiresUniqueEmail="true"
        passwordFormat="Hashed" />
  </providers>
</membership>
```

Dále je nutné provést konfiguraci v části konfiguračního souboru v elementu `<system.serviceModel>`, kde provedeme nastavení služby tímto kódem.

```
<system.serviceModel>
<bindings>
  <wsHttpBinding>
    <binding name="MembershipBinding">
      <security mode ="Message">
        <message clientCredentialType ="UserName"/>
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
</system.serviceModel>
```

Jako poslední je důležité nastavit způsob ověřování. [32]

```
<serviceCredentials>
      <userNameAuthentication userNamePasswordValidationMode
      ="MembershipProvider"
      membershipProviderName ="SqlMembershipProvider"/>
</serviceCredentials>
```

3.2 Validátor uživatelského jména a hesla

Prostředí WCF se neomezuje pouze na využívání již hotových komponent ověřování, ale umožňuje vývojáři implementovat i vlastní řešení. To umožňuje třída validátor implementovaná v .Net Framework. Pro začlenění vlastního validátoru do Vaší aplikace stačí implementovat vlastní třídu, která bude zděděna od třídy `UserNamePasswordValidator` z .Net Framework. Vaše třída bude přepisovat metodu `Validate` přebírající dva parametry jméno a heslo, oba typu `string`, ve které implementujete vlastní ověřovací mechanizmus. V jazyce C# by mohla třída vlastního validátoru vypadat následovně. [33]

```
public class CustomUserNameValidator : UserNamePasswordValidator
{
      public override void Validate(string userName, string password)
```

```
      {
            if (null == userName || null == password)
            {
                  throw new ArgumentNullException();
            }

            if (!(userName == "uzivatel" && password == "test1")
            && !(userName == "uzivatel2" && password == "test2"))
            {
                  throw new FaultException("Unknown Username or Incorrect
                  Password");
            }
      }
}
```

Pokud implementujete vlastní validátor bude zapotřebí upravit i konfigurační soubor Vaší aplikace, konkrétně upravit element `<system.serviceModel>`. [33]

```
<system.serviceModel>
  <bindings>
  <wsHttpBinding>
      <binding name="MyBinding">
        <security mode="Message">
          <message clientCredentialType="UserName" />
        </security>
      </binding>
    </wsHttpBinding>
  </bindings>
</system.serviceModel>

<serviceCredentials>
  <userNameAuthentication userNamePasswordValidationMode="Custom"
  customUserNamePasswordValidatorType=
  "Microsoft.ServiceModel.Samples.CustomUserNameValidator, service" />
</serviceCredentials>
```

3.3 Autentizace a ověření identity služby

Každá WCF služba je v prostředí internetu nebo počítačové síti přístupná pomocí koncových bodů tzv. endpointů. Každému endpointu je možné nakonfigurovat způsob, jakým se služba vůči klientu ověřuje, takzvanou identitu. Vždy, když klient iniciuje komunikaci s koncovým bodem služby, sám vyhodnotí v ověřovacím procesu pomocí

porovnávací hodnoty, zda se spojuje opravdu s očekávaným endpointem. Toto zabezpečení koncových bodů služby slouží k ochraně proti *phishingu*[8] tím, že zabraňuje k přesměrování klienta na jinou službu. V konfiguračním souboru můžeme nastavit vždy jednu z šesti následujících typů identit:

- Domain Name System (DNS),
- Certificate,
- Certificate Reference,
- RSA,
- User principal name (UPN),
- Service principal name (SPN). [34]

Nastavení např. DNS identity se provádí v konfiguračním souboru. [34]

```
<configuration>
<system.serviceModel>
      <bindings>
            <wsHttpBinding>
                  <binding name="WSHttpBinding_IMyInterface ">
                        <security>
                        <message clientCredentialType="Windows"/>
                        </security>
                  </binding>
            </wsHttpBinding>
      </bindings>
  <client>
      <endpoint address="http://localhost:7777/BusinessService"
            binding="wsHttpBinding"
            bindingConfiguration="WSHttpBinding_IMyInterface "
            contract="IMyInterface"
            name="WSHttpBinding_IMyInterface">
            <identity>
            <dns value="fai.utb.com" />
```

[8] Phishing je podvodná technika používaná na Internetu k získávání citlivých údajů a funguje na principu rozesílání e-mailových zpráv

```
            </identity>
        </endpoint>
    </client>
</system.serviceModel>
</configuration>
```

3.4 Autorizace klienta

Při autorizaci se provádí ověření práv přihlášeného uživatele. Každý přihlášený uživatel může mít v rámci aplikace různě omezená práva v přístupu k operacím, které může provádět. Potom takový uživatel, který má omezená práva na určité úrovni, může např. otevřít soubor pouze pro čtení nebo jen editovat záznam v aplikaci. Proto je nutné vytvořit v systému uživatelské role. Tyto role je možné realizovat několika způsoby, uvedu ty nejběžnější.

3.4.1 Uživatelské role v ASP.NET Role Provider

Role Provider je rozhraní, které nám nabízí společně s *Membership provider* komplexní řešení pro správu uživatelských účtů a rolí. *Membership provider* použijeme pro správu uživatelských účtů a *Role Provider* pro správu uživatelských rolí v systému. Toto rozhraní nám umožňuje snadno vytvářet uživatelské účty, správu hesel, validaci uživatelského jména a hesla atd. Ovšem musíme si uvědomit, že *Membership provider* umožňuje pouze autentifikaci uživatele, nikoli však už jeho autorizaci, tj. zajištění oprávnění ke zdrojům. Na toto nesmíme zapomenout při vytváření naší aplikace. K autorizaci uživatele musíme mít definovány v datovém modelu pro *Role Provider* pro každý uživatelský účet i hodnotu k identifikaci uživatelské role. *Role Provider* i *Membership provider* využívají společnou vlastní databázi k uložení uživatelských účtů. V .NET Framework nabízí vývojáři vestavěnou podporu pro *Membership provider a Role Provider*, která obsahuje dvě implementace *Active directory* a *SqlMembership provider*. Také nám nabízí následující vizuální ovládací prvky, které umí pracovat s *Membership provider a Role Provider*.

- **Login:** ovládací prvek pro zadání uživatelského jména a hesla.
- **LoginView:** jednoduchý prvek, který zobrazuje stav podle toho, zda je uživatel přihlášený nebo aplikaci používá anonymní uživatel.

- **LoginStatus:** poskytuje stav uživatele a nabízí možnosti přihlásit/odhlásit.
- **LoginName:** zobrazuje uživatelské jméno.
- **PasswordRecovery:** umožňuje uživateli vygenerovat a odeslat na emailovou adresu nové přístupové heslo.
- **CreateUserWizard:** tento komplexnější prvek provede založení nového uživatele. Je možné jej parametrizovat podle vlastní potřeby.
- **ChangePassword:** zajistí změnu hesla uživatele. [35]

3.5 Uživatelské účty a role ve vlastním datovém úložišti

V tomto případě se uživatelské účty ukládají stejně jako v předchozím případě do databáze, ovšem struktura a implementace přístupu i správy účtů je v plné režii programátora. Každý vývojář si sám vytvoří mechanizmus, kterým bude přistupovat k uživatelským účtům a jak budou uloženy. Nejjednodušší způsob je jedna tabulka v databázi pro uživatelské účty i role. Programátor musí sám vymyslet a implementovat metody a funkce pro manipulaci s uživatelskými účty. Tento vlastní způsob je časově náročnější než předchozí použití již hotového prostředí *Membership provider*, ale na druhou stranu má vývojář nad vlastním kódem plnou kontrolu.

3.6 Nastavení pravidel přístupu metodám WCF služby

Vytvořením uživatelských účtů a nastavení uživatelských rolí jsme splnili první podmínku k autorizaci klienta a uživatele. Dále je nutné zajistit, aby k veřejným metodám rozhraní služby měli přístup pouze oprávnění uživatelé. Před každou metodu rozhraní přidáme atribut s parametrem role, které mohou tuto metody volat viz. následující zdrojový kód v jazyce C#.

```
[PrincipalPermission(SecurityAction.Demand, Role = "ucetni")]
public double GetPrumernaMzdaZamestnance(int zamestnanecId)
{...}
[PrincipalPermission(SecurityAction.Demand, Role = "manager")]
public double GetPrumernyZiskZaObdobi(DateTime datumOd, DateTime datumDo)
{...}
```

3.7 Zabezpečení citlivých dat

Pokud budeme ukládat uživatelské účty, půjde o soukromé informace o uživateli, které by měly zůstat utajeny, jako je například heslo, informace o platebních kartách atd., je nutné, aby i tato data byla v databázi zabezpečena proti přímému čtení nepovolanými subjekty. Z hlediska zpětného získání informace rozlišujeme metody zabezpečení na vratné a jednocestné.

3.7.1 Jednocestná metoda zabezpečení

Typickými daty, pro která se využívá jednocestná metoda zabezpečení, jsou uživatelská hesla, protože uložené heslo v textové podobě je snadno prolomitelné. Z toho důvodu je vhodné heslo uživatele šifrovat. Případný útočník nebo správce databáze nikdy původní heslo neuvidí.

Jednocestné zabezpečení se provádí pomocí hashovacích algoritmů. Vstupem hashovací funkce je libovolný textový řetězec a jejím výstupem je řetězec pevné délky tzv. hash. K dispozici máme několik hashovacích algoritmů jako např. MD2, MD4, MD5, RIPEMD, SHA-0, SHA-1, SHA-256/224, SHA-512/384. [7]

Hashovací algoritmy musí splňovat následující požadavky:

- pro dva různé vstupní řetězce by neměl být vytvořen stejný hash,
- vlastnost algoritmu, kdy malá změna na vstupu způsobí radikální změnu na výstupu.

U hashovacích funkcí prakticky neexistuje jiná možnost prolomení výsledného hashe než použití tzv. hrubé síly (*brute force*), což znamená postupně generovat všechna možná hesla a jejich hash porovnávat s tím, který chceme prolomit. Největším rizikem bývají krátká hesla, protože obsahují jen malou množinu znaků a není velký problém jej odhalit.

3.7.2 Vratná metoda zabezpečení

Zatímco u jednocestné metody jsme po zašifrování vstupního řetězce měli k dispozici pouze výsledný hash pevné délky, u vratné metody nám jde o to, neztratit možnost získat zpět šifrovaný řetězec do původní formy. Tato metoda se využívá například u kreditních karet a většinou využívá asymetrické šifry.

4 AUTENTIZACE SLUŽBY

Při komunikaci mezi službou a klientem je důležité, aby se ověřila i samotná služba vůči klientu. I klient musí mít jistotu, že komunikuje v prostředí internetu opravdu s tou službou, za kterou se služba vydává. Pokud se služba prokáže platným ověřovacím certifikátem, který vydala pravá certifikační autorita, probíhá předávání informací mezi klientem a službou (serverem) prostřednictvím zabezpečeného protokolu. Toto nám zajišťuje takzvaná vrstva bezpečných socketů *Secure Sockets Layer* (SSL).

4.1 Vrstva SSL

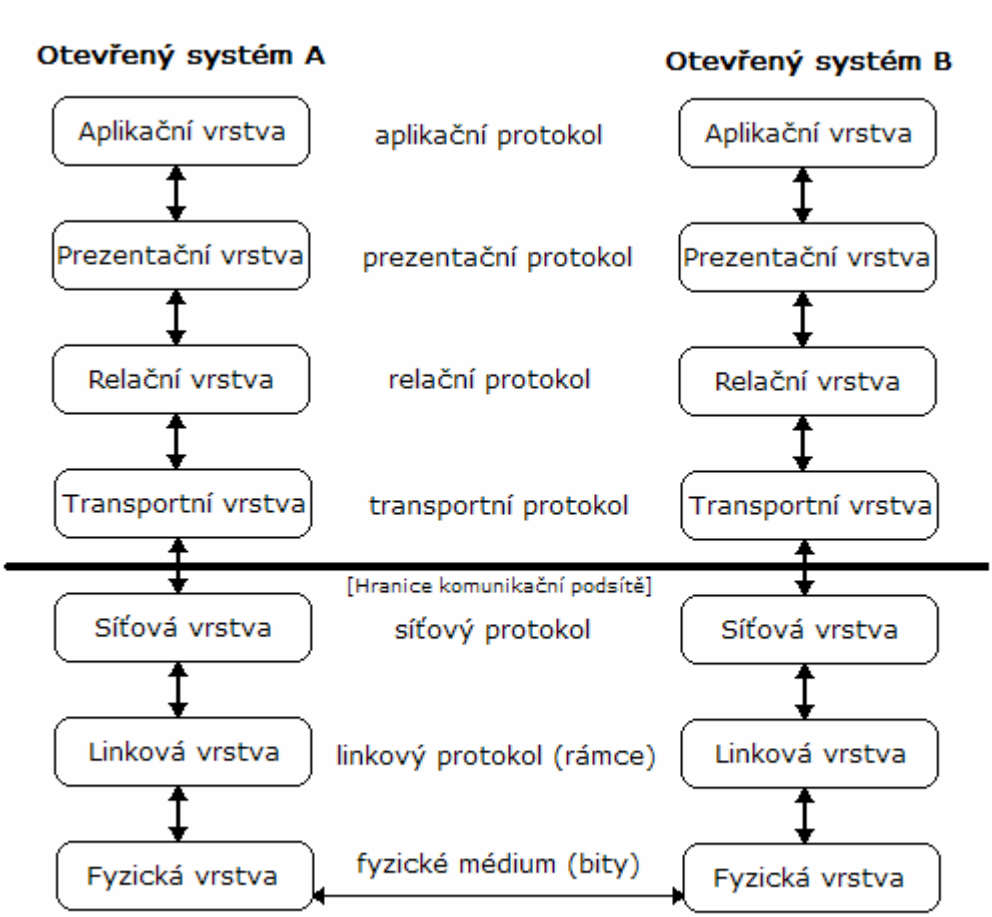
Je to vrstva vložená mezi transportní a aplikační vrstvu referenčního modelu OSI (*Open System Interconnection*), která zajišťuje zabezpečení komunikace šifrováním a autentizaci komunikujících stran. Spojení zabezpečené pomocí SSL funguje na principu asymetrické šifry, kde každá z komunikujících stran má dvojici šifrovacích klíčů – veřejný a soukromý. Příjemce zpráv zveřejní svůj veřejný klíč, kterým odesílatel zašifruje zprávu pro příjemce. Potom je zajištěno, že příjemce svým soukromým klíčem bude moci přijatou zprávu rozšifrovat. Průběh komunikace mezi serverovou službou a klientem zabezpečenou spojením SSL probíhá v následujících krocích.

- Klient pošle serveru požadavek na SSL spojení.
- Server pošle klientovi odpověď na jeho požadavek, která obsahuje stejný typ informací a hlavně digitální certifikát serveru.
- Podle přijatého certifikátu si klient ověří autentičnost serveru. Certifikát také obsahuje veřejný klíč serveru.
- Na základě dosud obdržených informací vygeneruje klient základ šifrovacího klíče, kterým se bude šifrovat následná komunikace. Ten zašifruje veřejným klíčem serveru a pošle mu ho.
- Server použije svůj soukromý klíč k rozšifrování základu šifrovacího klíče. Z tohoto základu vygenerují jak server, tak klient hlavní šifrovací klíč.
- Klient a server si navzájem potvrdí, že od teď bude jejich komunikace šifrovaná tímto klíčem.

Nyní je zabezpečené spojení šifrované vygenerovaným šifrovacím klíčem a aplikace nadále komunikuje se serverem šifrovaným spojením. Vrstva SSL se nejčastěji využívá pro bezpečnou komunikaci v internetu pomocí HTTPS protokolu, což je zabezpečená verze komunikačního protokolu HTTP. [22], ([4], kap. 19)

4.1.1 Referenční model OSI

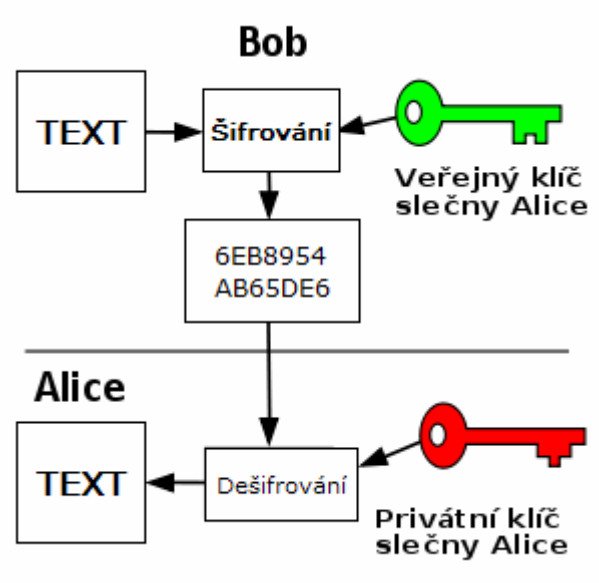
Tato architektura byla vytvořená v roce 1979 organizací ISO (*Industrial Standard Organization*). Tato charakteristika rozděluje uzly systému na koncové uzly komunikující po síti. Komunikace je rozdělena do sedmi komunikačních vrstev, jak je znázorněno na obrázku (Obr. 11). Síťová komunikace probíhá přesně definovaným způsobem mezi sousedními vrstvami prostřednictvím služebních entit (vrstev). [4], ([8], kap. 1.4.2)



Obr. 11: Referenční model OSI (převzato z [8], s. 16)

4.1.2 Asymetrická šifra

Tato šifra se vyznačuje nutností dvou šifrovacích klíčů tzv. veřejného a soukromého, tedy pro zašifrování a dešifrování zprávy se používají rozdílné klíče viz. obrázek (Obr. 12).



Obr. 12: Asymetrické šifrování a dešifrování (převzato z [19])

Výhodou asymetrické šifry je absence nutnosti výměny klíčů, protože ten, kdo šifruje nějakou zprávu, má k dispozici pouze veřejný klíč příjemce. Příjemce si zprávu dešifruje pouze za pomocí svého soukromého klíče. Je zřejmé, že šifrovací klíč a dešifrovací klíč spolu musí být matematicky svázány, avšak nezbytnou podmínkou pro užitečnost šifry je praktická nemožnost ze znalosti šifrovacího klíče spočítat dešifrovací. Matematicky je tedy možné postupovat následujícím způsobem.

Šifrování $c = f(m,e)$, kde m je šifrovaná zpráva a e je šifrovací klíč.

Dešifrování $m = g(c,d)$, kde c je zašifrovaná zpráva a d je dešifrovací klíč.

Šifrovací i dešifrovací funkce se mohou lišit, ale matematicky si jsou vždy velmi podobné. Asymetrické šifrování je založeno na tzv. jednocestných funkcích nebo-li funkcích snadno proveditelných v jednom směru, například ze vstupu funkce je snadné vypočítat výstup, ale opačně nalézt z výstupu vstupní hodnoty je velmi obtížné. [19]

4.1.3 Protokol HTTPS

Zkratka HTTPS je z anglického názvu *Hypertext Transfer Protocol Security* což je zabezpečená verze protokolu HTTP. Protokol HTTP je určený pro výměnu hypertextových dokumentů ve formátu HTML prostřednictvím internetu a využívá obvykle port TCP/80. HTTPS je nadstavbou tohoto síťového protokolu, která umožňuje komunikaci zabezpečit před odposloucháváním, podvržením dat a umožňuje též ověřit identitu protistrany. HTTPS používá protokol HTTP, přičemž přenášená data jsou šifrována pomocí SSL vrstvou a standardní port na straně serveru je 443. Princip fungování protokolu využívá asymetrické šifrování, kdy si před zahájením komunikace obě strany vygenerují vlastní privátní a veřejný klíč a veřejné klíče si vymění. Každá z komunikujících stran si přijatý veřejný klíč ověří jiným komunikačním kanálem nebo přenesením důvěry z certifikační autority, která veřejný klíč podepsala. Pokud je certifikační autorita, která přijatý klíč od protistrany podepsala v úložišti důvěryhodných autorit, můžeme i klíč považovat za důvěryhodný. Pokud se oběma stranám podaří ověřit veřejné klíče probíhá nadále komunikace šifrovaně, tedy každá z komunikujících stran před odesláním zprávu zašifruje obdrženým veřejným klíčem protistrany a po přijetí zašifrované zprávy od druhé strany zprávu dešifruje vlastním privátním klíčem viz. obrázek (Obr. 12). [20]

4.2 Certifikační autorita

Certifikační autorita (zkráceně CA) je subjekt, který je oprávněn vydávat digitální certifikáty tak, že svojí autoritou potvrzuje pravdivost údajů uvedených v certifikátu a ostatním subjektům tak umožňuje tento certifikát považovat za důvěryhodný. Jinak řečeno důvěřujeme-li samotné certifikační autoritě, můžeme považovat certifikáty vydané touto autoritou za důvěryhodné.

4.2.1 Činnost certifikační autority

Certifikační autorita vydává digitální certifikáty (elektronicky podepsané veřejné klíče), u kterých se zaručuje za správnost identifikačních informací o majiteli certifikátu, které jsou v něm uvedené. Pokud CA podepíše vydaný certifikát vlastním klíčem, jedná se o certifikát podepsaný sám sebou (také známý pod anglickým názvem *self-signed certificate*). Žadatel o vydání digitálního certifikátu důvěryhodným způsobem musí

certifikační autoritu přesvědčit, že jeho poskytnuté údaje uvedené ve veřejném klíči odpovídají skutečnosti. Fyzická osoba tuto skutečnost prokáže předložením občanského průkazu zástupci certifikační autority, u právnické osoby postačí předložení výpisu z obchodního rejstříku. Důležitou součástí digitálního certifikátu je elektronický podpis, pomocí kterého lze snadno ověřit jeho autentičnost. Pokud by došlo k neoprávněné změně údajů v certifikátu, při ověřování elektronického podpisu by došlo k odhalení těchto změn. [10]

4.2.2 Důvěra v certifikační autoritu

Pro certifikační autoritu je velmi důležitá její míra důvěryhodnosti vůči svému okolí, protože od toho se odvíjí i důvěryhodnost certifikátů, které vydává a samozřejmě i jejich cena. O míře důvěryhodnosti CA se můžeme dozvědět např. z jejich webových stránek, diskusích na internetu, tisku, médií apod. Důvěru v certifikační autoritu můžeme logicky přenášet i na všechny certifikáty, které vydala. Pokud víme o dané CA, že patří mezi důvěryhodné, můžeme při ověřování autentičnosti veřejného klíče, který je touto autoritou podepsaný, přenést důvěru v CA na něj a přesunout ho do úložiště důvěryhodných certifikátů. [10]

4.2.3 Kvalifikovaná certifikační autorita

Kvalifikovaná certifikační autorita je rámci České republiky definována *Zákonem o elektronickém podpisu* (zákon č. 227/2000 Sb.). Seznam akreditovaných certifikačních autorit, které mohou vydávat kvalifikované certifikáty, zveřejňuje Ministerstvo vnitra České republiky. Kvalifikovaná akreditovaná certifikační autorita vydává zpoplatněné kvalifikované certifikáty, což jsou standardní digitální certifikáty, které jsou výše zmíněným zákonem uznávány v rámci komunikace se státními institucemi České republiky. Kvalifikovaný certifikát je ze zákona akceptován stejně jako třeba občanský průkaz. [10]

4.3 Digitální certifikát

V podstatě jde o digitálně podepsaný veřejný šifrovací klíč, který je vydán certifikační autoritou a uchovává se ve formátu kryptografického standardu *X.509*[9], který obsahuje informace o majiteli veřejného klíče a vydavateli certifikátu viz. obrázek (Obr. 13). Certifikát musí obsahovat tyto položky:

- sériové číslo certifikátu,
- identifikační údaje majitele certifikátu,
- algoritmus použitý k vytvoření podpisu,
- identifikační údaje vydavatele certifikátu,
- datum počátku platnosti certifikátu,
- datum konce platnosti certifikátu,
- účel veřejného klíče a jeho bitová délka,
- algoritmus hash certifikátu, který zaručuje neporušení certifikátu. [11]

Digitální certifikát je vydáván certifikační autoritou, která nejprve ověří údaje o majiteli certifikátu a na základě těchto údajů vygeneruje soukromý a veřejný klíč a vytvoří certifikát, který obsahuje ověřené informace. Certifikační autoritou může být soukromá nebo státní instituce. [11]

4.3.1 Typy certifikátů

Všechny certifikační autority vydávají vlastní druhy certifikátů, které se liší podle účelu jejich použití, typu majitele, úrovni důvěryhodnosti atd. Tyto podmínky jsou dány bezpečnostní politikou konkrétní autority. Dále pak existují kvalifikované certifikáty, které mohou vydávat pouze akreditované certifikační autority, které jsou definovány příslušným Zákonem o elektronickém podpisu. Dalším typem certifikátu je tzv. *Self-signed* certifikát

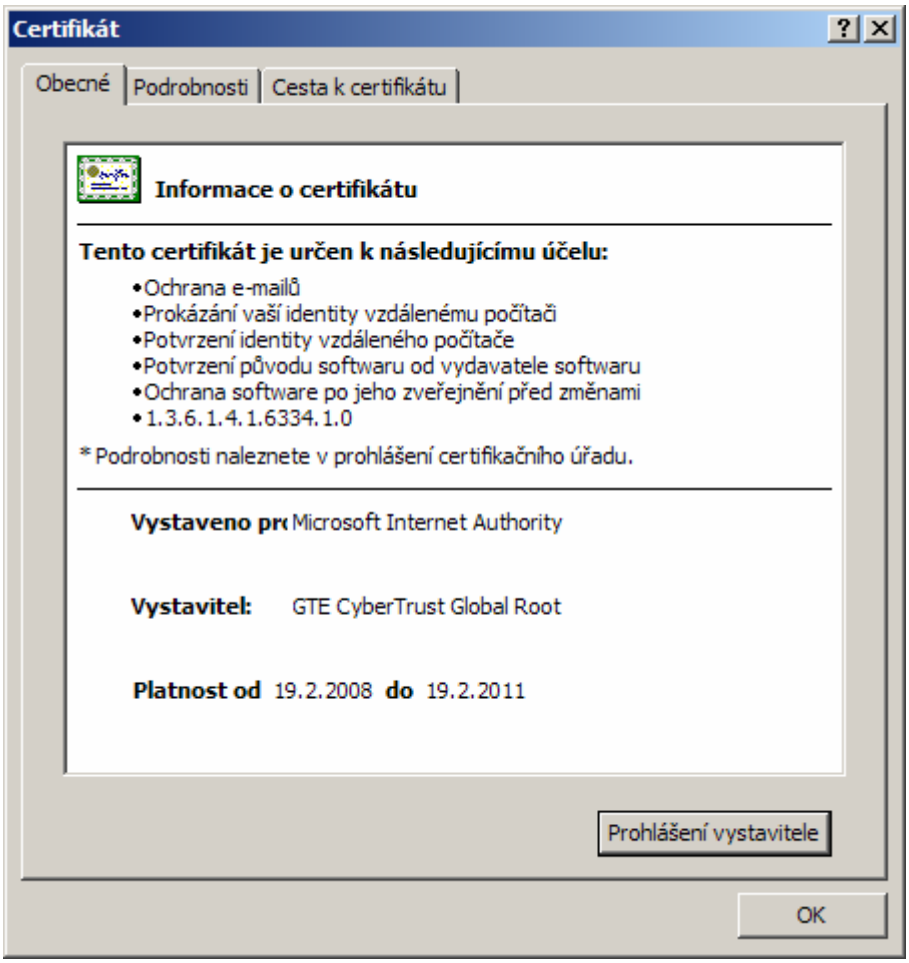
[9] X.509 je standard pro systémy založené na veřejném klíči k jednoduchému podepisování.

(certifikát podepsaný sám sebou). Jedná se o certifikát, který podepíše sám jeho tvůrce. To znamená, že osoba, která ho vytvořila, se podepsáním zaručuje za jeho platnost.

4.3.2 Kvalifikovaný certifikát

Je to standardní certifikát uznávaný podle výše zmíněného Zákona o elektronickém podpisu pro komunikaci se státními institucemi České republiky. Kvalifikovaný certifikát je ze zákona akceptován stejně jako např. občanský průkaz, ovšem jeho využití je omezeno pouze na některé případy:

- komunikace se státní správou pomocí emailu,
- ověřování elektronických podpisů,
- zajištění neodmítnutelnosti odpovědnosti. [11]



Obr. 13: Příklad digitálního certifikátu

4.3.3 Certifikát podepsaný sám sebou

Tento certifikát (známý též pod anglickým názvem *Self-signed certificate*) je podepsaný jeho tvůrcem. To znamená, že osoba, která tento certifikát vytvořila, se také podepsáním zaručuje za jeho platnost. V uspořádání infrastruktury veřejných klíčů je platnost jednotlivých certifikátů doložena digitálním podpisem. Klienti kontrolují, jestli soukromý klíč použitý k podepsání určitého certifikátu odpovídá veřejnému klíči na certifikátu. Tento typ certifikátu je možné použít také s omezení jen pro některé činnosti:

- podepisování vlastních softwarových produktů,
- podepisování serverů, aplikací, osob atd.,
- k bezpečné komunikaci např. mezi webovým prohlížečem a serverem.

Tento certifikát si může každý vytvořit sám na svém počítači a má nejnižší úroveň zabezpečení a proto je vhodné ho používat např. pro testovací účely. [11]

4.3.4 Platnost certifikátu

Každý certifikát má omezenou platnost, která je závislá na typu certifikátu. Většinou jsou běžné certifikáty vydávány na jeden rok. Platnost je omezena hlavně z důvodu bezpečnosti, aby nemohlo dojít ke zneužití podpisu, protože ke každému konkrétnímu podpisu existuje více alternativních zpráv, které by s použitím stejného certifikátu prošly ověřením podpisu a mohly by se vydávat za námi podepsané dokumenty. Nalezení alternativní zprávy by trvalo i tisíce let, ovšem v dnešní době za použití moderních výpočetních prostředků by se tato doba dala výrazně zkrátit. Aby k tomuto nemohlo dojít, je nutné neustále zlepšovat kryptografické techniky, algoritmy a délky klíčů a právě proto musí mít certifikáty omezenou platnost. [11]

4.3.5 Vytvoření Self-signed certificate ve Windows

V prostředí operačního systému Windows je možné pro vytvoření certifikátu využít utilitu *makecert.exe*, kterou systém nabízí. Použitím této utility s parametry vygenerujeme

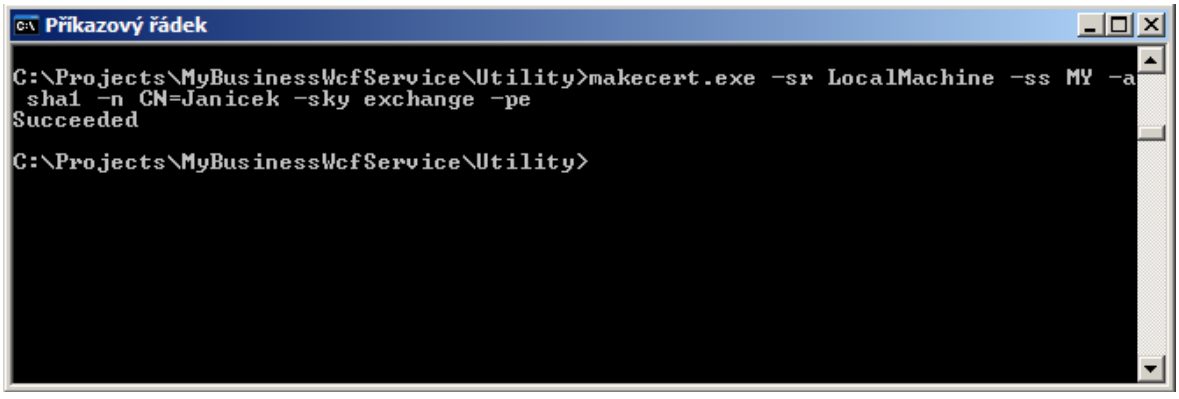
vlastní certifikát podle standardu *X.509*[10] pro testovací účely, který bude obsahovat veřejný a privátní klíč. Utilita *makecert.exe* obsahuje parametry, kterými lze výsledný výstup ovlivnit, následně vysvětlím ty nejčastěji používané:

- **-n** určuje jméno subjektu,
- **-sr** specifikuje umístění certifikátu v úložišti,
- **-ss** udává předmět využití certifikátu,
- **-a** šifrovací algoritmus,
- **-sky** popisuje typ klíče,
- **-pe** způsob jakým se má předávat veřejný klíč.

Více o parametrech této utility se dočtete v [9]. Pokud tedy chcete vytvořit vlastní certifikát spusťte v operačním systému Windows příkazovou řádku a zapište do ní následující příkaz a potvrďte.

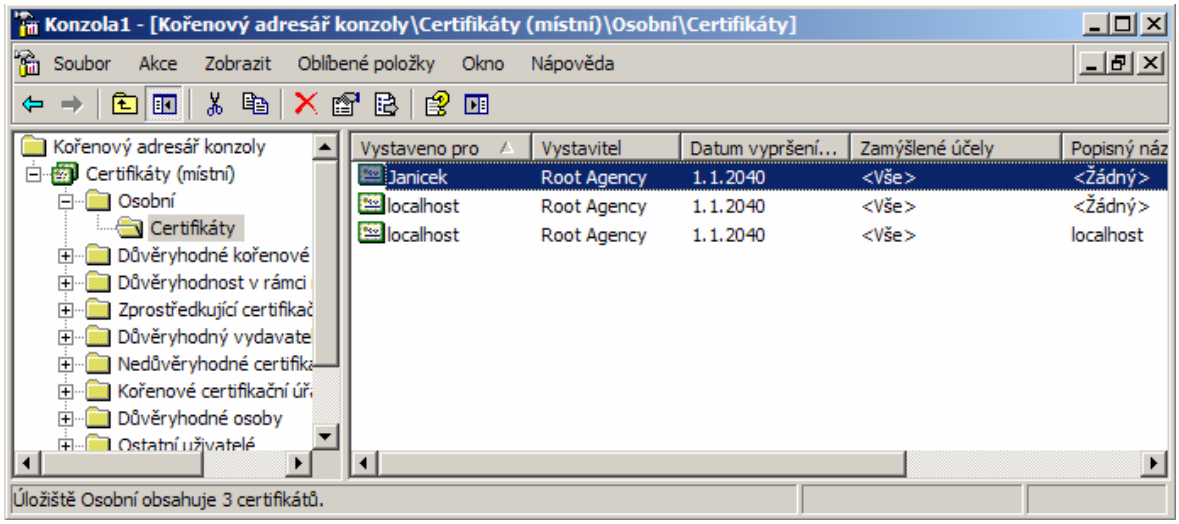
```
makecert.exe -sr LocalMachine -ss MY -a sha1 -n CN=Janicek -sky exchange
-pe
```

Následující výpis v příkazové řádce by měl vypadat podobně jako na obrázku (Obr. 14). Tímto příkazem vytvoříte vlastní certifikát, který naleznete v úložišti osobních certifikátů viz. obrázek (Obr. 15) a můžeme jej použít.



Obr. 14: Vytvoření certifikátu utilitou makecert.exe v příkazové řádce

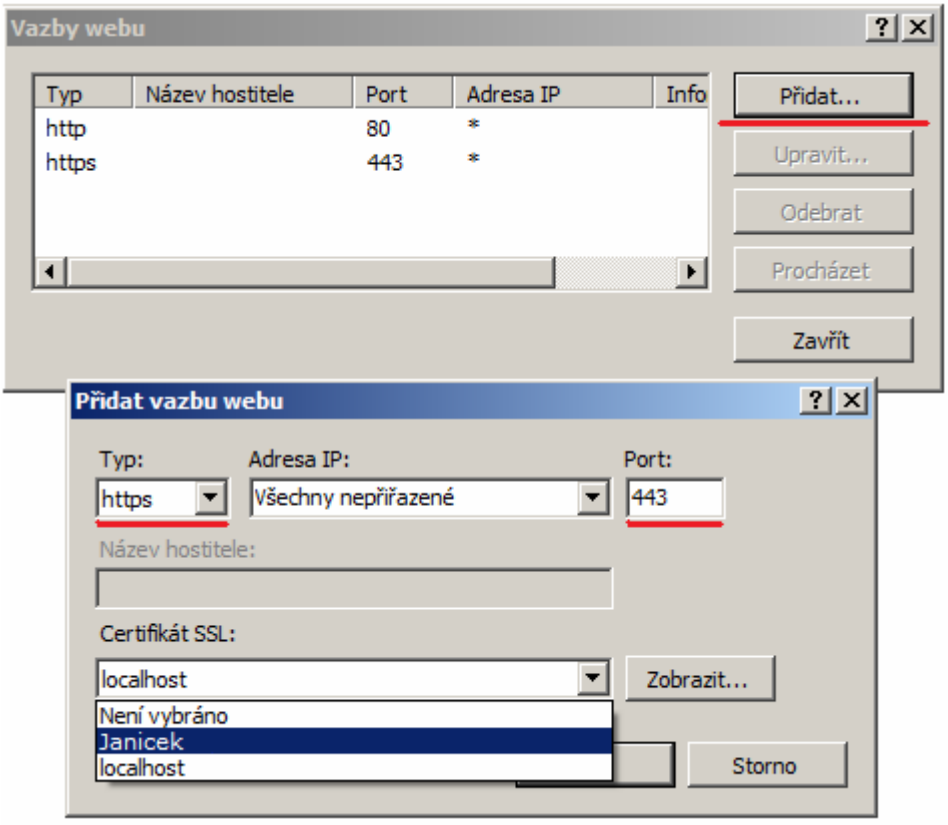
[10] X.509 je standard pro systémy založené na veřejném klíči k jednoduchému podepisování.



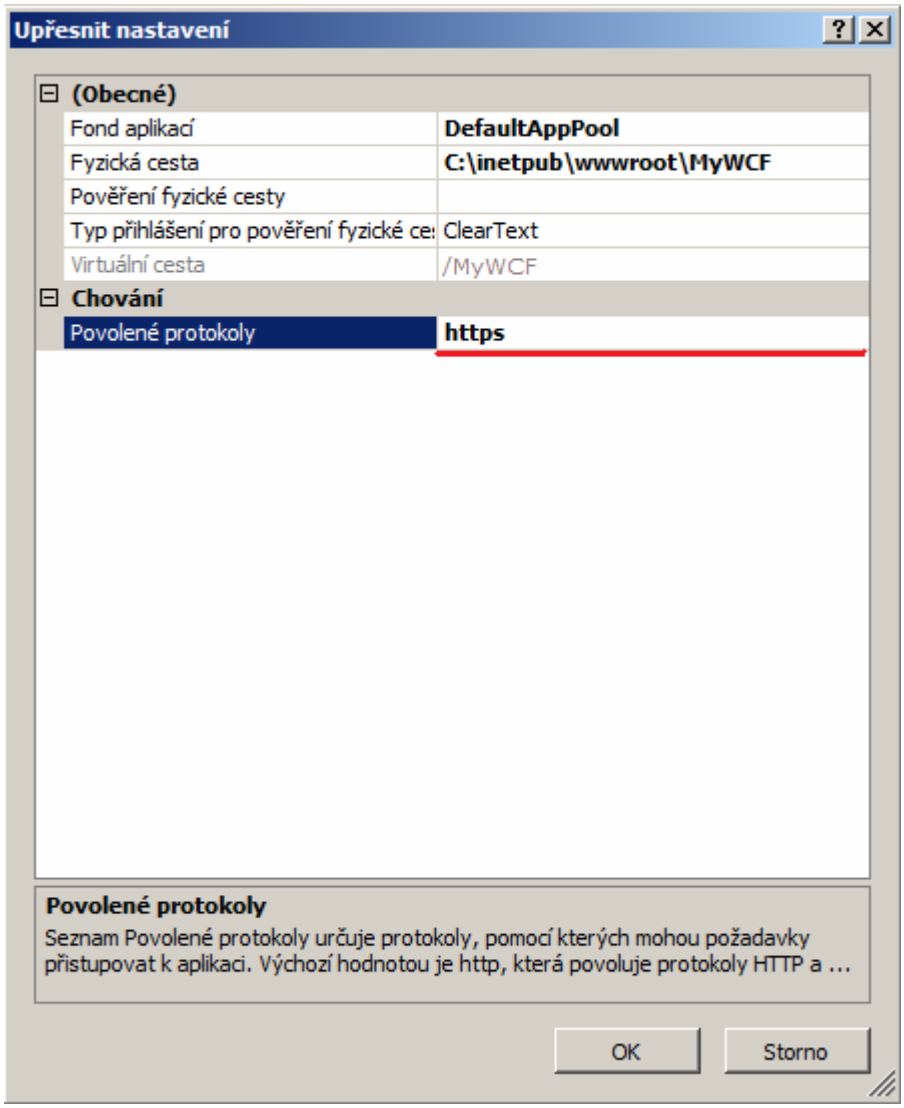
Obr. 15: Zobrazení vytvořeného osobního certifikátu

4.4 Zabezpečení WCF služby hostované v IIS 7

Internetová informační služba verze 7 (IIS7) operačního systému Windows je způsob jak můžeme WCF službu zpřístupnit přes HTTP nebo *net.tcp* protokol. Pokud máme WCF službu hostovanou v IIS7, bude pro její zabezpečené provozování potřeba, aby komunikace s ní probíhala prostřednictvím protokolu HTTPS. To provedeme snadno nástrojem IIS Manager tak, že navážeme na webovou aplikaci certifikát a budeme vyžadovat, aby komunikace na této adrese probíhala přes HTTPS protokol a nastavíme port 443, jak je znázorněno na obrázcích (Obr. 16 a 17). Pokud budeme mít certifikát úspěšně svázaný s webovou aplikací, kde je hostována naše WCF služba, musíme ještě provést úpravu její konfigurace.



Obr. 16: Navázání certifikátu na webovou aplikaci



Obr. 17: Nastavení HTTPS protokolu

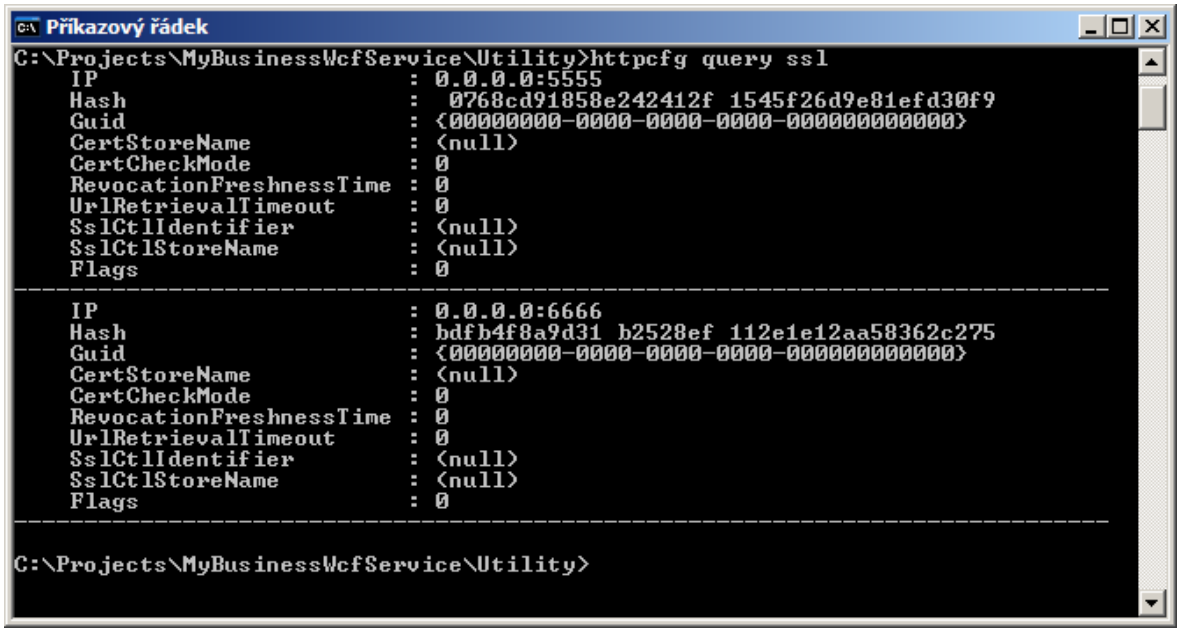
4.5 Zabezpečení WCF služby hostované ve Windows Service

Službu WCF můžeme hostovat různými způsoby a další možností je hostování jako Windows službu (*Windows service*). I v tomto případě budeme vyžadovat, aby komunikace klienta a služby probíhala zabezpečeně a nemohlo dojít k úniku citlivých informací. Při hostování ve Windows službě je postup zabezpečení o trochu složitější než tomu bylo při hostování v IIS7. Nejprve budeme muset registrovat IP adresu, kterou rezervujeme pro naši službu a určíme komunikační port. Potom na port adresy navážeme certifikát.

4.5.1 Registrování IP adresy a navázaní certifikátu na komunikační port

Pro registrování IP v operačním systému Windows použijeme utilitu systému *httpcfg.exe*. Tato utilita nám umožňuje spravovat IP adresy registrované v počítači. Nejprve než registrujeme novou IP adresu je nezbytné pomocí utility vypsat všechny již obsazené adresy, aby nedošlo ke konfliktu. To provedeme z příkazové řádky následujícím příkazem a dostaneme výpis podobný tomu na obrázku (Obr. 18).

```
httpcfg query ssl
```



Obr. 18: Výpis všech registrovaných URL adres v počítači

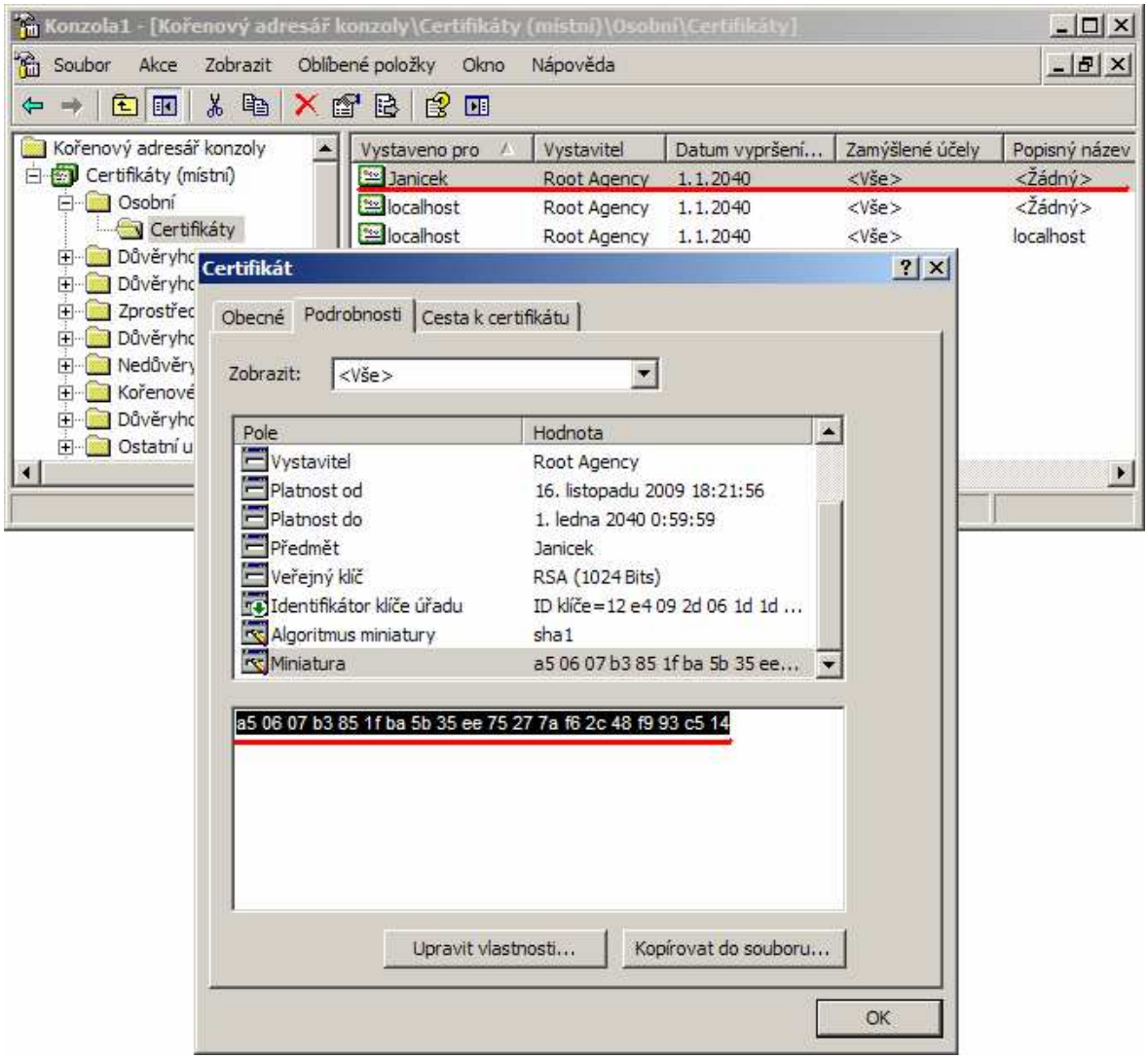
Nyní zvolíme novou ještě nepoužitou adresu např. `0.0.0.0:7777` a registrujeme ji. Samé nuly v IP adrese znamenají, že se má použít výchozí adresa lokálního počítače (*localhost*). Při registraci adresy rovnou provedeme i provázání certifikátu s portem adresy, v tomto případě půjde o port `7777`. Nejprve budeme potřebovat zjistit hash (tzv. miniatura) našeho certifikátu, který najdeme v podrobnostech certifikátu viz. obrázek (Obr. 19). Miniaturu zkopírujeme a v libovolném textovém editoru odstraníme všechny mezery tak, abychom měli souvislý textový řetězec. Utilita *httpcfg* obsahuje velké množství nepovinných vstupních parametrů, o kterých se dočtete v [21]. Nám budou stačit pro naše účely pouze dva z nich:

- **-i** tento parametr říká, že námi zvolená adresa bude doplněna o port,
- **-h** specifikuje miniaturu certifikátu, který chceme na port navázat.

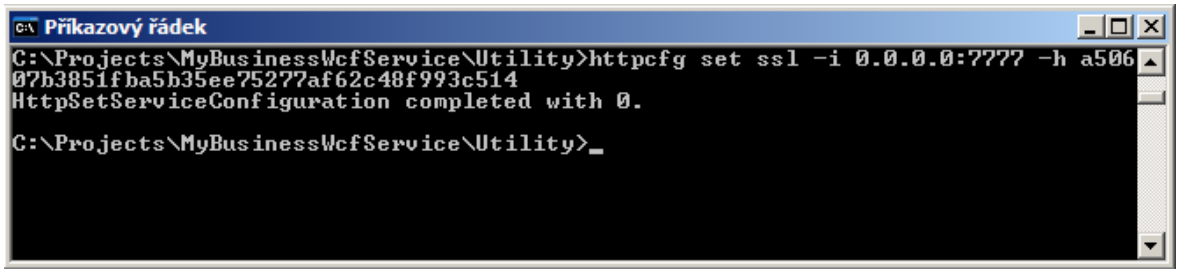
Pro registrování nové adresy použijeme následující příkaz z příkazové řádky:

```
httpcfg set ssl -i 0.0.0.0:7777 –h
a50607b3851fba5b35ee75277af62c48f993c514
```

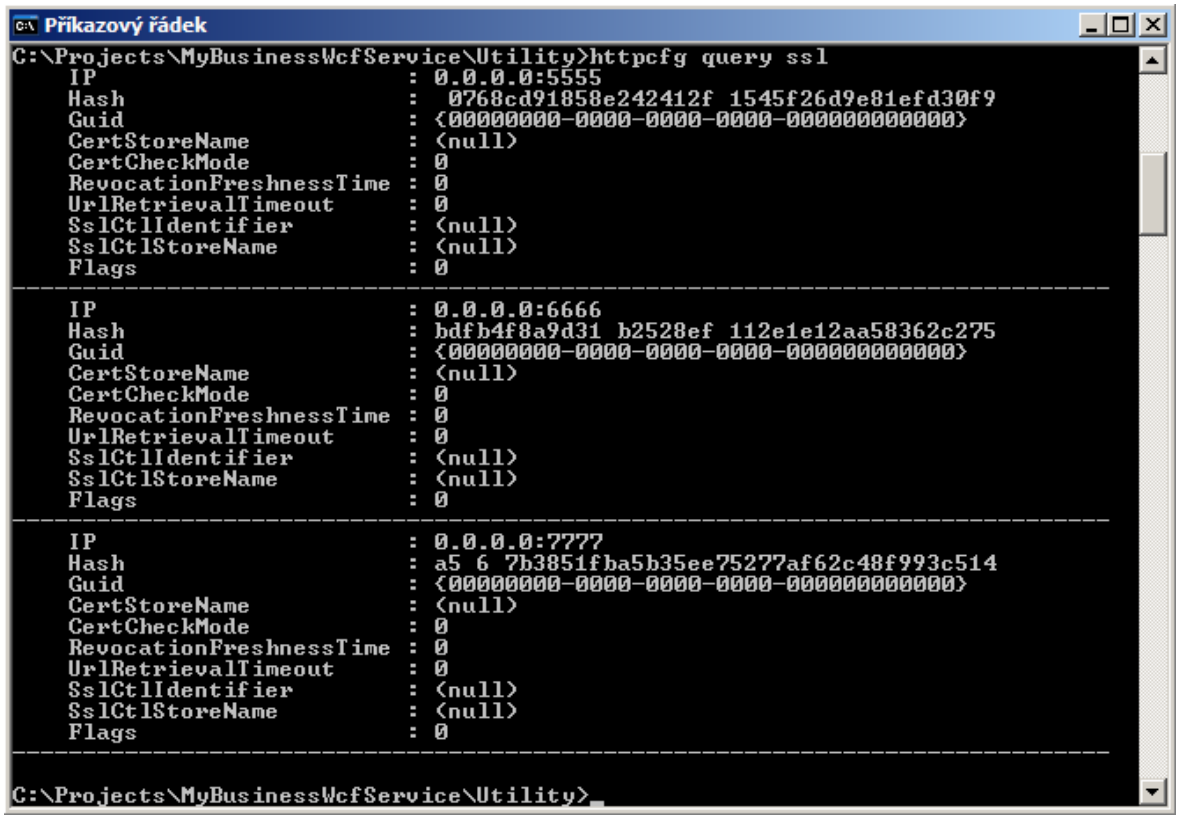
Po provedení příkazu by měl výpis v konzole vypadat jako na obrázku (Obr. 20). Všimněte si, že v příkazu registrování IP adresy je použit přímo řetězec miniatury našeho certifikátu. Ještě zbývá pro kontrolu provést vypsání všech registrovaných adres opět příkazem `httpcfg query ssl`. Pokud registrace proběhla úspěšně, měl by konečný výpis konzoly obsahovat i naši registrovanou adresu `0.0.0.0:7777`, včetně hash certifikátu podobně jako na obrázku (Obr. 21).



Obr. 19: Zjištění miniatury (hash) certifikátu



Obr. 20: Registrace IP adresy a svázání portu s certifikátem



Obr. 21: Konečný výpis všech registrovaných IP adres

4.5.2 Registrace URL adresy pro WCF službu

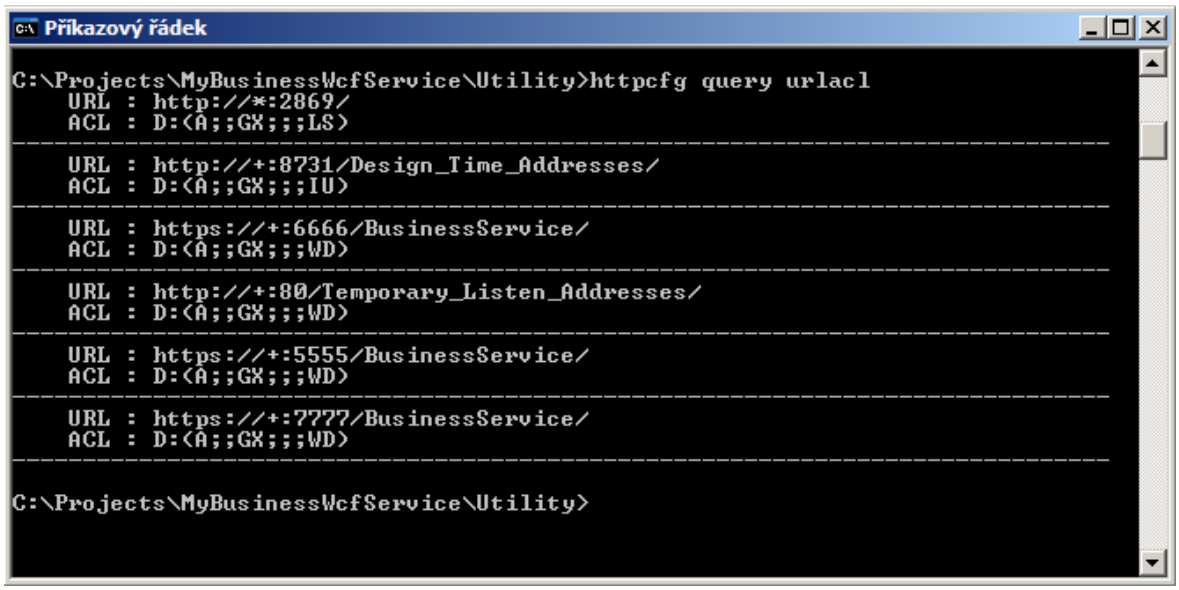
Jakmile jsme zaregistrovali IP adresu je potřeba ještě zaregistrovat URL adresu, abychom naši službu mohli využívat v počítačové síti pomocí HTTP protokolu. Pro registraci využijeme opět utilitu *httpcfg.exe*. Do příkazové řádky zapíšeme následující příkaz a potvrdíme:

```
httpcfg set urlacl /u "https://+:7777/BusinessService" /a  D:(A;;GX;;;WD)
```

V řetězci adresy jsme použili HTTPS, aby komunikace probíhala přes tento zabezpečený protokol. Znak `+` v adrese je zkrácený výraz pro adresu lokálního počítače (*locahost*)

a použili jsme stejný port jako při registraci IP adresy, protože tento port máme svázaný s certifikátem. Nyní ještě zbývá vypsat pro kontrolu všechny registrované URL adresy příkazem a přesvědčit se, že registrace proběhla úspěšně viz. obrázek (Obr. 22).

```
httpcfg query urlacl
```



Obr. 22: Výpis všech registrovaných URL adres

4.6 Konfigurace WCF pro podporu zabezpečeného přenosu

Pokud máme WCF službu hostovánu v IIS7 nebo jako Windows službu a použili jsme certifikát a protokol HTTPS pro komunikaci s ní, nebude nám ještě vše fungovat správně. Nyní je nutné provést konfiguraci samotné služby i klienta, aby podporovali SSL. Konfigurace se musí provést v konfiguračním souboru. Pokud je WCF služba hostována v IIS7 jako webová aplikace, půjde o soubor `web.config`, pokud jako Windows služba je konfiguračním souborem `app.config`.

4.6.1 Konfigurace WCF služby

Konfigurační soubor je psán v jazyku XML, takže je snadné jej upravovat. V konfiguračním souboru služby musíme upravit nastavení v elementu `<system.serviceModel>`. Konkrétně půjde o nastavení módu zabezpečení a nastavení adresy koncových bodů. Nastavení v konfiguračním souboru by mohlo vypadat obdobně, jako je názorně uvedeno v následujícím kódu.

```
<system.serviceModel>
    <services>
        <service behaviorConfiguration="Namespace.MyServiceBehavior"
        name="Namespace.MyService">

        <endpoint address=https://localhost:7777/BusinessService.svc
        binding="basicHttpBinding" bindingConfiguration="Binding"
        contract="Namespace.IMyService"/>

      <host>
        <baseAddresses>
         <add baseAddress="https://localhost:7777/BusinessService.svc"/>
        </baseAddresses>
      </host>
    </service>
</services>

<behaviors>
    <serviceBehaviors>
        <behavior name="Namespace.MyServiceBehavior">
        <serviceMetadata httpsGetEnabled="true"
        httpsGetUrl="https://localhost:7777/BusinessService.svc" />
        <serviceDebug includeExceptionDetailInFaults="true" />
    </behavior>
</behaviors>

<bindings>
    <basicHttpBinding>
        <binding name="Binding">
            <security mode="Transport">
                 <transport clientCredentialType="None"/>
            </security>
        </binding>
    </basicHttpBinding>
</bindings>
</system.serviceModel>
```

4.6.2 Konfigurace klienta

Stejně jako službu je potřeba mít správně nakonfigurovanou klientskou část systému, aby podporovala zabezpečenou komunikaci. Konfiguraci klienta provedeme obdobně, jak tomu bylo u služby v jejím konfiguračním souboru. Nejdůležitější bude nastavit URL

adresu koncových bodů a úroveň zabezpečení. Konfiguraci klienta nám demonstruje následující příklad.

```
<system.serviceModel>
    <client>
        <endpoint
        address="https://localhost:7777/BusinessService/IBusinessService"
        binding="basicHttpBinding" bindingConfiguration="ServiceBinding"
        contract="WcfBusinessService.IService1" name="BusinessService"/>
    </client>
    <bindings>
        <basicHttpBinding>
             <binding name="ServiceBinding">
                  <security mode="Transport"/>
                    </binding>
        </basicHttpBinding>
    </bindings>
</system.serviceModel>
```

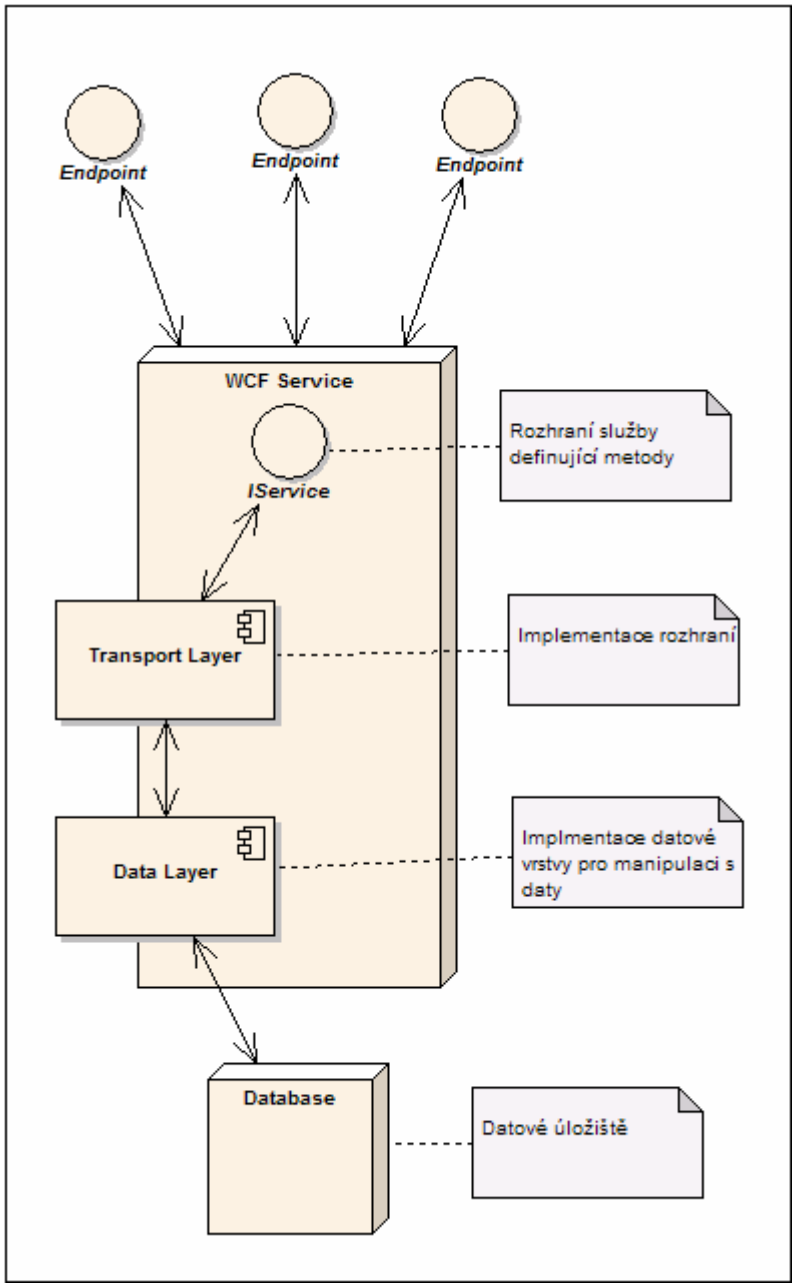
II. PRAKTICKÁ ČÁST

5 ARCHITEKTURA APLIKACE A TECHNOLOGICKÉ ŘEŠENÍ

Vytvořil jsem jednoduchou distribuovanou aplikaci typu klient/server v prostředí .NET Framework implementovánu v jazyce C#. Tuto aplikaci jsem na základě výše uvedených poznatků zabezpečil z hlediska komunikace mezi klientskou a serverovou částí. Komunikaci jsem zabezpečil vrstvou SSL, která nám umožňuje šifrování a autentizaci komunikujících stran ověřujících se certifikátem typu self-signed. Klient je autentifikován svým uživatelským jménem a heslem a autorizován rolí uživatele v systému. Tato distribuovaná aplikace je rozdělena do dvou aplikačních částí. První je serverová část, kterou tvoří WCF služba hostovaná ve Windows service. Druhou částí je klientská aplikace, která je samostatným spustitelným souborem.

5.1 Architektura WCF služby

Službu, která běží v prostředí operačního systému jako Windows process (Windows service) jsem navrhl složenou ze dvou samostatných vrstev. První vrstva je tzv. transportní vrstva a tvoří ji samotná WCF služba svým rozhraním a implementací metod definovaných tímto rozhraním. Dále jsou v této vrstvě implementovány objekty, které jsou při probíhající komunikaci mezi klientem a službou serializovány a v podobě datového proudu (*stream)* posílány na klienta. Druhou vrstvou je datová vrstva, která nám zajišťuje veškerou manipulaci s daty nad databází. K její implementaci jsem použil technologii LINQ to SQL od společnosti Microsoft fungující na principu mapování databázových tabulek. Každá databázová tabulka je ve formě entity implementována v datové vrstvě a dotazy do databáze jsem realizoval jazykem LINQ přímo v programovém kódu jazyka C#. Tímto řešením jsem docílil rozdělení výpočetní zátěže databázového serveru a přenesl ji z části na datovou vrstvu. Datové úložiště je řešeno databází, která je provozována na Microsoft SQL Serveru 2008. Struktura služby a její jednotlivé vrstvy jsou znázorněny na obrázku (Obr. 23).



Obr. 23: Architektura WCF služby

5.2 Technologické řešení implementace WCF služby

Implementoval jsem rozhraní `IService` WCF služby, které zpřístupňuje metody a umožňuje jejich volání externě. Toto rozhraní je zároveň vzdáleným koncovým bodem, ke kterému se připojuje klientská aplikace. V rozhraní jsem přiřadil atribut `[ServiceContract]` třídě reprezentující službu a jednotlivým metodám atribut `[OperationContract]`, podrobněji jsou oba předchozí atributy popsány v kap. 2.2. Následující zdrojový kód je částečnou ukázkou implementace služby, kterou jsem vytvořil.

```
[ServiceContract]
public interface IService
{
        #region User
        [OperationContract]
        User GetUserById(int id);

        [OperationContract]
        List<User> GetUsers(int page, int pageSize, SortParam sortParam,
        int companyId, UserType type, string search, out int count);

        [OperationContract]
        User Login(string login, string password);

        [OperationContract]
        List<UserRole> GetUserRole();

        [OperationContract]
        bool DeleteUser(int id);

        [OperationContract]
        WCFIdentityResult SaveUser(User newData);
        #endregion

        #region Company
        [OperationContract]
        bool InitDatabase();

        [OperationContract]
        List<Company> GetCompanies(int page, int pageSize, SortParam
        sortParam, string search, CompanyType type, out int count);

        [OperationContract]
        bool DeleteBranch(int id);

        [OperationContract]
        bool DeleteSupplier(int id);
...
#endregion
}
```

Jednotlivé metody rozhraní `IService` jsem následně implementoval v samostatných třídách služby. Uvedu zde zdrojový kód části implementace metod rozhraní v třídě `User`.

```
public partial class Service : IService
{
        #region IService Members
        public List<UserRole> GetUserRole()
        {
            List<UserRole> ctr = new List<UserRole>();
            Dictionary<int, string> roles =
            WcfBusiness.Extension.EnumValueDataAttribute.
            GetListItems(typeof(WcfBusiness.Extension.UserType));
            foreach (var item in roles)
            {
              ctr.Add(new UserRole { Id = item.Key, Name = item.Value });
            }
            return ctr;
        }
        public User GetUserById(int id)
        {
            using (WcfBusiness.WcfDataClassesDataContext kdc = new
            WcfBusiness.WcfDataClassesDataContext())
            {
                return Transform(WcfBusiness.User.GetUserById(kdc, id));
            }
        }
        public List<User> GetUsers(int page, int pageSize, SortParam
        sortParam, int companyId, UserType type, string search, out int
        count)
        {
            using (WcfBusiness.WcfDataClassesDataContext kdc = new
            WcfBusiness.WcfDataClassesDataContext())
            {
                return Transform(WcfBusiness.User.GetUsers(kdc, page,
                pageSize, search, sortParam, companyId, (int)type, out
                count).ToList());
            }
        }
...
}
```

Pro přenášení dat slouží objekty, které je nutné serializovat, aby je bylo možné posílat klientské aplikaci. Proto jsem vytvořil třídy objektů s parametry `[DataContract]` a `[DataMember]` viz. následující ukázka kódu třídy `User`. Použití a význam obou předchozích parametrů je vysvětleno v kapitole 2.2 a pojem serializace v kap. 1.2.

```
[DataContract]
public class User
{
        private int id;
        [DataMember]
        public int Id
        {
            get { return id; }
            set { id = value; }
        }
        private string firstName;
        [DataMember]
        public string FirstName
        {
            get { return firstName; }
            set { firstName = value; }
        }
        private string lastName;
        [DataMember]
        public string LastName
        {
            get { return lastName; }
            set { lastName = value; }
        }
        private string email;
        [DataMember]
        public string Email
        {
            get { return email; }
            set { email = value; }
        }
        private string login;
        [DataMember]
        public string Login
        {
            get { return login; }
            set { login = value; }
        }
...
}
```

Všechny metody, funkce a objekty tvoří samostatnou (transportní) vrstvu WCF služby. Tato vrstva je nadřazená datové vrstvě v hierarchii architektury služby. Datovou vrstvu jsem vytvořil využitím technologie LINQ[11] to SQL od společnosti Microsoft.

5.2.1 Datová vrstva

V datové vrstvě odpovídá každá třída objektu ekvivalentní entitě datového modelu, která je vygenerována pomocí vývojových prostředků již zmiňované technologie LINQ to SQL postavené na principu mapování jednotlivých databázových tabulek. Do každé vygenerované třídy jsem implementoval vlastní metody a vlastnosti, které jsou volány z „nadřazené“ transportní vrstvy. Tyto funkce ve většině případů provádí dotazy do databáze, které jsem napsal v dotazovací direktivě LINQ. Jako příklad uvedu následující kód, kde je znázorněna implementace entity `User`.

```
public partial class User
{
        public static IEnumerable<User> Get(WcfDataClassesDataContext
        kdc, int userType, SortParam sortParam)
        {
            int count = 0;
            return GetPaged(kdc, 0, 0, sortParam, userType, out count);
        }
        public static IEnumerable<User>
        GetPaged(WcfDataClassesDataContext kdc, int page, int pageSize,
        SortParam sortParam, int userType, out int count)
        {
            {
                var result = (from x in kdc.Users
                              where x.Discriminator == userType
                              select x);
                if (sortParam != null) result =
                result.OrderBy(sortParam.ToString());
                count = result.Count();
```

[11] LINQ to SQL je technologie od společnosti Microsoft, které umožňuje snadný a rychlý vývoj aplikací manipulujících s daty a je postaveno na principu mapování databázových tabulek do entit vytvořených přímo v kódu programovacího jazyka.

```
                if (page == 0 && pageSize == 0)
                {
                    return result;
                }
                else
                {
                    //paged
                    return result.Skip((page - 1) *
                    pageSize).Take(pageSize);
                }
            }
        }
        public static User LoginUser(WcfDataClassesDataContext kdc,
        string login, string password)
        {
            User user = null;
            {
                user = kdc.Users.FirstOrDefault(u =>
                u.Login.Equals(login) && u.Password.Equals(password));
                return user;
            }
        }
...
}
```

5.3 Technologické řešení klientské aplikace

Klientskou aplikaci jsem vyvinul v prostředí .NET Framework technologickými prostředky Windows Presentation Foundation[12] (WPF). WPF je nástupnickou technologií v .NET Framework pro vytváření moderního grafického uživatelského rozhraní formulářových aplikací. Pro vytvoření klientské aplikace jsem použil návrhový vzor MVVM (*Model-View-ViewModel*). Tento návrhový vzor zde nebudu detailně popisovat, neboť jeho podrobné vysvětlení není předmětem této práce, ale uvedu jeho zevrubný popis. Tento model dotváří základní charakteristiku architektury klientské aplikace a využívá

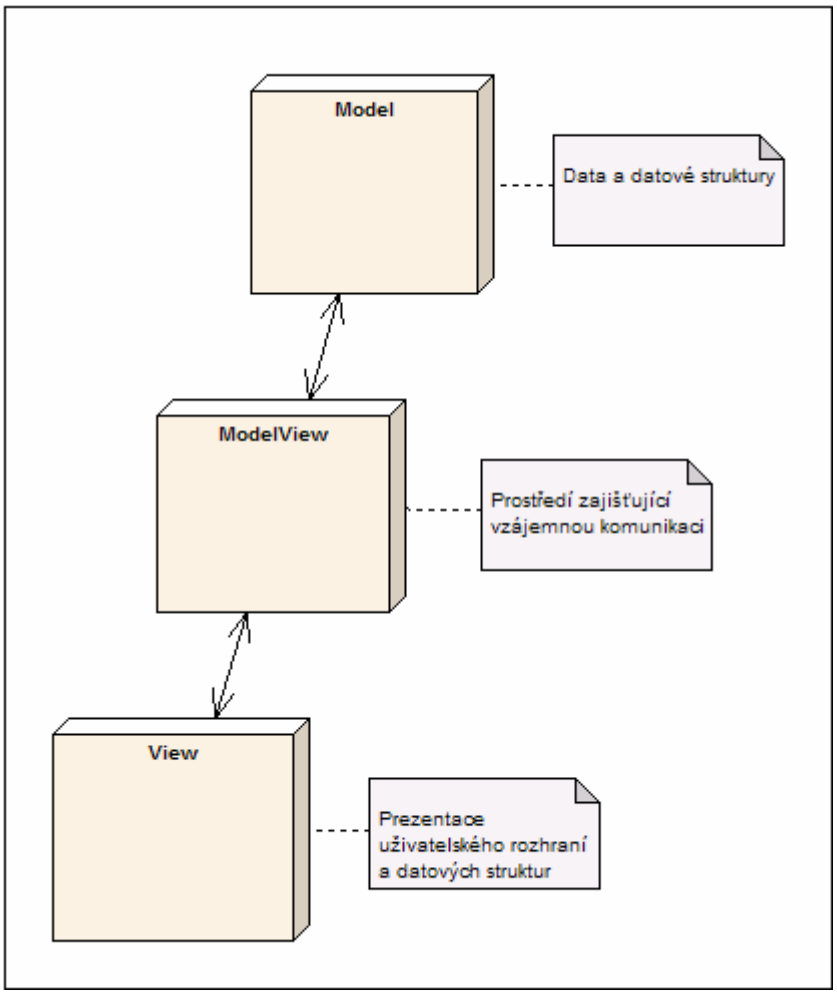
[12] WPF je technologie pro vývoj moderního uživatelského prostředí formulářových aplikací v .NET Framework a nahrazuje tak starší Winfoms.

zabudované funkcionality oboustranného *databinding*[13] v technologii WPF. Zjednodušeně řečeno je klientská aplikace rozdělena na tři části *Model*, *View* a *ViewModel*, které spolu úzce souvisí viz. obrázek (Obr. 24). *Model* reprezentuje datové struktury (objekty, pole, atd.) a veškerou jejich funkcionalitu, logiku a výpočty. *View* je vizuální prezentací *Modelu*, tedy datových struktur v paměti a dalších prvků grafického uživatelského rozhraní. *ViewModel* představuje prostředníka mezi *View* a *Model*, který provádí následující činnosti:

- adaptuje *Model* pro potřeby *View*,
- je zodpovědný za stav View ,
- volá metody *Modelu* přes jejich rozhraní,
- volá model nebo služby přes jejich interface,
- vystavuje veřejné vlastnosti objektů, které jsou prostřednictvím *Databinding* zpřístupňovány z *Modelu* do *View*.

Aby mohl *View* a *ViewModel* vzájemně komunikovat musí být veřejné vlastnosti implementovány jako notifikační. Jinak řečeno, aby vlastnosti třídy mohly být publikovány ve *View* musí třída implementovat rozhraní `INotifyPropertyChanged`, které vyvolává událost, při jakékoliv změně v datové struktuře *Modelu*. Kolekce prvků musí být implementovány pomocí `ObservableCollection<T>`. V následujícím kódu je na ukázku část třídy `wpfUser`, jak je naprogramovaná v *Modelu* klientské aplikace. Je zde implementováno i výše zmiňované rozhraní `INotifyPropertyChanged`.

[13] Databinding je technologický prostředek, který dokáže reagovat na změny v datech a automaticky je přenést je do grafické vizualizace a naopak. Tím udržuje neustále aktuálnost dat mezi daty a jejich grafickou reprezentací.



Obr. 24: Architektura návrhového vzoru Model-View-ViewModel

```
public class wpfUser : wpfBase<User>
{
        public event EventHandler UserSavedFail;
        public event EventHandler UserSaved;
        public event EventHandler UserLoad;
        public event EventHandler UserLoadFail;
        #region properties
        private string firstName;
        public string FirstName
        {
            get { return firstName; }
            set
            {
                firstName = value;
                this.NotifyPropertyChanged("FirstName");
                if (String.IsNullOrEmpty(value))
                    throw new ApplicationException("Jméno je povinné");
            }
```

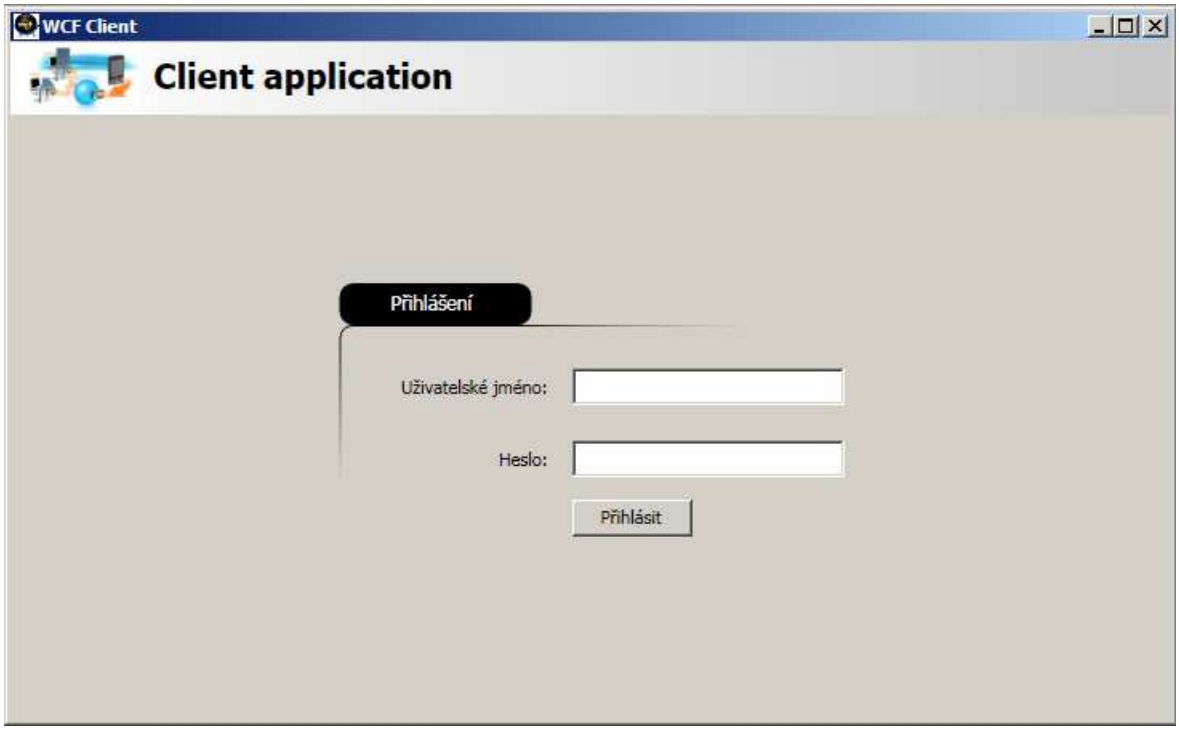
```
        }
        private string lastName;
        public string LastName
        {
            get { return lastName; }
            set
            {
                lastName = value;
                this.NotifyPropertyChanged("LastName");
                if (String.IsNullOrEmpty(value))
                   throw new ApplicationException("Příjmení je povinné");
            }
        }
...
}
```

5.3.1 Funkční struktura a popis uživatelského rozhraní aplikace

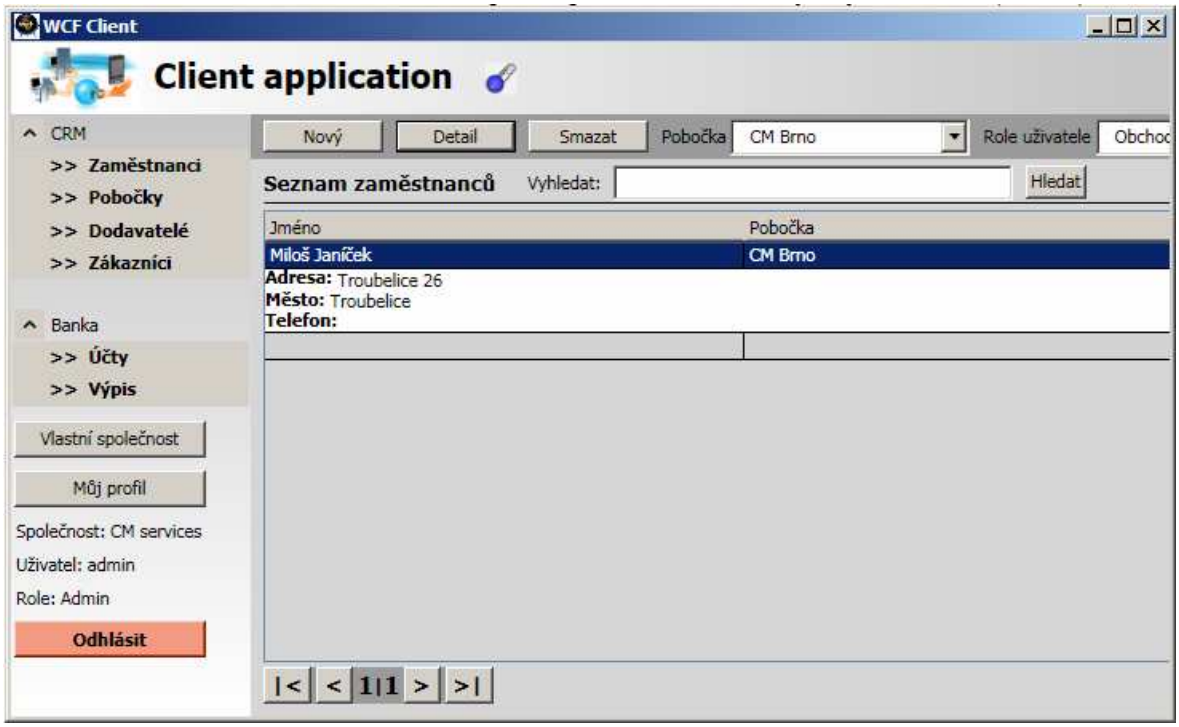
V aplikaci jsem vytvořil dva samostatné moduly, modul banka a modul CRM (*Customer relationship management*)[14]. I když jsou oba moduly samostatné, je mezi nimi určitá provázanost např. na úrovni uživatelských rolí a přístupu k jednotlivým funkcím. Veškerou funkcionalitu programu z hlediska uživatele i programátora jsem znázornil diagramem případů užití (*Use Case diagram*) na obrázku (Obr. 27).

Při spuštění programu, pokud je vše správně nastaveno a služba zprovozněna, zobrazí se v hlavním okně klientské aplikace přihlašovací formulář jako je na obrázku (Obr. 25), ve kterém musí každý uživatel zadat své přihlašovací jméno a heslo. Při prvním spuštění aplikace dojde k inicializaci databáze a je vytvořen účet administrátora s uživatelským jménem i heslem „admin“. Po přihlášení se otevře hlavní formulář celé aplikace, kde je v levé části hlavní menu a v pravé aktuálně otevřený seznam viz. obrázek (Obr. 26).

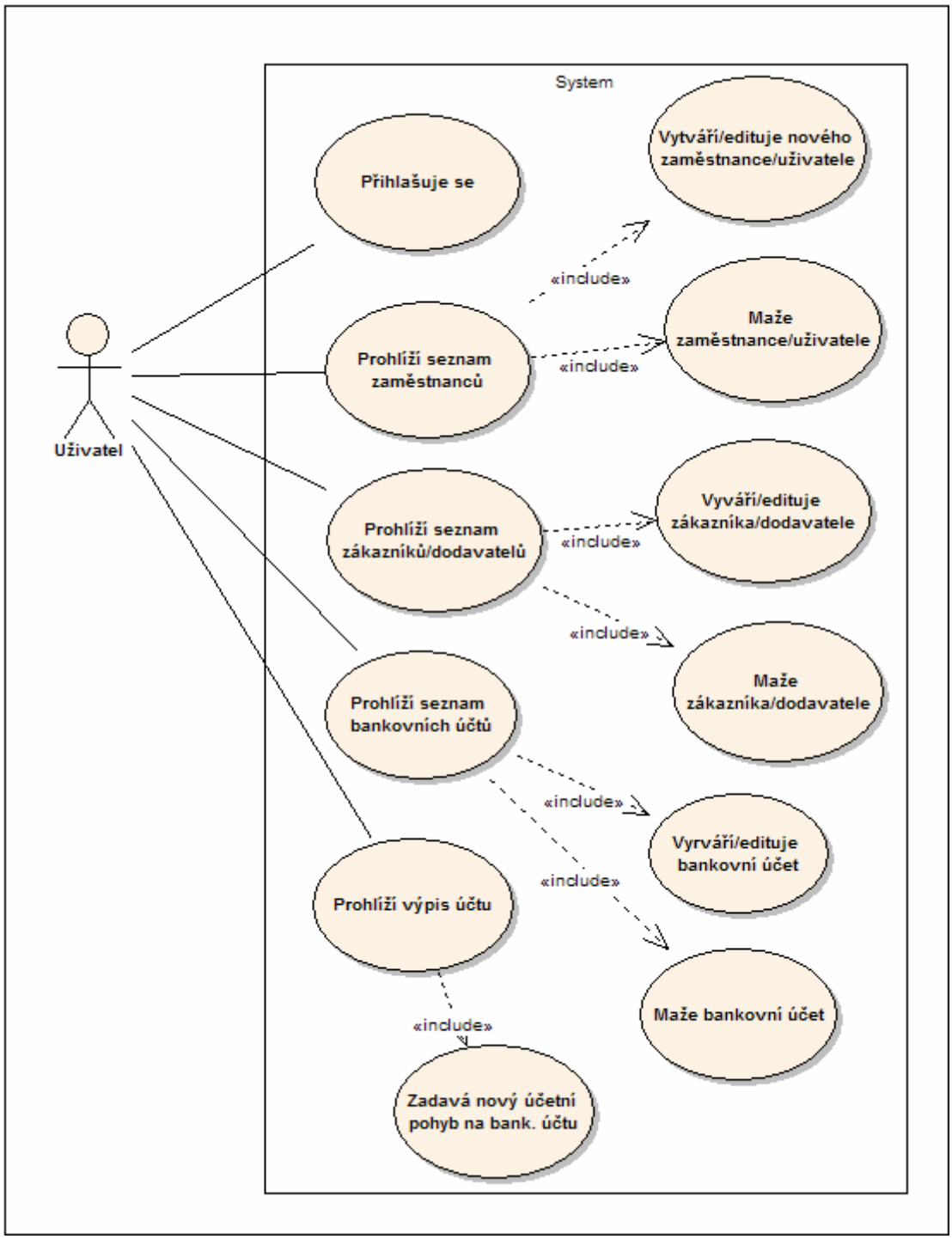
[14] *Customer relationship management* je proces shromažďování, zpracování a využití informací o zákaznících, zaměstnancích, dodavatelích, atd. společnosti.



Obr. 25: Přihlašovací formulář klientské aplikace



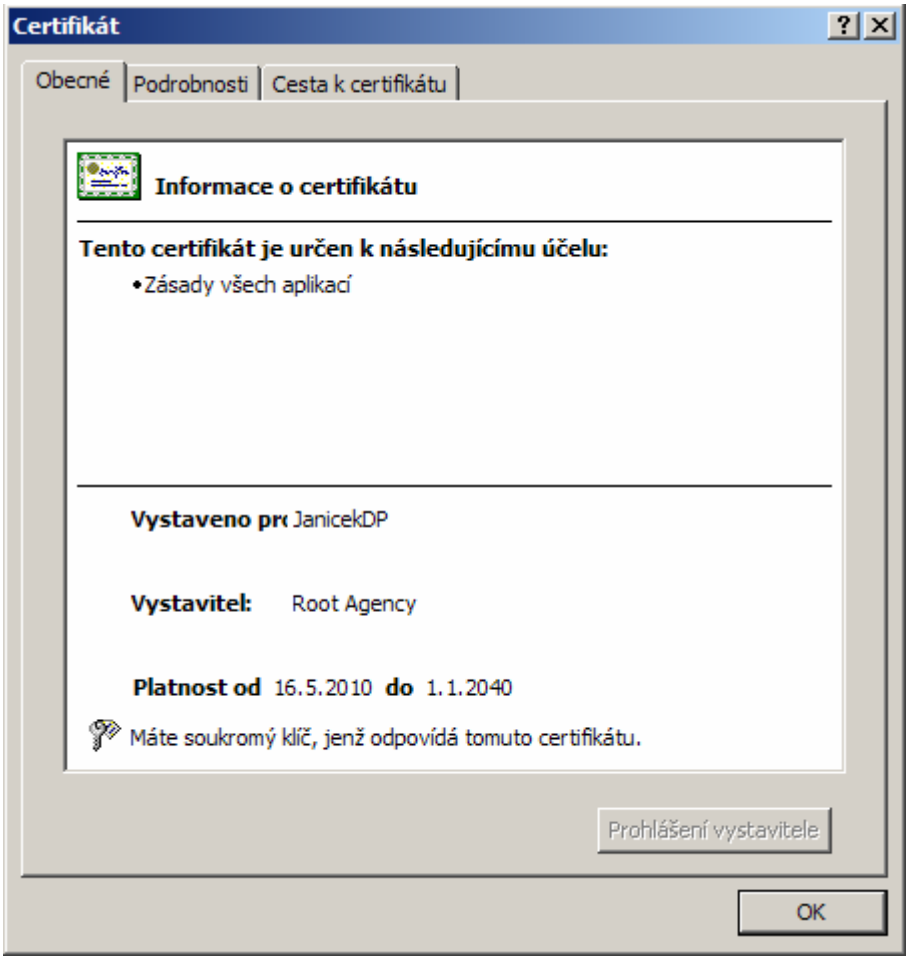
Obr. 26: Hlavní formulář klientské aplikace



Obr. 27: Use Case diagram

6 ZABEZPEČENÍ WCF SLUŽBY

Komunikační kanál mezi WCF službou a klientskou aplikací jsem zabezpečil certifikátem typu *self-signed* viz. obrázek (Obr. 28), což je certifikát, který jsem si sám vygeneroval na lokálním počítači a slouží pouze pro testovací účely. V ostrém provozu požádáme o vydání certifikátu kvalifikovanou certifikační autoritu, která certifikát vygeneruje, podepíše a tím potvrdí pravost uvedených údajů v certifikátu. Službu jsem nakonfiguroval tak, aby ji bylo možné provozovat na HTTPS protokolu, který pro zabezpečení používá vrstvu SSL, a implementoval jsem ověřovací metodu klienta na straně služby. Podrobné informace o certifikační autoritě a digitálním certifikátu se dočtete v kapitolách 4.2., 4.3. a protokol HTTPS a vrstva SSL je detailně popsána v kapitolách 4.1. a 4.1.3.



Obr. 28: Vygenerovaný certifikát

6.1 Vytvoření vlastního certifikátu a navázání na komunikační port

Certifikát jsem vytvořil následujícím příkazem z příkazové řádky pomocí utility *makecert* operačního systému Windows. Podmínkou pro vygenerování certifikátu je umístění zmiňované utility, kterou obsahuje přiložené CD (\DistributedApplication\Utility), ve složce v jaké spouštíme tento příkaz. Utilita *makecert* a vytvoření certifikátu typu *self-signed* je podrobně popsáno v teoretické části kapitola 4.3.5, ve které se také dočtete, jak je možné překontrolovat, zda byl certifikát vytvořen.

```
makecert.exe -sr LocalMachine -ss MY -a sha1 -n CN=JanicekDP -sky
exchange –pe
```

Vytvořením certifikátu práce s ním ještě nekončí, dále je velice důležité pro zabezpečení komunikačního kanálu tento certifikát navázat na komunikační port. K tomu jsem použil utilitu *httpcfg* operačního systému Windows, která je rovněž na přiloženém CD (\DistributedApplication\Utility). Komunikační port jsem zvolil `:3682` a navázal jsem na něj certifikát následujícím příkazem z příkazové řádky. Stejně jako u předchozího platí, že příkaz musí být spuštěn ve složce v jaké se nachází utilita.

```
httpcfg set ssl -i 0.0.0.0:3682 -h
0ad2d666b7f483caeba8d1971309636d3cff88ca
```

Je důležité si uvědomit, že tímto příkazem, jsem nenavázal na komunikační port konkrétně certifikát, ale jeho hashovací značku (miniaturu) viz. kapitola 4.5.1.

6.2 Konfigurace zabezpečení WCF služby

Po vytvoření certifikátu a provázání s komunikačním portem jsem v konfiguračním souboru provedl nastavení služby, aby podporovala zabezpečenou komunikaci a nastavil URL adresu, na které ji budu provozovat. V následujícím kódu jsou tučně vyznačeny části, které jsem nastavil pro bezpečný provoz služby v konfiguračním souboru.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <compilation debug="true" />
```

```
  </system.web>
  <system.serviceModel>
    <bindings>
      <mexHttpsBinding>
        <binding name="MexBinding" />
      </mexHttpsBinding>
      <basicHttpBinding>
        <binding name="Binding" maxReceivedMessageSize="20971520"
        maxBufferPoolSize="20971520">
          <readerQuotas maxStringContentLength="20971520"
          maxArrayLength="20971520" maxBytesPerRead="8192"
          maxNameTableCharCount="20971520"/>
          <security mode="Transport"></security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <services>
      <service name="WcfServiceLibrary.Service"
        behaviorConfiguration="WcfServiceLibrary.ServiceBehavior">
        <host>
          <baseAddresses>
            <add baseAddress=
     "https://localhost:3682/WCF_Addresses/WcfServiceLibrary/Service/" />
          </baseAddresses>
        </host>
        <endpoint address="mex" binding="mexHttpsBinding"
         contract="IMetadataExchange" />
        <endpoint address="ServiceA" binding="basicHttpBinding"
            bindingConfiguration="Binding"
            name="Service" bindingName="Binding"
            contract="WcfServiceLibrary.IService" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="WcfServiceLibrary.ServiceBehavior">
          <serviceMetadata httpsGetEnabled="true"
          policyVersion="Policy15" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

6.3 Implementace autentizace klienta

K ověření klienta jsem vytvořil formulář pro přihlášení viz. obrázek (Obr. 25), kde se po zadání údajů odešle přihlašovací jméno a heslo WCF službě. Zde jsem implementoval metodu, která ověří platnost jména a hesla uživatele a v případě, že je uživatel ověřen, provede přihlášení a uloží uživatelskou roli do proměnné `private static UserType role;`. Metoda pro ověření uživatele je znázorněna v následující ukázce zdrojového kódu.

```
public User Login(string login, string password)
{
     using (WcfBusiness.WcfDataClassesDataContext kdc = new
     WcfBusiness.WcfDataClassesDataContext())
     {
User user = Transform(WcfBusiness.User.LoginUser(kdc, login,
System.Web.Security.FormsAuthentication.HashPasswordForStoringInConfigFil
e(password, "MD5")));
                if (user != null)
                {
                    role = user.Type;
                    return user;
                }
                else
                    return null;
            }
}
```

Pro autentizaci klienta jsem vytvořil metodu `AuthenticateClient(UserType role)`, která přebírá jako parametr roli uživatele v systému a podle toho rozhoduje zda je uživatel oprávněn provádět volanou operaci ve službě nebo nikoliv. Pokud uživatel nemá oprávnění k volání metody, služba vyvolá vyjímku typu `SecurityAccessDeniedException` a pošle ji klientovi. Tato kontrola práv uživatele má dva důvody. Prvním z nich je, že také služba musí vědět, že komunikuje opravdu z oprváněnou osobou systému a nemůže dojít k situaci, kdy si nějaký šikovný programátor naprogramuje vlastní klientskou aplikaci a bude libovolně využívat její funkce. Druhým důvodem je, aby i přihlášený uživatel měl omezená práva vztahující se ke své roli, např. aby jakýkoliv zaměstnanec nemohl libovolně přepisovat nebo mazat data či jinak „škodit“ i neúmyslně v systému. V komerčních systémech jsou funkce uživatelského rozhraní, které z titulu své role nemůže přihlášený uživatel používat zneviditelněny. V aplikaci, kterou jsem vytvořil tuto funkcionalitu záměrně neskrývám právě proto, aby bylo možné při neoprávněném použití funkce vyvolat a vizuálně si ověřit vyjímku `SecurityAccessDeniedException`. Řešení mojí autentizační metody představuje následující ukázka kódu.

```
private void AuthenticateClient(UserType role, User user)
{
      if (role != user.Type)
      {
          throw new SecurityAccessDeniedException("Nemáte oprávnění
         k provedení této operace");
      }
      if (user == null)
      {
          throw new SecurityAccessDeniedException("Nemáte oprávnění
          k provedení této operace");
      }
      KeyValuePair<string, string> pair = new KeyValuePair<string,
      string>(user.Login, user.Password);
      if (!Session.CurrentSession.Contains(pair))
      {
          throw new SecurityAccessDeniedException("Nemáte oprávnění
          k provedení této operace");
      }
}
```

Pro zvýšenou bezpečnost aplikace jsem implementoval funkcionalitu hashování uživatelského hesla a díky tomu je do databáze ukládán pouze hash textového řetězce. Při ověřování hesla uživatele, který se do systému přihlašuje se porovnávají hashované textové řetězce a samotná hodnota hesla zůstává utajena i pro správce databáze. Hashování provádím následující funkcí. Podrobnosti o zabezpečení citlivých dat se dočtete v kap. 3.7.

```
...
if (!String.IsNullOrEmpty(this.Password))
{
        string password =
System.Web.Security.FormsAuthentication.HashPasswordForStoringInConfigFil
e(this.Password, "MD5");
        this.Password = password;
}
...
```

7 INSTALACE A NASTAVENÍ APLIKACE

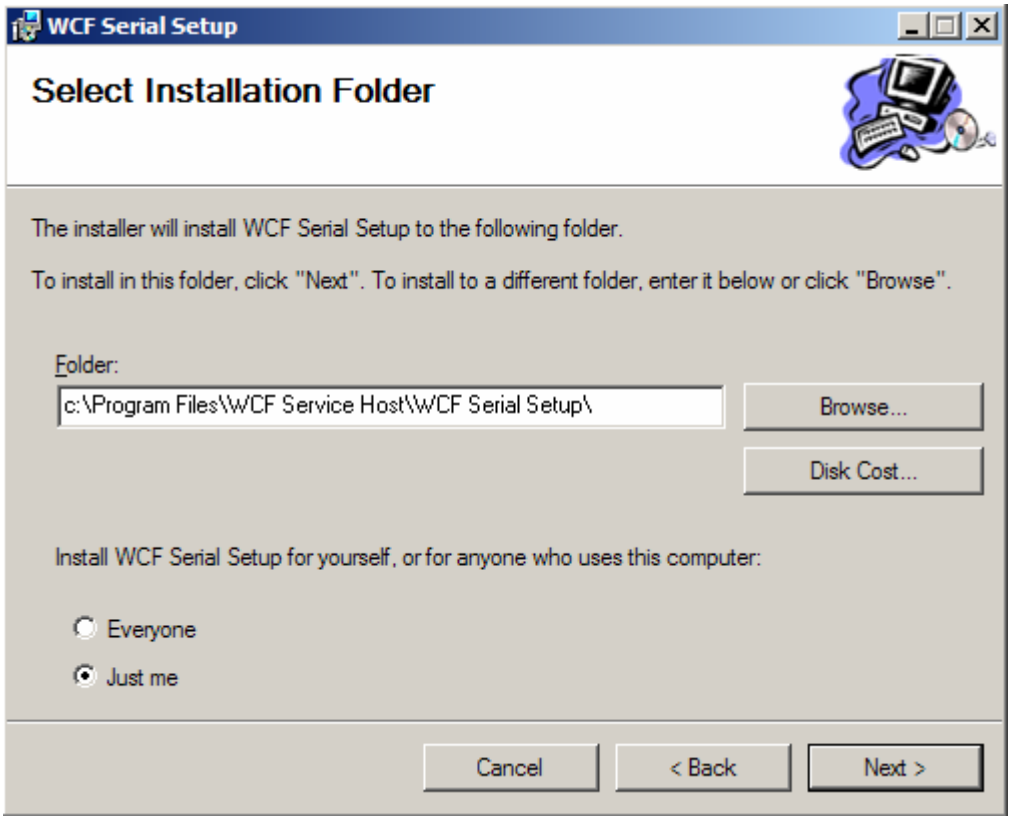
Instalace distribuované aplikace není nijak náročná, ale má určitá specifika oproti instalaci klasických programů. Klientská aplikace i WCF služba se instaluje zvlášť a na přiloženém CD je k oběma částem dodán instalátor. Kromě instalace těchto dvou celků je nutné vytvořit certifikát a navázat jej na komunikační port, nainstalovat Microsoft SQL Server 2008, kde vytvoříme ze skriptu databázi. Na klientské stanici i na serveru je k běhu obou částí aplikace vyžadováno nainstalovat .NET Framework 3.5 SP1. Produkty Microsoft SQL Server 2008 (stačí Express verze) i .NET Framework 3.5 SP1 je možné vyhledat a stáhnout z internetu na webových stránkách http://www.microsoft.com/cs/cz/.

7.1 Vytvoření databáze

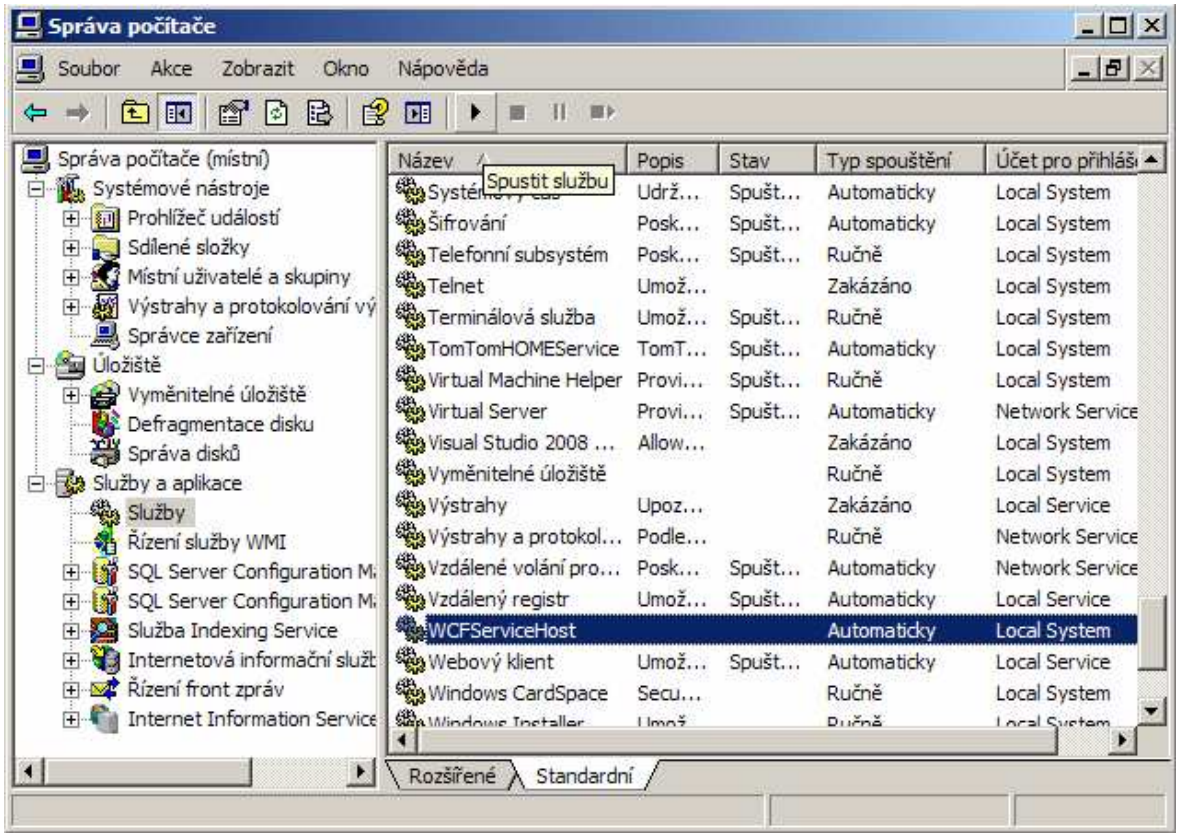
V manažeru SQL Serveru 2008 (Management Studio 2008) vytvořte databázi s názvem `WCF_db` a spusťte SQL skript, který naleznete na přiloženém CD ve složce \Database soubor `SQLQuery.sql`. Tímto skriptem se vytvoří struktura tabulek a databáze je připravena k použití.

7.2 Instalace a zprovoznění WCF služby

Na přiloženém CD ve složce `\WCF Serial Setup` je instalační soubor `setup.exe` služby WCF. Po spuštění instalačního souboru se zobrazí klasický průvodce instalací, ve kterém je možné nastavit např. cílovou složku a další parametry instalace viz. obrázek (Obr. 29). Služba se spouští automaticky při každém spuštění operačního systému, ovšem po instalaci je standardně zastavená a spustit ji lze ručně v ovládacích panelech Windows viz. obrázek (Obr. 30) nebo restartováním počítače. Před vlastním spuštění služby je potřeba vytvořit certifikát a navázat jej na komunikační port adresy, na které budeme službu provozovat, jak je popsáno v kapitole 6.1.

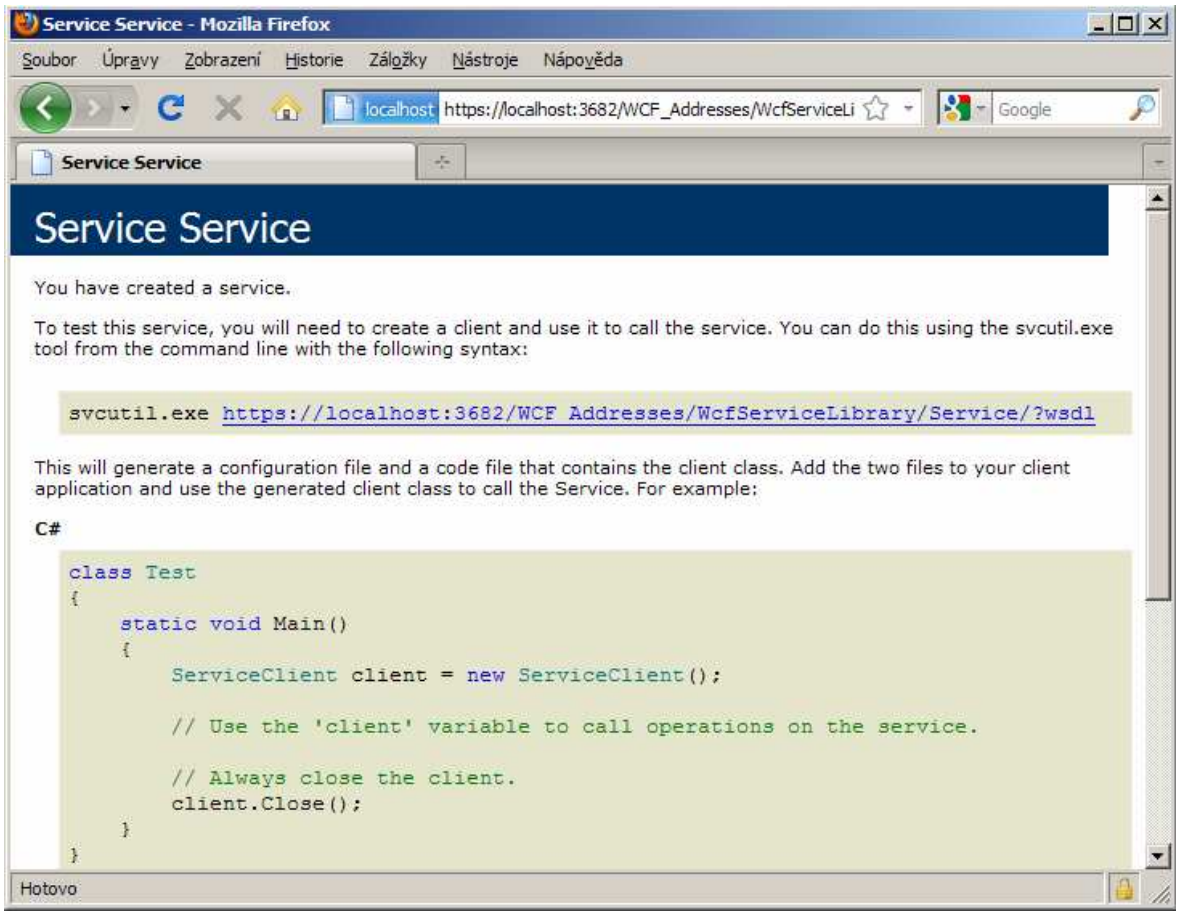


Obr. 29: Průvodce instalací WCF služby



Obr. 30: Spuštění služby

Pokud je nainstalováno prostředí .NET Framework 3.5 SP1, správně vytvořený certifikát a službu se podařilo bez konfliktů spustit, můžeme si ověřit ve webovém prohlížeči zda služba běží. Otevřeme libovolný webový prohlížeč a do adresy zadáme URL adresu naší služby. Pokud je vše pořádku měli bychom v prohlížeči přejít na stránku s rozhraním WCF služby, jak je na obrázku (Obr. 31).



Obr. 31: Kontrola služby ve webovém prohlížeči

7.3 Instalace klientské aplikace

Instalace klientské aplikace je obdobná jako u WCF služby, kterou jsem popsal v kapitole 7.2. I v tomto případě je na přiloženém CD ve složce `\WPF Client Setup` k dispozici instalační soubor `setup.exe`. Po jeho spuštění Vás standardní průvodce provede instalací viz. obrázek (Obr. 29). Při instalaci by měl být automaticky přidán na pracovní plochu a do hlavního menu zástupce klientské aplikace, kterým ji po dokončení instalace můžete spustit.

ZÁVĚR

Windows Communication Foundation je moderní technologie od firmy Microsoft, která přináší velké zjednodušení a zrychlení práce vývojářů i co se týká oblasti bezpečnosti komunikace klienta se serverem. WCF nám nabízí více způsobů implementace zabezpečení, který z nich zvolíme, závisí na způsobu hostování služby. V praxi je při použití WCF vždy velice důležité zvolit správný způsob nasazení služby. Pokud bude služba využívána ve vnitropodnikové síti a nebude zpřístupněná v prostředí internetu, je ideálním řešením pro hostování služby a úroveň zabezpečení Windows služba. Pokud nás ale požadavky na komunikaci aplikace donutí službu využívat i v internetu, je nutné zvolit úroveň zabezpečení a hostování v Internetové Informační službě např. IIS7.

V dnešních moderních distribuovaných aplikacích jako jsou například rozsáhlé *Enterprise Resource Planning* (ERP) systémy je již běžné, že mezi standardní požadavky patří dostupnost odkudkoliv z internetu, aby pro všechny uživatele v systému byl operativně dostupný a aby informace, které uchovává, byly vždy k dispozici. Je velice časté, že prostřednictví internetu si klientská aplikace se serverem (službou) vymění velké množství interních informací, ať už podnikových nebo osobních, v každém případě musí být samozřejmostí tato data zabezpečit proti odcizení útočníkem třetí strany tzv. počítačovým hackerem, jehož zájmem by bylo zneužití těchto informací.

V této práci jsem zvolil zabezpečení WCF služby hostované ve Windows službě z důvodu snadnější demonstrace na lokálním počítači. Stejné postupy zabezpečení je možné použít i při hostování v Internetové Informační službě např. IIS7. Jediným rozdílem je způsob instalace služby. V oblasti zabezpečení se mi podařilo službu nastavit tak, aby ji bylo možné provozovat na HTTPS protokolu využívající SSL vrstvu. Pro zabezpečení komunikačního kanálu mezi klientem a serverem jsem použil digitální certifikát typu *self-signed*. Vyřešil jsem ověření klienta vlastní ověřovací metodou služby, tak aby nemohlo dojít k volání vystavených metod služby nepovolanými osobami.

ZÁVĚR V ANGLIČTINĚ

Windows Communication Foundation is a new technology from Microsoft that delivers great speed and simplification the work of developers and what concerns the security of client server communication. WCF offers us multiple ways to implement security, which one we choose depends on the type of hosting service. In practice, when using WCF it is always very important to choose the correct way of deploying service. If the service is used in the internal network and will not be available in the internet environment, Windows service is ideal resolution for hosting service and the security level. But if requirements on comunication of aplication compel us to use the service in internet, it is necessary to choose the security level and hosting in internet information service e.g. IIS7.

In today's modern distributed applications such as large Enterprise Resource Planning (ERP) systems is common that the standard requirements include availability from anywhere on the internet, that for all users of the system will be operationally available and the informations which it holds, will be always available. It is very common that through the internet can a client application and a server (service) exchanges a large amount of internal informations, either corporate or personal, in any case it must be obvious that this data are secured against theft attacker of third party called a computer hacker, whom interest would be misuse of these information.

For easier demonstration at a local computer I choose security of WCF service hosted in Windows service in this work.The same security procedures can also be used during hosting in internet information service such as IIS7. The only difference is the method of installation services. In the area of security I managed to configure the service so that it can be used with HTTPS protocol which uses SSL layer. I used the digotal certificate type self-signed to secure communication channel between the client and the server. I solved the client´s authentication thank my own service authentification method so that exposed service methods couldn´t be called by unauthorised person.

SEZNAM POUŽITÉ LITERATURY

Monografie:

[1] SHARP, John. *Microsoft Windows Communication Foundation Step by Step*. Praha : Microsoft Press, 2007. 448 s. ISBN 9780735623361.

[2] KLEIN, Scott. *Professional WCF programming: .NET development with the Windows Communication Foundation*. Indianapolis : John Wiley & Sons, 2007. 430 s. ISBN 978-0-470-08984-2.

[3] PEIRIS, Chris, MULDER, Dennis. *Pro WCF: practical Microsoft SOA implementation*. Berkeley : Apress, 2007. 475 s. ISBN 978-1-59059-702-6.

[4] DOSTÁLEK, Libor, et al. *Velký průvodce protokoly TCP/IP: Bezpečnost* . 2. aktualiz. vyd. Brno : Computer Press, 2003. 592 s. ISBN 80-7226-849-X.

[5] SHARP, John. *Microsoft Visual C# 2008 krok za krokem*. Brno : Computer Press, 2008. 592 s. ISBN 978-80-251-2027-9.

[6] SHARP, John, JAGGER, Jon. *Microsoft Visual C# .NET krok za krokem*. Brno : Mobil Media a.s., 2002. 655 s. ISBN 80-86593-27-4.

[7] PIPER, Fred, MURPHY, Sean. *Kryptografie*. Praha : Dokořán, 2006. 157 s. ISBN 80-7363-074-5.

[8] KOPKA, Martin. *Počítačové sítě*. Olomouc : Univerzita Palackého v Olomouci, 1996. 98 s.

Internetové zdroje:

[9] Certificate Creation Tool (Makecert.exe). *MSDN .NET Framework Developer Center* [online]. 2009 [cit. 2009-11-16]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/bfsktky3%28VS.80%29.aspx>.

[10] *Certifikační autorita* [online]. [s.l.] : [s.n.], 3.1.2010 [cit. 2009-11-26]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Certifika%C4%8Dn%C3%AD_autorita>.

[11] *Digitální certifikát* [online]. [s.l.] : [s.n.], 6.3.2010 [cit. 2009-11-25]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Digit%C3%A1ln%C3%AD_certifik%C3%A1t>.

[12] LAMPA, Petr. *CORBA a IIOP* [online]. Brno : Ústav informatiky a výpočetní techniky, Fakulta elektrotechniky a informatiky, VUT v Brně, [2005] [cit. 2010-04-13]. Dostupné z WWW: <http://www.fit.vutbr.cz/~lampa/papers/corba.html>.

[13] ANTONIO, Robert. *Component Object Model* [online]. [s.l.] : [s.n.], 22.9.2003 [cit. 2010-04-13]. Dostupné z WWW: <http://antonio.cz/com/1.html>.

[14] STRNAD, Pavel. *Remote Method Invocation* [online]. [s.l.] : [s.n.], 2.5.2000 [cit. 2010-04-13]. Dostupné z WWW: <http://www.volny.cz/drd/rmi.html>.

[15] BOSÁK, Rostislav; FANTA, Martin; PEŘINA, Martin. *Kapitola 13 Distribuované (objektové) systémy* [online]. [s.l.] : [s.n.], 1999, 19.9.2000 [cit. 2010-05-01]. Dostupné z WWW: <http://www.ataco.cz/perina/par-apek/Chapter13/Chap13.html>.

[16] *Remote procedure call* [online]. [s.l.] : [s.n.], 20.4.2010 [cit. 2010-02-05]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/RPC>.

[17] LARYŠ, Kryštof. *WCF pro začátečníky – 2. díl: contract, binding, service behavior* [online]. 13. února 2009 [cit. 2010-05-01]. Dostupné z WWW: <http://www.netstudent.cz/Články/tabid/56/articleType/ArticleView/ArticleID/226/PageID/209/Default.aspx>.

[18] LARYŠ, Kryštof. *WCF pro začátečníky – 1. díl: teorie, základní pojmy* [online]. 13. února 2009 [cit. 2010-05-01]. Dostupné z WWW: <http://www.netstudent.cz/%C4%8Cl%C3%A1nky/tabid/56/articleType/ArticleView/articleId/224/WCF-pro-zatenky--1-dl-teorie-zkladn-pojmy.aspx>.

[19] *Asymetrická kryptografie* [online]. [s.l.] : [s.n.], 28.3.2010 [cit. 2010-11-21]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Asymetrick%C3%A1_kryptografie>.

[20] *HTTPS* [online]. [s.l.] : [s.n.], 15.3.2010 [cit. 2010-11-23]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/HTTPS>.

[21] Microsoft Corporation. *Httpcfg Syntax* [online]. [s.l.] : [s.n.], 23.3.2003 [cit. 2009-12-18]. Dostupné z WWW:

<http://technet.microsoft.com/en-us/library/cc781601%28WS.10%29.aspx>.

[22] *Secure Sockets Layer* [online]. [s.l.] : [s.n.], 26. 6. 2009 [cit. 2010-15-03]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Secure_Sockets_Layer>.

[23] PUŠ, Petr. *Poznáváme C# a Microsoft.NET 40. díl – serializace* [online]. [s.l.] : [s.n.], 9.9.2005 [cit. 2010-02-12]. Dostupné z WWW:

<http://www.zive.cz/clanky/poznavame-c-a-microsoftnet-40-dil--serializace/sc-3-a-126553/default.aspx>.

[24] EVANS, Simon. *Overview of WCF architecture* [online]. [s.l.] : [s.n.], 17.9.2007 [cit. 2010-03-17]. Dostupné z WWW:

<http://consultingblogs.emc.com/simonevans/archive/2007/09/17/A-comprehensive-guide-to-using-MsmqIntegrationBinding-with-MSMQ-3.0-in-WCF.aspx>.

[25] LARYŠ, Kryštof. *WCF pro začátečníky – 5. díl: ošetření chyb na straně služby pomocí FaultContract* [online]. [s.l.] : Ústav informatiky a výpočetní techniky, Fakulta elektrotechniky a informatiky, VUT v Brně, 20.3.2010 [cit. 2010-05-01]. Dostupné z WWW:

<http://www.netstudent.cz/%C4%8Cl%C3%A1nky/tabid/56/articleType/ArticleView/articleId/303/Default.aspx>.

[26] Microsoft Corporation. *Internet Unsecured Client and Service* [online]. [s.l.] : [s.n.], 2010 [cit. 2010-05-01]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms733091%28v=VS.100%29.aspx>.

[27] Microsoft Corporation. *Intranet Unsecured Client and Service* [online]. [s.l.] : [s.n.], 2010 [cit. 2010-05-01]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms734784%28v=VS.100%29.aspx>.

[28] KOŠŤÁL, Marián. *WCF - Security* [online]. [s.l.] : [s.n.], 7.5.2007 [cit. 2010-05-01]. Dostupné z WWW: <http://www.vyvojar.cz/Articles/470-wcf-security.aspx>.

[29] Microsoft Corporation. *Security Overview* [online]. [s.l.] : [s.n.], 2010 [cit. 2010-01-04]. Dostupné z WWW:

<http://msdn.microsoft.com/en-us/library/ms735093%28v=VS.100%29.aspx>.

[30] *WCF FAQ Part 3 - 10 security related FAQ* [online]. [s.l.] : [s.n.], 23.6.2009 [cit. 2010-01-10]. Dostupné z WWW:

<http://www.c-sharpcorner.com/UploadFile/shivprasadk/35645765052620090511 28AM/35645765.aspx>.

[31] Microsoft Corporation. *Programming WCF Security* [online]. [s.l.] : [s.n.], 2010 [cit. 2009-11-20]. Dostupné z WWW:

<http://msdn.microsoft.com/en-us/library/ms731925%28v=VS.100%29.aspx>.

[32] Microsoft Corporation. *How to: Use the ASP.NET Membership Provider* [online]. [s.l.] : [s.n.], 2010 [cit. 2010-11-23]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms731049%28v=VS.100%29.aspx>.

[33] Microsoft Corporation. *How to: Use a Custom User Name and Password Validator* [online]. [s.l.] : [s.n.], 2010 [cit. 2010-11-28]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/aa702565%28v=VS.100%29.aspx>.

[34] Microsoft Corporation. *Service Identity and Authentication* [online]. [s.l.] : [s.n.], 2010 [cit. 2010-11-28]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms733130%28v=VS.100%29.aspx>.

[35] JIRAVA, J. *Úvodní pohled na membership provider* [online]. [s.l.] : [s.n.], 2009 [cit. 2010-05-01]. Dostupné z WWW: <http://jirava.net/blog/Archive/pohled-na-membership-provider.aspx>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

WCF	Windows Communication Foundation.
IIS	Internetová Informační Služba.
HTTP	Hypertext Transfer Protocol.
RMI	Remote Method Invocation.
ORB	Object Request Broker.
RPC	Remote procedure call.
COM	Component Object Model.
DCOM	Distributed extension of the Component Object Model.
CORBA	Common Object Request Brooker Architecture.
OMG	Object Management Group.
IDL	Interface Definition Language.
DII	Dynamic Invocation Interface.
DSI	Dynamic Skeleton Interface.
GIOP	General Inter ORB Protocol.
MSMQ	Microsoft Message Queuing.
SOA	Service-Oriented Architecture.
SOAP	Simple Object Access Protocol.
CA	Certifikační autorita.
XML	Extensible Markup Language.
MD	Message-Digest.
SHA	Secure Hash Algorithm.
SSL	Secure Sockets Layer.
OSI	Open System Interconnection.
URL	Uniform Resource Locator.

SQL	Structured Query Language.
LINQ	Language Integrated Query.
WPF	Windows Presentation Foundation.
MVVM	Model-View-ViewModel.
CRM	Customer relationship management.

SEZNAM OBRÁZKŮ