JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Pedagogická fakulta

Katedra informatiky

Služby WCF

Bakalářská práce

Boris Eninger

Vedoucí práce: Ing. Václav Novák, CSc.

České Budějovice, 2009

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách.

V Českých Budějovicích dne 15.4.2009 …….……...

Boris Eninger

Anotace

Cílem této práce je seznámit čtenáře s moderní technologií Windows Communication Foundation. Zájemci se prostřednictvím několika výukových lekcí dozví, jak vytvářet a nastavovat služby WCF. Práce mimo jiné popisuje některé techniky zabezpečení služeb a podává základní pohled do oblasti hostování. Problematika je vysvětlována pomocí názorných příkladů. Součástí je i vzorová audiovizuální prezentace.

Abstract

The main goal of this thesis is to acquaint readers with the modern technology Windows Communication Foundation. People who are interested in it can learn via several tutorial lessons how to create and set WCF services. In addition to it, the work describes some techniques of the security service and gives the basic look into hosting area. The problems are explained through visual aids. The sample of audiovisual presentation is available.

Obsah

Seznam obrázků

Seznam tabulek

Seznam příkladů

1 Úvod

Windows Communication Foundation je jednou z nových perspektivních přístupů pro vzájemnou komunikaci aplikací. Můžeme si pod ním představit snahu Microsoftu o vytvoření jednotného komunikačního modelu sloučením jejich dosavadních distribuovaných technologií. Takového, který by byl na jednu stranu komplexní a rozšiřitelný, s možností široké konfigurace, ale na druhou stranu jednoduchý a lehce pochopitelný. WCF umožňuje psát služby, jak pro výměnu informací mezi Windows systémy, tak mezi jakýmikoli jinými systémy a platformami, které podporují otevřené (např. SOAP) standardy výměny dat. Umožňuje použití nejrůznější škály komunikačních protokolů, nastavení a přizpůsobení si způsobu vaší komunikace. Již od začátku je pak kladen důraz na moderní, servisně-orientovaný přístup programování, na kterém je WCF celý založen.

Cílem této práce je vytvoření výukového materiálu technologie Windows Communication Foundation (WCF). Zájemci získají prostřednictvím ní možnost proniknout od základů až po pokročilé vlastnosti tvorby služeb WCF a to v několika lekcích. Každá lekce obsahuje teoretický základ, doplněný o vzorový příklad, na kterém je názorně ukázáno, jak s danou problematikou pracovat. Právě vysvětlení problému pomocí příkladu je stěžejní metodikou výuky. Považoval jsem za rozumné popisovat kapitoly právě tímto způsobem, neboť si myslím, že čtenář lépe pochopí danou problematiku na praktické ukázce, než prostřednictvím detailnějšího rozboru teorie. Součástí výukového materiálu je i audiovisuální videoukázka pro ještě názornější popsání jednoho z příkladů. Student tak pro přečtení kapitol a zhlédnutí příslušného videa získá poměrně dobré vědomosti v dané oblasti a také odrazový můstek pro své případné další studium.

Ačkoli není příručka určena jen specifickému okruhu programátorů, ale každému zájemci, pro dobré pochopení se předpokládají alespoň nejzákladnější vědomosti v oblasti programování pod platformou .NET (nejlépe C#). Kvůli koncipování na širší okruh zájemců, se práce snaží držet srozumitelného jazyka a postupuje „krok za krokem“ od nejzákladnějších věcí až po pokročilé prvky. Začíná úvodním seznámením s technologií a poté už se samotnými výukovými lekcemi. První polovina obsahuje lekce se základy WCF určené pro začátečníky. Kapitoly tam na sebe navazují a měly by se číst postupně. Vyšší lekce už spolu logicky nesouvisejí a není tedy nutné, je číst přímo za sebou. Jsou určené spíše pro pokročilejší zájemce, nicméně jsou zvládnutelné i pro začátečníky, kteří přečtou první část příručky. Pokročilejší čtenář se pak může sám rozhodnout podle svých dosavadních vědomostí a odhodlání, jaký rys služeb WCF je pro něj důležitý a jaký naopak přeskočí.

2 Technologie WCF

2.1 Distribuované programování dříve a dnes

Ještě dávno před Internetem si lidé pohrávali s myšlenkou, a poté i s realizací, rozložení funkce programu na části, jenž poběží současně na odlišných počítačích, které spolu budou komunikovat pomocí sítě. Tomuto přístupu říkáme distribuované programování. S rozmachem Internetu stoupaly požadavky na tyto aplikace. Trvalo však ještě řadu let, než dosáhla dnešní podoby. Silnými hráči v této oblasti byly i technologie firmy Microsoft. Až do současnosti jsme od této společnosti měli při tvorbě distribuovaných aplikací v .NET 2.0 na výběr jednu z následujících možností:

- ASP.NET Web Services (ASMX)
- Web Services Enhancements (WSE)
- .NET Remoting
- Microsoft Message Queue (MSMQ)
- Enterprise Services/COM+

Každá z těchto technologií má svoje klady a zápory. Např. k ASMX jde přistupovat z různých platforem, na rozdíl od ostatních. Na druhou stranu nepodává tak dobrý výkon jako .NET Remoting a Enterprise Services. Všechny mají odlišné možnosti rozšíření a přizpůsobení, ale nejdůležitějším nedostatkem je jejich vzájemná nekompatibilita, nebo alespoň obtížná slučitelnost. Programátor se tak musí již na začátku rozhodnou, jakou z nich při tvorbě systému použije a té se držet po celý vývoj. V případě změny požadavků, komunikačního kanálu nebo platformy, nedostatečné propustnosti linky a podobně, dojde většinou k velkému zásahu do aplikace a mnohdy to znamená změnu celé distribuované technologie a nutnost začínat zcela od začátku.

Tyto nedostatky si firma Microsoft dobře uvědomovala, a proto v novém Frameworku 3.0 představuje jednotný přístup tvorby distribuovaných aplikací nazvaný Windows Communication Foundation. WCF sjednocuje dosavadní zmíněné technologie pod jednu a odstraňuje jejich jednotlivé nedostatky. Díky flexibilnímu konfiguračnímu modelu je možné snadno nastavit, jakým způsobem bude nová aplikace komunikovat, jaký protokol a způsob zabezpečení bude používat, atd., a to bez nutnosti změny programové části kódu. Můžeme si snadno zvolit, zda nám vyhovuje vyšší výkon, na úkor použití pouze v prostředí Windows, anebo se spokojíme s nižším výkonem, ale potřebujeme přenositelnost a použitelnost v rozličných operačních prostředích. Tuto otázku si už nemusíme pokládat na samém začátku vývoje, jako u

technologií před WCF. ale můžeme změny provést dodatečně, neboť takové úprava bude znamenat jen minimální zásah do aplikace.

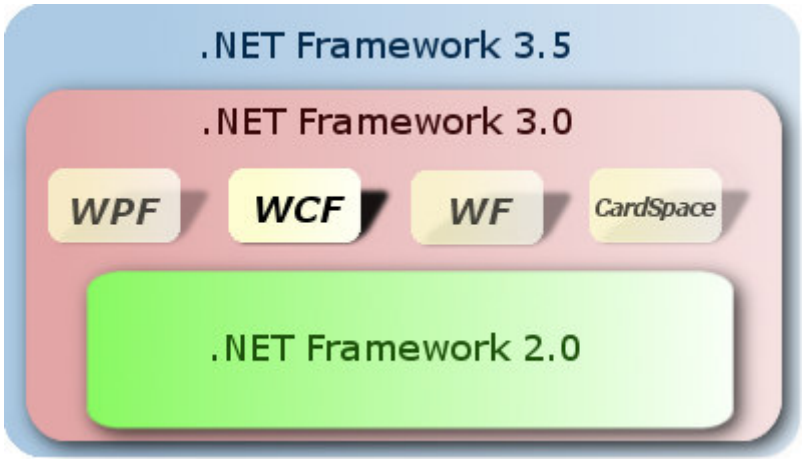
2.2 Začlenění v .NET Frameworku

Technologii Windows Communication Foundation oficiálně uvedla společnost Microsoft v roce 2006 jako jednu ze čtyř novinek vydaného .NET Frameworku 3.0. Jsou to:

- Windows Presentation Foundation (grafické uživatelských rozhraní).
- Windows Communication Foundation (komunikační technologie).
- Windows Workflow Foundation (definování a řízení toků v procesech).
- Windows Card Space (mechanismus pro ověřování identity uživatelů).

.NET Framework je řízený programový model pro Windows, který v budoucnu nahradí Win32 API. V současné době je dostupná nejnovější verze .NET Frameworku 3.5 SP1. Ačkoli pro chod WCF dostačuje verze 3.0, doporučuji doinstalovat tuto. Získáte tím nejen LINQ a nové rysy jazyka C# ve verzi 3.0, ale i určitá rozšíření přímo pro WCF. Jsou to AJAX, JSON, REST, POX, RSS, ATOM, a několik nových WS-* standardů a dalších vylepšení. Navíc v této verzi budete moci používat WCF i v mobilních zařízeních, a to díky .NET Compact Frameworku 3.5, který je jednodušší a okleštěná verze plnohodnotného Frameworku 3.5.

.NET Framework 3.0 je automaticky zahrnut v operačních systémech Windows Vista a Windows Server 2008. Framework 3.5 není v současnosti předinstalován v žádném operačním systému, ale je ho možný doinstalovat do Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008 a novějších.



Obrázek 1: WCF uvnitř .NET Frameworku

2.3 Hlavní rysy

WCF není pouze další z cest pro tvorbu distribuovaných aplikací, ale poskytuje dle ([3], str. 29) oproti svým předchůdcům několik podstatných výhod. Hlavními výhodami jsou:

- Sjednocení předchozích technologií
- Interoperabilita napříč platformami
- Orientace na služby (service-oriented)

2.3.1 Sjednocení předchozích technologií

V dnešní době rozličných podnikových systémů existuje mnoho distribuovaných technologií. Každá má specifickou roli a odlišný význam. Nehledě na to, že každá je založena na odlišných programových modelech. Například když vyvíjíme aplikaci, která komunikuje přes http, a chceme ji převézt do TCP, musíme předělat program. Nebo když tvoříte klasickou webovou službu, nemáme možnost přidat podporu fronty zpráv bez předělání programu. To je nepříjemné pro vývojáře, kteří se pak musí učit odlišná API pro rozdílné způsoby tvorby distribuovaných komponent. Neustálý boj mezi distribuovanými technologiemi vedl k otázce: Proč neexistuje jen jedna technologie pro všechny situace? WCF je možné řešení této otázky.

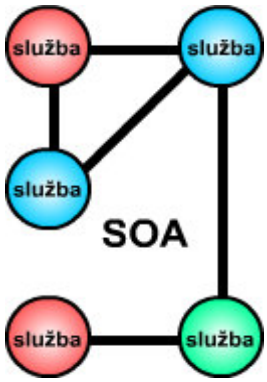
2.3.2 Interoperabilita napříč platformami

Mnoho velkých softwarových společností používá vlastní protokoly, které jsou úzce spojeny s nějakou specifickou platformou. To může být problémem při spolupráci s jiným softwarem na jiných platformách a dokonce i mezi vnitropodnikovými aplikacemi, neboť podnikové systémy se skládají z mnoha rozdílných a často nekompatibilních částí.

Proto je velmi důležitá vzájemná interoperabilita. Naštěstí velké softwarové společnosti vzájemně vytvořily Web Services Interoperability organization (WS-I), která vyvinula množství standardů pro vzájemnou komunikaci softwaru napříč odlišnými platformami. Je pak jen na nás, jaký standart implementujeme, podle toho, jaký potřebujeme (např. WS-Security pro zabezpečení). WCF rozumí těmto specifikacím a je tedy připravena pro interoperabilitu mezi platformami. WCF služby používají SOAP protokol, což je otevřený standard výměny dat mezi odlišnými technologiemi a operačními systémy. Takže .NET aplikace na Windows může bez problému komunikovat s Java aplikací v Linuxu, aniž by znala detaily implementace té druhé. Stačí, když obě rozumí otevřeným standardům výměny dat.

2.3.3 Orientace na služby (service-oriented)

WCF je kompletně navrhnut v souladu se service-oriented architekturou (SOA). SOA definuje nejlepší praktiky pro budování moderních distribuovaných aplikací. Nejedná se o nový koncept, ale záležitost už zhruba 8 let starou. Vznikla jako odpověď na Object-Oriented a Component-Based architektury. Jednoduše řečeno, jedná se o způsob návrhu distribuovaných systémů, kde několik autonomních služeb pracuje ve vzájemném spojení pomocí posíláním zpráv napříč hranicemi. Používají přitom dobře známá rozhraní.



Obrázek 2: SOA

Již při návrhu WCF se brali v úvahu 4 základní zásady SOA:

1. **Služba má explicitní hranice**

Každá služba je oddělena hranicemi, jako je umístění a technologie. Svojí funkcionalitu pak vyjadřuje pomocí dobře známých rozhraní, např. popisuje každý datový člen, jejich parametry a návratové hodnoty. Jediný způsob, jak je možné komunikovat s WCF službou, je skrze tato rozhraní. Všechny ostatní implementační rysy jsou před externími zájemci skryty.

2. **Služby jsou autonomní**

Služby nepotřebují pro svoji existenci nic od klientů nebo ostatních služeb, ale mohou existovat nezávisle. Autonomní služby si můžeme představit jako izolované ostrůvky. Uvnitř se sami rozvíjejí a mohou být pozměněny, ovšem jejich kontrakt a hranice musí být zachovány. Je to podobné, jako při programování rozhraní. Jakmile je jednou rozhraní služby zveřejněné, už by se nemělo měnit, neboť tím riskujeme porušení soudržnosti pro dosavadní klienty. Co můžeme udělat, je měnit implementaci, když zůstane rozhraní zachováno. Pokud ale potřebujeme funkcionalitu WCF služby rozšířit, tak jednoduše přidáme nová rozhraní pro novou funkcionalitu.

3. **Služby komunikují přes kontrakt, ne implementaci**

Všechno, co služba zveřejní, musí být 100 procentně technologicky nezávislé a popsané v kontraktu. Služba musí počítat s tím, že klient a jiné služby, se kterými komunikuje, neví nic o jejích implementačních detailech. Musí být schopna převést svoje nativní i vlastní datové typy do neutrální reprezentace a popsat v kontraktu. Služba tedy neposílá data a typy, ale komunikuje přes jasně definovaná veřejná rozhraní (service kontrakt) pomocí datového schématu (data kontrakt).

4. **Služby jsou kompatibilní založené na politice**

Politika definuje chování služby a způsob, jak s ní mohou klienti komunikovat. To co je definováno v politice (jako je transakčnost, bezpečnost, spolehlivost doručení a podobně), by mělo být odděleno od implementačních detailů služby. Ne všechny služby mohou komunikovat se všemi klienty, což je zcela správné. Politika je tím souborem pravidel, která určují jejich vzájemnou kompatibilitu. Díky ní také můžeme přemístit služby z jednoho prostředí do druhého, aniž by se změnilo chování služby.

2.3.4 Programování a nastavení

WCF umožňuje 2 způsoby konfigurací.

- **Administrativní** nastavení v konfiguračním souboru. To je soubor s příponou config. Jeho název závisí na hostujícím prostředí. Web.config pro IIS hosting, App.config pro self hosting. Nastavení v konfiguračním souboru je doporučené a má tyto výhody oproti programovému (imperativnímu) přístupu. Protože Config soubor je klasický textový XML soubor, umožňuje měnit konfiguraci bez nutnosti speciálního editoru a re-kompilace kódu a to i obyčejným administrátorům.

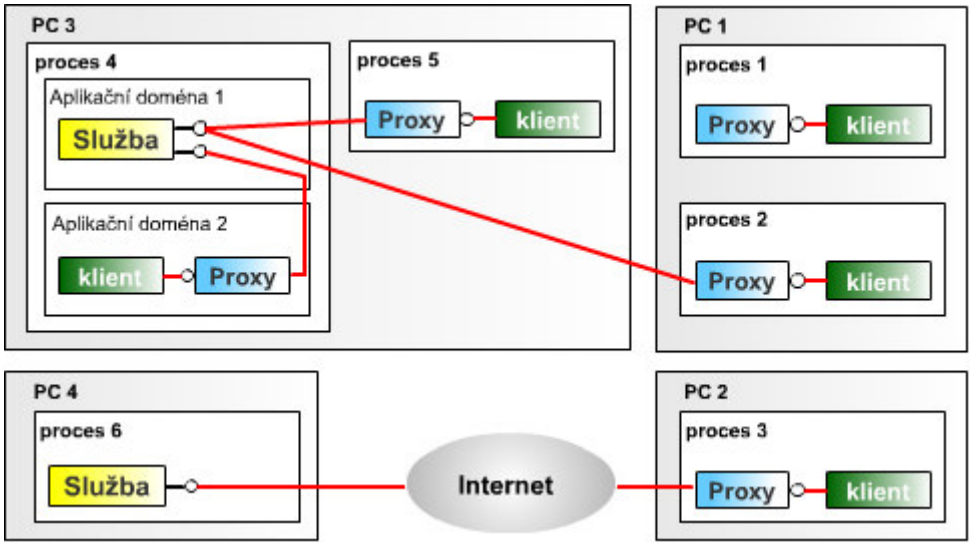
- **Programová (imperativní)** konfigurace v kódu programu. V souborech s příponou cs (C#) nebo vb (Visual Basic). Imperativní způsob není příliš pružný jako administrativní, po každé změně je třeba kompilace a změny může provádět pouze programátor. Na druhou stranu je při kompilaci prováděna syntaktická kontrola, která u deklarativního nastavení není a chyby se tam projevují až při běhu programu. Další výhodou je plná kontrola nad nastavením a implementací, umožňující techniky, které v deklarativním přístupu nejsou možné – ve většině případů ale nejsou ani potřeba.

- **Kombinace** deklarativního a imperativního přístupu. Tento hybridní přístup je uveden pouze z důvodu úplnosti. Je samozřejmý a volíme ho intuitivně podle potřeby. Část nastavení je převzata z konfiguračního souboru a část z kódu.

2.4 Služby WCF

WCF je založen na službách (services) což jsou softwarové entity, které nabízí funkčnost svému okolí a splňují 4 hlavní zásady SOA (viz 2.3.3). Pro programátora je služba vlastně aplikace, která čeká na požadavky klientů a kontroluje komunikaci.

Služby komunikují s klienty nebo mezi sebou prostřednictvím SOAP zpráv, které jsou nezávislé na přenosovém protokolu. To umožňuje WCF službám spolupracovat s ne-WCF klienty a naopak. Služby s klienty nikdy nekomunikují přímo a to ani, když se jedná o 2 lokální entity na stejném počítači (in-memory services), ale vždy komunikují prostřednictví proxy. Proxy poskytuje stejné operace jako služba + některé další. Výhodou proxy je fakt, že služba komunikuje vždy stejným způsobem a to jak s klientem nebo jinou službou na lokálním PC nebo v internetu.

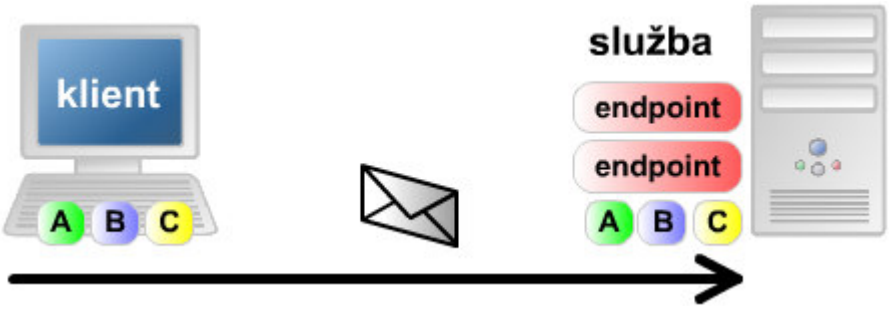


Obrázek 3: Různé možnosti umístění služeb a klientů

2.4.1 Endpoint

Každá služba má alespoň jeden nebo i více endpointů (endpoints). Endpoint je vždy tvořen třemi základními prvky, známé jako ABC (Address, Binding, Contract) :

- **Address:** Kde se služba nachází .
- **Binding:** Jak služba komunikuje.
- **Contract:** Co služba dělá.



Obrázek 4: Endpoint služby v komunikaci s klientem

Endpoint popisuje standardizovaným způsobem, kam se zprávy budou posílat, jak se budou posílat a jak budou vypadat. Služba může vystavit tyto informace jako metadata, které klientská aplikace zpracuje a vygeneruje patřičného WCF klienta.

Endpoint je možné konfigurovat buď administrativně v konfiguračním souboru, nebo programově v kódu. K jednotlivým složkám endpointu se ještě dostaneme.

```
<system.serviceModel>
  <services>
    <service name = "UkazkaWCF.MojeSluzba">
      <endpoint
        address  = "http://localhost:8000/PrvniSluzba/"
        binding  = "wsHttpBinding"
        contract = "UkazkaWCF.IMojeSluzba" />
      <endpoint
        address  = "net.tcp://localhost:8001/DalsiSluzba/"
        binding  = "netTcpBinding"
        contract = "UkazkaWCF.IMojeSluzba" />
    </service>
  </services>
</system.serviceModel>
```

Příklad 1: Konfigurace endpointů služby - administrativně

```
ServiceHost host = new ServiceHost(typeof(MojeSluzba));
Binding wsBinding = new WSHttpBinding();
Binding tcpBinding = new NetTcpBinding();
host.AddServiceEndpoint(typeof(IMojeSluzba), wsBinding,
"http://localhost:8000/MyService");
host.AddServiceEndpoint(typeof(IMojeSluzba), tcpBinding,
"net.tcp://localhost:8001/MyService");
host.Open();
```

Příklad 2: Konfigurace endpointů služby - programově

2.4.2 Contract

2.4.2.1 Service Contract

Říká nám, co služba dělá a co od ní můžeme očekávat. Specifikuje signaturu služby, neboli jaké operace poskytuje. Například služba telefonní společnosti by mohla mít 2 operace. První by vracela telefonní číslo na základě poskytnutého jména a příjmení a druhá by naopak klientům vracela jméno a příjmení na základě telefonního čísla. Nejčastěji ho definujeme vytvořením rozhraní, které označíme ServiceContractAttribute atributem. A metody, které chceme zahrnout do contractu, označíme OperationContractAttribute atributem. Vlastní kód služby pak implementuje toto rozhraní.

```
[ServiceContract]
public interface IKnizniObchod
{
    [OperationContract]
    void zaregistrujZakaznika(Osoba zakaznik);
    [OperationContract]
    string najdiKnihu(string nazev);
    [OperationContract]
    string najdiKnihu(string ISBN);
    [OperationContract]
    bool zpracujObjednavku(KnizniObjednavka objednavka);

}
```

Příklad 3: Ukázka Service Contract

2.4.2.2 Data Contract

Datové typy, které služba používá, musí být popsány v metadatech. Toho dosáhneme použitím data kontraktu. WCF definuje jednoduché typy (jako int, string, atd.) automaticky a nemusíme u nich explicitně používat data contract. Vlastní objekty zahrneme do data Contractu tak, že třídu reprezentující datový typ, označíme atributem DataContractAttribute a datové členy, které chceme zahrnout do contractu, označíme atributy DataMemeberAttribute ze jmenného prostoru System.Runtime.Serialization.

```
[DataContract]
public class Osoba
{
    [DataMember]
    public string Jmeno;
    [DataMember]
    public string Prijmeni;
    [DataMember]
    public int Vek;
}
```

Příklad 4: Ukázka Data Contract

2.4.2.3 Message Contract

Poskytuje neomezenou kontrolu nad SOAP zprávami posílanými a přijímanými službou. Umožňuje upravit parametry, jako je její hlava a tělo. Můžete nastavit například jaká úroveň zabezpečení má být použita na jaký element zprávy a podobně. Pro většinu případů ale postačuje kontrolovat pouze datové typy, prostřednictvím Data Contractu a vše ostatní nechat na WCF. Pokud přesto z nějakého důvodu chcete upravovat samotné SOAP zprávy, můžete to udělat tak, že typ zprávy označíte MessageContractAttribute atributem, hlavu zprávy MessageHeaderAttribute atributem a prvky zprávy MessageBodyMemberAttribute atributy. V příkladu pro zjednodušení neuvádím zapouzdření do vlastností jako u Data Contractu.

```
[MessageContract]
public class KnizniObjednavka
{
    [MessageHeader]
    public string ISBN;
    [MessageBodyMember]
    public int Mnozstvi;
    [MessageBodyMember]
    public string Jmeno;
    [MessageBodyMember]
    public string Prijmeni;
}
```

Příklad 5: Ukázka Message Contract

2.4.2.4 Fault Contract

Definuje, jaké chyby mohou být předávány službou. Fault Contract je přiřazen operaci služby k označení chyb, které mohou být vráceny volajícímu. Tyto chyby jsou takzvané SOAP faulty, které jsou .NET Frameworkem mapovány na klasické výjimky. Fault Contract definujeme pomocí FaultContractAttribute atributu aplikovaného na (Operation Contract) metodu, která výjimku může vyvolat. Je možné předávat jako výjimky klasické .Net výjimky nebo vlastní typy, ty je ovšem nutné zahrnout do DataContractu, aby byly serializované a mohly být poslány v SOAP faultu.

```
[DataContract]
public class KnihaFaul
{
    [DataMember]
    public string NazevKnihy
    [DataMember]
    public string Problem
    [DataMember]
    public string NavrhovaneReseni
}
```

Příklad 6: Ukázka definování objektu pro Fault Contract

```
public interface IKnizniObchod
{
    [OperationContract]
    [FaultContract(typeof(InvalidOperationException))]
    void zaregistrujZakaznika(Osoba zakaznika);
    [OperationContract]
    [FaultContract(typeof(KnihaFaul))]
    string NajdiKnihu(string Nazev);
    [OperationContract]
    [FaultContract(typeof(KnihaFaul))]
    string NajdiKnihu(string ISBN);
    [OperationContract]
    bool ZpracujObjednavku(KnizniObjednavka objednavka);
}
```

Příklad 7: Ukázka Fault Contract

2.4.2.5 Vnitřní reprezentace Contractů

Aby mohl WCF spolupracovat i s ne-WCF klienty a naopak, je třeba popsat službu a předávat data v univerzálních formátech. Přestože jsou všechny Contrakty definovány v .NET aplikaci jako CLR typy, kterým ne-WCF aplikace nerozumí, jsou v kódu označeny příslušným atributem (např `[ServiceContract]`). Díky těmto atributům .NET pozná, že se mají vnitřně v metadatech reprezentovat jako univerzální XML data ve WSDL,SOAP, nebo XSD formátu.

- **Service Contract:** Je mapován na Web Services Description Language (WSDL).
- **Data Contract:** Je mapován na XML Schema Definition (XSD)
- **Message Contract:** Je mapován na Simple Object Access Protocol (SOAP) Messages.
- **Fault Contract:** Je mapován na SOAP Faults.

2.4.3 Binding

Binding určuje způsob, jak bude služba komunikovat. Binding se liší podle možností přenosového protokolu (HTTP, MSMQ,TCP,…), kódování (text, binary, MTOM…), způsobu zabezpečení, spolehlivosti doručení, podpory transakcí, duplexu, WS-* protokolů (WS-Security, WS-Federation, WS-Reliability,WS-Transactions) a podobně.

Existuje několik standardních systémových Binding, implementovaných přímo.NET frameworkem:

- **BasicHttpBinding** : Pro komunikaci s webovými službami, splňující WS-Basic specifikaci. To jsou například ASP.NET Webové služby (ASMX). Binding používá http pro transport a text/XML kódování.
- **WSHttpBinding** : Zabezpečený a interoperabilní Binding bez podpory duplex kontraktů.
  WSDualHttpBinding : Zabezpečený a interoperabilní Binding s podporou duplex kontraktů nebo komunikace přes SOAP prostředníka.
- **WSFederationHttpBinding** : Zabezpečený a interoperabilní Binding s podporou WS-Federation protokolu. Tedy podporuje autentizaci a autorizaci uživatelů.
- **NetTcpBinding** : Zabezpečený a optimalizovaný Binding pro komunikaci mezi WCF aplikacemi.
- **NetPeerTcpBinding** : Binding podporuje zabezpečení a více počítačovou komunikaci.
- **NetNamedPipeBinding** : Zabezpečený spolehlivý a optimalizovaný Binding pro komunikaci WCF aplikací na jednom počítači.
- **NetMsmqBinding** : Binding s podporou front MSMQ.
- **MsmqIntegrationBinding** : Binding pro komunikaci mezi WCF aplikací a existující MSMQ aplikací.

Binding [schema]	Transport	Encoding	Interoperable	Security	Session	Transactions	Duplex
BasicHttpBinding [http, https]	HTTP/HTTPS	Text	BP 1.1	N|T|M|X			
WsHttpBinding [http, https]	HTTP/HTTPS	Text, MTOM	WS	N|T|M|X	N|T|R	X	
WsDualHttpBinding [http]	HTTP	Text, MTOM	WS	N|M	R	X	X
WSFederationHttpBinding [http, https]	HTTP/HTTPS	Text, MTOM	WS-federat.	N|M|X	N|R	X	
NetTcpBinding [net.tcp]	TCP	Binary	.NET	N|T|M|X	T|R	X	X
NetPeerTcpBinding [net.p2p]	P2P	Binary	Peer	N|T|M|X			X
NetNamedPipesBinding [net.pipe]	IPC	Binary	.NET	N|T	N|T	X	X
NetMsmqBinding [net.msmq]	MSMQ	Binary	.NET	N|T|M|B		X	
MsmqIntegrationBinding	MSMQ	*	MSMQ	N|T		X	

Tabulka 1: Systémové Binding

(upraveno z [13])

Pro většinu případů plně dostačuje zvolit jeden z předdefinovaných systémových Binding a poupravit si jeho vlastnosti. Můžeme změnit například výchozí kódování, zabezpečení a podobně, to uděláme jednoduše buď v kódu, upravením jeho vlastností anebo v konfiguraci, přidáním do Binding elementu patřičné parametry. Každý výchozí Binding má svoje specifické parametry. Jejich výčet vlastností přesahuje rámec této práce.

Někdy se může stát, že nám předdefinovaný Binding nebude dostačovat ani po upravení. Potom si můžeme vytvořit vlastní Binding a to 3 způsoby.

- Odvozením z některého ze standardních systémových Binding.
- Odvozením od třídy System.ServiceModel.Channels.CustomBinding.
- Odvozením od třídy System.ServiceModel.Channels.Binding

```
<system.serviceModel>
  <bindings>
    <wsHttpBinding>
      <binding name = "wshttpbind"
      messageEncoding= "Mtom"
      textencoding = "utf-8"
      receiveTimeout = "00:00:30"
      transactionFlow= "true">
      </binding>
    </wsHttpBinding>
    <netTCPBinding>
      <binding name = "nettcpbind"
      portsharingenabled = "true"
      listenbacklog = "10"
      closeTimeout = "00:00:30"
      <security mode="Message">>
      </binding>
    </netTCPBinding>
  </bindings>
</system.ServiceModel>
```

Příklad 8: Konfigurace Binding

2.4.4 Address

Jak již zaznělo dříve, každý endpoint služby potřebuje Address (adresu). Adresa určuje, kde se endpoint služby nachází. Jedná se vlastně o jedinečné URI endpointu. Díky tomu klienti na této adrese endpoint služby najdou a identifikují.

Adresa ve WCF se skládá z transportního protokolu nebo schématu, jména počítače, kde služba běží a cesty. Volitelný je port, který je závislý na protokolu. WCF adresa je podobná URI, jaké známe z webových stránek:

protokol://<jmenoPocitace>[:port]/cesta1/cesta2/…

WCF podporuje několik transportních schémat, každý z nich má trochu odlišnou strukturu:

- HTTP
- TCP
- Peer network
- IPC (Inter-Process Communication over named pipes)
- MSMQ

2.4.4.1 HTTP

Asi nejpoužívanější adresa je HTTP. Pokud neuvedete port je defaultně 80. Kromě HTTP je možné použít i zabezpečenou variantu HTTPS. Tvar HTTP/HTTPS je následovný

```
http|https://doménové jméno|jméno počítače [:port]/cesta
http://localhost:8080/mojeSluzba
http://ahoj.cz/nazdar
https://153.111.32.2/zabezpeceneSluzby/Sluzba5
```

2.4.4.2 TCP

TCP adresy používají net.tcp schéma.

```
net.tcp://localhost:8002/MojeSluzba
net.tcp://localhost:8002/JinaSluzba
net.tcp://localhost:8003/JinaSluzba
```

2.4.4.3 Peer network

Peer network používá pro transport Peer-to-peer síť. Její tvar může vypadat jako na následující ukázce.

```
net.p2p://MujMeshIdentifikator/MojeEndpointJmeno
```

2.4.4.4 IPC

IPC adresy používají net.pipe schéma. Net.pipe ve WCF slouží pouze k lokální komunikaci mezi procesy na jednom počítači. Nemůže být tedy použit pro komunikaci služeb na jiných počítačích. Net.pipe schéma nepoužívá porty.

```
net.pipe://localhost/MojeSluzba1
net.pipe://localhost/MojeSluzba2
```

2.4.4.5 MSMQ

Microsoft Message Queue (MSMQ) používá net.msmq schéma, které je poněkud odlišné od ostatních. MSMQ má veřejné (public) a soukromé

(private) fronty. Veřejné fronty mohou být dostupné vzdáleně skrze Active Directory, kdežto soukromé fronty jsou dostupné pouze na lokálním počítači. MSMQ nepoužívá čísla portů. Její tvar je následovný.

```
net.msmq://hostname / [private] / queue-name
net.msmq://localhost/private/SoukromaSluzba
net.msmq://ahoj.cz/VerejnaSluzba
```

2.4.5 Metadata

Metadata v základě standardizovaným způsobem popisují, jak může spolupracovat okolní svět se službou a jaký je Contract služby. Aby byla metadata služby čitelná i pro ne Microsoftské platformy, jsou metadata v XML formátu (skládají se z více standardů: WSDL, XSD, atd.). Vývojáři z mnoha prostředí, jazyků a platforem mohou použít tyto veřejné Contract informace k vytvoření aplikace (klienta), která umí komunikovat se službou.

WSDL, XSD, SOAP, WS standardy, atd. jsou komplikované jazyky pro programování přímo v nich. Naštěstí WCF převádí managed .NET typy a nastavení služby do těchto jazyků za nás v pozadí. Jediné co je třeba udělat je označit příslušné třídy a metody v contractu potřebnými atributy a povolit generování metadat. Defaultně jsou totiž z bezpečnostních důvodů metadata vypnuta. Proto, když je chceme publikovat, musíme manuálně nakonfigurovat službu. Existují 2 způsoby, jak to udělat.

2.4.5.1 HTTP-GET

Klienti mohou získat metadata služby použitím HTTP/GET dotazu. WCF poskytuje publikaci přes http-get automaticky, povoluje se přes nastavení httpGetEnabled = "true". Služba pak bude publikovat metadata ze svojí **base Address**+**?wsdl** (base Address je tedy `http://localhost:8000/` a adresa metadat bude `http://localhost:8000/?wsdl`).

2.4.5.2 Metadata Exchange Endpoint

Metadata může služba publikovat též ze speciálního endpointu nazývaného MEX (Metadata Exchange Endpoint). Obvykle máme jeden pro celou službu, ale můžeme jich mít více, třeba jeden pro každý endpoint. Tak jako každý jiný endpoint i MEX se skládá z ABC.

MEX Address

Za MEX adresu můžeme zvolit jakoukoli adresu, musíme jen dodržet kompatibilitu s Binding. Obvykle se přidává prefix /mex, například `net.tcp://localhost/MojeSluzba/mex`.

MEX Binding

Máme na výběr 4 Binding. mexHttpBinding pro HTTP publikaci, mexHttpsBinding pro HTTPS publikaci, mexNamedPipeBinding pro named pipe publikaci, mexTcpBinding pro TCP publikaci.

MEX Contract

WCF automaticky poskytuje IMetadataExchange rozhraní a jeho implementaci. Takže se nemusíme starat o to, jak MEX Contract funguje, stačí zvolit Address a Binding.

```
<system.serviceModel>
 <services>
   <service name = "MojeSluzba" >
     <endpoint address=""
               binding="basicHttpBinding"
               contract="IMojeSluzba" />
     <endpoint address="http://localhost:8001/Sluzba/mex"
               binding="mexHttpBinding"
               contract="IMetadataExchange" />
   </service>
 </services>
 <behaviors>
   <serviceBehaviors>
     <behavior name = "metadata">
       <serviceMetadata httpGetEnabled = "true"/>
     </behavior>
   </serviceBehaviors>
 </behaviors>
</system.serviceModel>
```

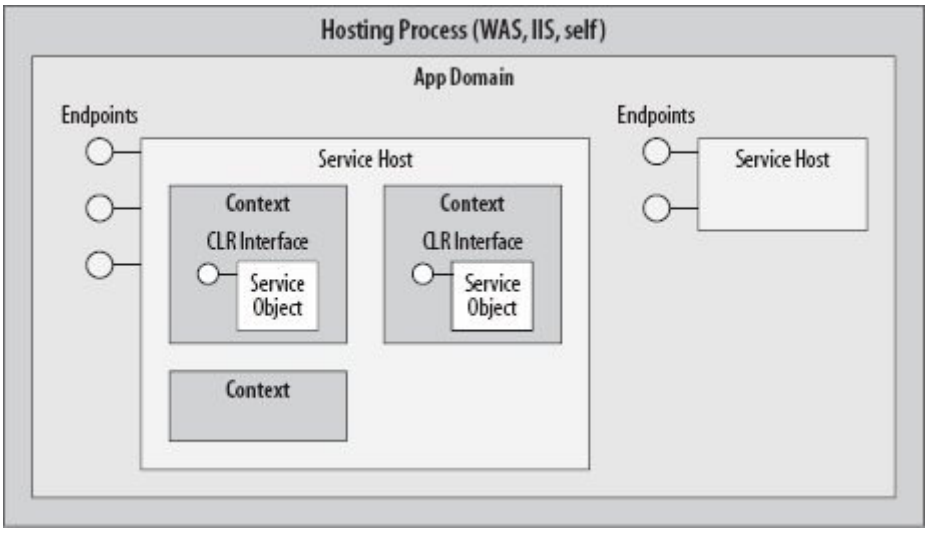
Příklad 9: MEX endpoint a HTTP-GET

2.5 Hostování služeb

Každá .NET třída musí existovat uvnitř nějakého windows procesu. WCF služba není výjimkou. Proces, kde je WCF služba hostována, nazýváme host process (volně přeloženo hostující proces). Jediný hostující proces může hostovat více služeb a naopak jedna a ta samá služba může být hostována ve více hostujících procesech. Hostující proces může být poskytnut několika způsoby: Prostřednictvím IIS, Widows Activation Service (WAS) anebo jako self-hosting.

2.5.1 Host architektura

Popis toho jak funguje host architektura je pěkně vysvětleno v ([2] kap. 1.11). K tomu, aby byl spuštěn kód služby v hostujícím procesu, jsou vytvořeny assembly. Assembly nejsou hostovány přímo v hostujícím procesu. Místo toho common language runtime (CLR) isoluje řízený kód služby vytvořením oddělených logických částí, nazývaných application domain (aplikační domény). Při vytvoření procesu CLR automaticky vytvoří jednu defaultní aplikační doménu. Každý proces hostuje alespoň tuto jednu aplikační doménu, ale může hostovat (a často i hostuje) více aplikačních domén. Defaultní aplikační doména má životnost shodnou s procesem a zůstává, dokud proces „žije“. Ve většině případů, ale v defaultní aplikační doméně není žádný kód. Místo toho proces vytvoří novou aplikační doménu, která pak může být vypnuta nezávisle na procesu. Jako příklad více aplikačních domén v jednom procesu může být projekt v jednom procesu se serverovskou a klientskou částí, každá část ve vlastní aplikační doméně. Taková aplikace bude bez problému fungovat, protože aplikační domény fungují v rámci procesu nezávisle a isolovaně.



Obrázek 5: Host architektura
(převzato z [2], kap. 1.11)

To ale ještě není vše. Každá aplikační doména, může mít více nebo nemusí mít žádný System.ServiceModel.ServiceHost instanci. Se ServiceHost jsme se již setkali, jedná se o důležitou třídu, která hostuje a kontroluje životní cyklus služby. Pro každý typ služby je vyhrazena jedna ServiceHost instance. Aby to nebylo málo, tak každá ServiceHost instance má nula nebo více Context (kontextů). Kontext může být prázdný nebo konečně asociován se samotnou instancí služby.

2.5.2 Porovnání druhů hostingů

Podíváme se na vlastnosti jednotlivých hostingů.

- **Microsoft Internet Information Server (IIS):** Přináší řadu výhod. Vytvoří hostující proces po prvním požadavku klienta a automaticky řídí jeho životního cyklus. IIS monitoruje, v jakém stavu je proces služby, a podle toho jedná (například, když dlouho neodpovídá, tak ho recykluje). Bohužel podstatnou nevýhodou IIS je, že podporuje pouze http protokol.

- **Windows Activation Service (WAS):** Je dostupný na systémech Windows Vista, není ho možné použít na starších systémech. Je součástí IIS7, ale je možný instalovat a konfigurovat odděleně. WAS přebírá mnoho výhod IIS-hostingu, jako je aplikační pooling, recycling, identity management, isolation a podobně. Největší výhodou je možnost použití všech protokolů.

- **Self hosting:** Je způsob hostingu, kdy za životní cyklus hostujícího procesu služby je zodpovědný kód, který napíše samotný vývojář. Můžete volit jakýkoliv Windows proces, například konzolovou aplikaci, WinForm aplikaci, WPF aplikaci, službu systému Windows atd.

2.6 Klienti WCF

WCF klient je aplikace, která využívá funkcionality služby. Může se jednat o jakoukoliv aplikaci např. Windows Forms, WPF, ASP.NET nebo konzolovou aplikaci. Aplikace komunikuje se službou skrze endpoint. Aby to fungovalo, musí klient znát základní informace o službě, jako je adresa, na které endpoint naslouchá, Binding a Contract služby.

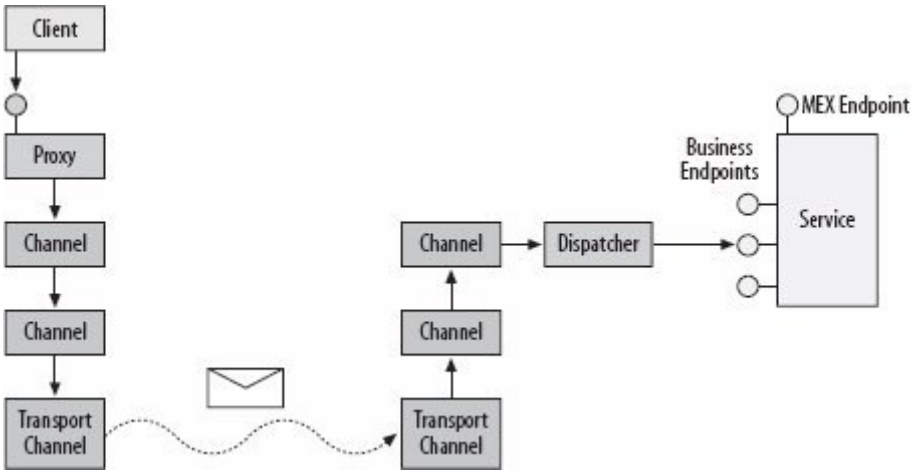
2.6.1 Architektura klienta

WCF poskytuje bohaté API pro klienty v System.ServiceModel assembly. Z toho vyplývá, že můžeme programovat přímo v tomto API anebo použít nástroj svcutil pro generování proxy třídy a konfiguračního souboru. Proxy je generovaný Service Contract služby a některé další metody navíc pro kontrolu spojení. Poskytuje funkce, které umožní klientům přeměnit volání metod na odchozí zprávy ke službě a naopak, příchozí zprávy ze služby naopak přeměňuje na informace, které klient normálně používá jako návratové hodnoty a parametry ve svých metodách.

Dalšími důležitými aspekty WCF klienta je IClientChannel rozhraní. Toto rozhraní sdružuje operace, které dovolují vývojářům kontrolovat funkcionalitu, jako je zavírání relací klienta a dispose kanálu. Nezapomeňme na client channel (kanál), založený na konfiguračních nastaveních a specifikovaných použitým Bindingem.

2.6.2 Komunikace klienta a služby

Na obrázku 6 je příklad komunikace klienta se službou z ([2], kap. 1.11). Komunikace začíná na straně Klienta tím, že Proxy serializuje volání metody Klienta na zprávu a pošle ji skupinou kanálů (channel). Kanál si můžeme představit jako zachytávač zprávy, jehož úkolem je provést nějakou činnost. Každý kanál na klientské straně provádí předúpravu zprávy, než bude poslána dále. Struktura a složení kanálů závisí většinou na B**i**nding. Například jeden kanál může být zodpovědný za kódování služby (text, bingy, …), jiný za zabezpečení, další za přenos transakcí, jiné za relace, šifrování atd.. Poslední kanál na straně Klienta je transportní kanál, který pošle zprávu na stranu hosta.



Obrázek 6: Klient – Service komunikace
(převzato z [2], kap. 1.11)

U hosta je zpráva obdobně předupravená. Nejprve ji přijme transportní kanál a dále prochází kanály jako u klienta. Dojde tedy k různým operacím, jako je dešifrování zprávy, dekódování zprávy, přenos transakce, nastavení zabezpečení, řízení relace a aktivování instance služby. Posledním kanálem na straně hosta je dispatcher. Dispatcher konvertuje zprávu na volání metod instance služby.

Služba (jak můžeme vydedukovat) neví, jestli byla volána lokálním klientem nebo někým z internetu. Ve skutečnosti je vždycky volána lokálně dispatcherem. To je důležitý fakt. Je tedy úplně jedno, jestli je klient lokální nebo vzdálený, dokonce, i když je ve stejném procesu nebo na druhém konci Světa, vždy se služba bude chovat úplně stejně.

Vraťme se ke komunikaci. Instance služby provede volání a předá řízení dispatcheru, který převede návratové hodnoty a chyby (jsou li) na „návratovou zprávu“. Proces je nyní opačný. Dispatcher přepošle zprávu skrze kanály, které provedou řízení transakcí, deaktivaci instance služby, zakódování a zašifrování

odpovědi a tak dále. Zpráva prochází nakonec přes transportní kanál, který ji pošle na klientskou stranu. U klienta zpráva projde přes transportní kanál a další kanály, které provedou dešifrování, dekódování, přijmutí nebo odmítnutí transakce atd.. Poslední kanál je proxy, které přemění zprávu například na návratové hodnoty nebo výjimky a předá řízení klientovi.

3 Výukový materiál WCF

3.1 Analýza situace

Technologie WCF mě zajímá, a když jsem se jí sám učil, musel jsem se prodírat množstvím neucelených informací a zdrojů. Nevýhodou bylo, že se mi výsledný obraz formoval velmi pomalu a nijak komplexně, spíš formou skládačky. Později jsem si obstaral některé zdařilé publikace. V nich již bylo časové vodítko a výuka z nich byla mnohem pohodlnější, než z internetových zdrojů. Je třeba ale podotknout, že publikace, o kterých mluvím, byly v anglickém jazyce. V českém jazyce je situace mnohem horší. V současné době existuje několik málo internetových zdrojů (například poměrně zdařilý seriál na vyvojar.cz), ale nejsem si vědom žádné tištěné nebo komplexnější publikace na toto téma. Navíc tyto internetové články jsou často okomentovanými zdrojovými kódy s trochou teorie. Pro začátečníka tam chybí postupný návod, jak a proč bylo kódu dosaženo. Pro zkušeného uživatele jsou zase tyto zdroje příliš povrchní a často postrádají pokročilá témata.

3.2 Cíle a řešení

Podle mého názoru tedy není situace, v oblasti českých výukových materiálů WCF, v současnosti příliš dobrá. Rozhodl jsem se proto, že se pokusím alespoň částečně vyplnit prázdné místo a vytvořím výukový materiál, který by pomohl jak začátečníkovi, tak snad i pokročilejšímu čtenáři, proniknout do tajů WCF. Také bych rád podotknul, že není mým cílem vytvořit komplexní a plnohodnotnou publikace na toto téma, jako jsou zmiňované anglické, a ani jsem nezamýšlel přijít s referenční příručkou této technologie. Jelikož jsem omezen časově, nemám v úmyslu ani pojmout všechny nebo alespoň většinu rysů WCF.

Vše, čeho chci dosáhnout, je pomoci českým zájemcům rychle proniknout do oblasti WCF, a to jednoduše bez nutnosti prohledávání internetových zdrojů a kupování zahraničních knih. Kladu si za cíl, aby práci porozuměli i naprostí začátečníci ve WCF, kteří jsou schopni pouze práce s počítačem a ovládají základy programování v .NET. Chci jim pomoci se rychle orientovat v problematice. Těm, kteří budou chtít WCF studovat detailněji, bych rád podal takový základ, na kterém budou moci stavět ve svém dalším studiu. Byl bych samozřejmě rád, aby práce pomohla, alespoň v některých aspektech technologie i pokročilejším zájemcům. Výukový materiál ale nemá a ani nemůže, jak jsem již zmiňoval, nahradit plnohodnotné publikace, které jsou na úplně odlišné kvantitativní, ale i kvalitativní rovině. Na druhou stranu mě potěší každý, komu práce pomůže rychle najít a pochopit, co potřebuje.

3.3 Metodika

Správná volba metodiky je stěžejní pro každou práci. Přemýšlel jsem, jak nejlépe postupovat při tvorbě výukového materiálu, aby byl pro čtenáře co nejpřínosnější. Nakonec jsem zvolil způsob, který nejvíce vyhovuje při čtení mě. Jako hlavní část práce jsem vytvořil příručku obsahující výukové lekce a vzorové příklady. Některé výukové lekce jsem ještě navíc doplnil i o pomocné videoukázky.

3.3.1 Vysvětlení problematiky na příkladu

Pro každou lekci příručky jsem vytvořil vzorový příklad. Tento příklad jsem většinou vymyslel sám pro potřeby dané lekce anebo jsem se inspiroval z příkladů v zahraničních WCF publikací. Postup vytvoření příkladu, nebo ve většině případů spíše popsání jeho nejdůležitějších částí, následuje vždy hned za teorii ve výukové lekci. Lekce se tedy skládají z krátkého teoretického základu a z popisu vzorového příkladu. Důležité je, že popis vzorového příkladu není jen okopírovaný kus kódu, ale návod, jak k němu dospět. Jinými slovy, podávám tam bod po bodu postup k dosažení cíle. V jednotlivých bodech vysvětluji funkci a důležitá místa příslušného kódu, nebo postupu. Na tomto přístupu je dobré, že zájemce vidí, v jakém pořadí program vznikal a proč, což jen samotný zdrojový kód s komentáři nikdy nemůže ukázat. Domnívám se, že právě jednotlivé příklady s jejich vysvětlením, je hlavní přínos práce. Také si myslím, že některé aspekty nejdou dobře pochopit ani za použití tohoto „bod po bodu postupu“. Někdy je potřeba vidět, jak program funguje v praxi. Do své práce jsem proto, kromě množství obrázků, zakomponoval i audiovisuální prezentaci, jako podpůrný (ale možná i klíčový) prostředek k rychlému pochopení problematiky jedné lekce. Vytvořil jsem videokurz za použití volně dostupného programu CamStudio, který umožňuje snímat plochu obrazovky a zvuk. Mohl jsem tak zájemcům ukázat, jak program ve skutečnosti funguje i některé vlastnosti, které bych jinak musel složitě opisovat. Hlavní přínos pro diváka je, že vidí přesně, co udělat a jakým způsobem, aby dosáhl stejného výsledku.

3.3.2 Členění příručky

Co se týče řazení kapitol, ani toto jsem nenechal náhodě. Jak jsem již předeslal, příručku jsem psal ve formě výukových lekcí (chcete-li kapitol). Každá kapitola pojednává o jiném aspektu WCF. Práci jsem koncipoval do dvou logických částí. První část se skládá ze tří lekcí, které spolu souvisejí: 1.Vytvoření WCF služby, 2.Hostování služby, 3.Vytvoření WCF klienta. Kapitoly jsou určené po úplné začátečníky ve WCF a prostřednictvím nich je čtenáři postupně vysvětleno a ukázáno, jak vytvořit a otestovat velmi

jednoduchou, ale funkční a plnohodnotnou WCF službu měřič Spotřeby spolu s klientskou aplikací.

Druhá část je určena pokročilejším zájemcům. Utvořena je ale i tak, aby byla zvládnutelná také pro začátečníky, kteří přečtou první část příručky. Výukové lekce v ní na sebe nenavazují a nemusí se tedy číst za sebou. To jsem udělal proto, aby si každý programátor mohl vybrat jen to, co ho zajímá a nebyl nucen, pro pochopení další lekce, číst předchozí. Myslím si, že je to vhodnější způsob, než psát strukturovaně, protože programátor je tvor lenivý a často si raději vyhledává jen to, co potřebuje. Do těchto kapitol jsem zahrnul takové rysy WCF, které jsem osobně nejvíce využil a doufám, že i pro další vývojáře, kteří budou tuto práci číst, nebudou zbytečné. Musel jsem vynechat spoustu méně, ale i více, užitečných vlastností WCF, neboť na ně prosté už nebyl prostor. Do kapitol jsem zahrnul například prvky kontroly výjimek, což je velmi potřebná vlastnost WCF, ale i kapitoly zajímavé pro reálné nasazení, kde vysvětluji řízení instancí a kontrolu zátěže na službě. Zařadil jsem i kapitolu o zabezpečení služeb. Podle mého názoru jsou všechny tyto aspekty velmi přínosné, což byl právě důvod, proč jsem je uvedl. Bohužel jsem byl nucen vybírat a do práce jsem nezahrnul transakce, relace, callback operace a další neméně užitečné prvky technologie.

3.4 Úvodní kapitola

Většina technologií v programování je spjata s nějakým prostředím a vývojářským nástrojem. WCF není výjimkou.

Nutné prostředí pro WCF je .NET Framework 3.0 a vyšší. Nemohu předpokládat, že každý, kdo čte tuto práci, má PC schopné provozovat .NET Framework 3.0 a vyšší, a proto jsem jako úvodní kapitolu přidal krátké pojednání o požadavcích WCF na systém. Snažil jsem se dbát o to, aby čtenář měl shodnou verzi Frameworku jako já. Je pravda, že teoreticky je jedno, jestli se použije .NET 3.0 nebo 3.5, ale praxe bývá často jiná. Z tohoto důvodu vyžaduji, aby měl nainstalován Framework 3.5 SP1, což je v současné době nejnovější verze. Předejde se tím zbytečným nesrovnalostem. Co se týká operačního systému, nechal jsem volbu na něm. Problém verzí operačního systému na funkčnost samotného programu není u WCF příliš podstatný, za zmínku stojí jen odlišná grafika stylů Windows Vista a XP a jiný vzhled WPF. Podstatné rozdíly se ale vyskytují v instalaci a použití programů a doplňků jako je IIS (probírám v lekci 2), balíků a podobně. V těchto sporných příkladech jsem se snažil otestovat a zveřejnit postup na obou současně používaných verzích operačního systému Windows, XP a Vista.

Jako vývojářský nástroj je možné při programování v .NET použít jakýkoli textový editor (například obyčejný poznámkový blok) a kompilátor .NET jazyka csc, nebo vbc. Nic dalšího pro funkčnost není třeba a všechny nutné

nástroje jsou nainstalovány jako součást balíku .NET Frameworku SDK. Pro většinu programátorů je ale tento způsob příliš komplikovaný a neefektivní, a ačkoli ho některé publikace preferují kvůli univerzálnosti, pro začátečníky i profesionální vývoj je nepoužitelný. V příručce tak požaduji, aby uživatelé měli nainstalován nějaký vývojářský nástroj, který programování diametrálně ulehčí a zrychlí. V současnosti vévodí na tomto poli Visual Studio firmy Microsoft. Výhodou Visual Studia je kromě výborné podpory ze strany samotného Microsoftu asi největší uživatelská oblíbenost. Na internetu je také většina návodů a článků k .NET (a samozřejmě WCF) ukazována právě na tomto nástroji. Neposlední výhodou je i ulehčení instalace .NET Frameworku SDK a dalších nástrojů pro vývoj v .NET. Ty se nainstalují automaticky spolu s Visual Studiem. Zdrojový kód pro výukové lekce vytvářím prostřednictvím Visual studia 2008 Professional v jazyce C#. Tato verze je zpoplatněná, a proto pro mnoho čtenářů nedostupná. Naštěstí existuje i volně dostupná varianta Visual Studio 2008 Express edition. Express edice plně dostačuje pro potřeby příkladů v příručce a její použití by se nemělo, v návodech co uvádím, oproti Professional edici příliš lišit. Přikládám tedy v úvodní kapitole odkaz, odkud se dá .NET Framework 3.5 SP1 a Visual Studio 2008 Express edition stáhnout.

3.5 Vytvoření WCF Služby

Stěžejní ve WCF jsou služby. První výukovou lekcí jsem proto chtěl čtenáře uvést do jejich problematiky. Tato kapitola nemluví o složitějších aspektech WCF služeb, ale o základních, nezbytně nutných při návrhu služby. Zahrnul jsem do ní vše potřebné pro to, aby zájemce uměl vytvořit tu nejjednodušší službu. Protože se jedná o první lekci, je mnohem více teoretická než ostatní části. Mluvím především o endpointech a prvcích, ze kterých se skládají: Address, Binding a Contract.

V kapitole také začínám psát první výukový příklad počítadlo spotřeby vozidel, který dále rozvíjím v následujících dvou kapitolách. Takže až ve třetí kapitole získá čtenář použitelný WCF program - jak službu a hosta, tak i klienta. První 3 lekce, jak už jsem zmiňoval, jsou určeny pro začátečníky a tvoří zhruba polovinu příručky. Pak následují kapitoly určené i pokročilejším. Toto členění jsem udělat proto, abych měl jistotu, že se v dalších kapitolách nebudu muset vracet k základům WCF, ale budu je již od čtenáře předpokládat.

3.5.1 Endpoint

Jedna z nejdůležitějších součástí WCF služeb je endpoint. Endpoint se často vysvětluje jako ABC (Address, Binding, Contract). Tato „abeceda“ je dobře zapamatovatelná a měla by čtenáři přejít „do krve“. Názorné je ukázat endpoint deklarativně v konfiguračním souboru (Příklad 1). To dělám hned na začátku lekce a pak se již zabývám jednotlivými prvky ABC.

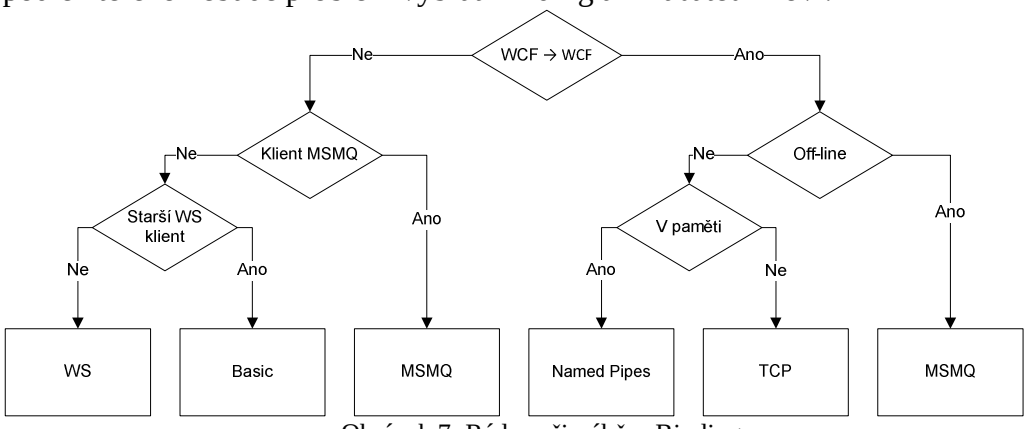
3.5.2 Koncipování CBA

Naprostá většina výukových zdrojů WCF je koncipována v případě endpointu v pořadí ABC. Čili nejprve se v nich čtenář seznámí s Address, pak s Binding, a nakonec Contract. Já jsem výukové lekce seřadil obráceně (CBA), což je sice lexikálně špatně, ale domnívám se, že je to lepší pro samotný návrh.

Hned vysvětlím, proč si to myslím. Při návrhu endpointu služby postupuji obvykle tak, že nejprve definuji Contract, poté Binding a nakonec Address. Je to z logických důvodů, protože tímto získám volnost, zvolit jakýkoli Contract chci. Binding a poté i Address přizpůsobím podle něj. Kdybych navrhoval jako první Address, musel bych už dopředu předvídat, pro jaké účely nasazení bude služba určena (internet: http, peer-to-peer: net.p2p), a měl bych svázané ruce u volby Binding. Binding by musel respektovat Address a nakonec Contract by musel být psán v souladu s Binding. Podle mého mínění je proto lepší začít s návrhem Contractu, tedy toho, co služba bude dělat a tam mít úplnou volnost. Raději si pak Binding a Address přizpůsobím podle něj. Samozřejmě je někdy nutné brát v potaz prostředí, kde bude služba běžet. Ale když ho dopředu neznáme, je podle mě vhodnější, navrhovat endpoint v pořadí CBA.

3.5.3 Výběr Binding

Při výběru systémových Binding je nutné dodržovat určitá pravidla. Obecně je třeba vědět, co od komunikace očekáváme, a podle toho Binding vybírat. Proto jsem do této části přidal ke každému z nich krátké pojednání, upravené z [13], které charakterizuje jejich nejdůležitější vlastnosti. Takto krátký popis ovšem většinou nestačí. Vytvořil jsem tedy i tabulku (Tabulka 1), popisující vlastnosti všech systémových Binding. Pro pokročilého zájemce je přínosná, protože si z ní může rychle nalézt, co potřebuje. Začátečník se ale nemusí umět ve změti parametrů dobře orientovat. Naštěstí jsem našel obrázek, podle kterého nebude problém vybrat Binding ani začátečníkovi.



Obrázek 7: Rádce při výběru Binding

(převzato z [14])

3.5.4 Base Address

Je to zvláštní, ale na způsob tvorby relativní adresy s pomocí base Address, se v českých článcích asi zapomnělo. Mám dojem, že by v mé příručce neměl chybět, protože tento způsob nemusí být každému znám, i když je poměrně intuitivní, proto ho asi autoři článků o WCF nezmiňují. Pokusil jsem se o jeho popsání. Píši o tom, jaký vztah je mezi absolutní a relativní adresou, jakým způsobem je relativní adresa vytvářena z base Address a jaký přínos tato reprezentace adresy přináší. Také nesmí chybět několik slov o nutnosti jedinečnosti adresy endpointu se stejným Contractem, ale poukážu i na možnost použít stejnou adresu u endpointů v případě odlišných Contractů.

Přece jen slovní popsání není tolik názorné, i když je pořád lepší než nic. Čtenář danou problematiku lépe pochopí na příkladu. Do příručky jsem proto přidal i příklad s vysvětlením (Příklad 10). Nadefinoval jsem tam dvě výchozí adresy služby. Jednu pro http a druhou pro tcp. Dále jsem použil 4 endpointy. Na prvních třech jsem chtěl ukázat využití relativní adresy, naproti tomu čtvrtá používá absolutní adresu. Na příkladu také může zájemce vidět, jak je podle Binding rozpoznána příslušná base Address k relativní adrese a jak je pak z jejich kombinace vytvořena adresa absolutní. Ukázka je navíc současně exhibicí různorodých systémových Binding, což jsem udělal proto, abych navázal na předchozí kapitolu o Binding a ukázal jak Binding souvisí s adresovým schématem.

```
<host>
  <baseAddresses>
    <add baseAddress="http://localhost/CR" />
    <add baseAddress="net.tcp://localhost:90/CR" />
  </baseAddresses>
</host>
<client>
  <endpoint
    address="Praha"            (absolutní adresa http://localhost/CR/Praha)
    binding="basicHttpBinding"
    contract="ISluzba" />
  <endpoint
    address="Brno"             (absolutní adresa http://localhost/CR/Brno)
    binding="wsHttpBinding"
    contract="ISluzba" />
  <endpoint
    address="Praha"            (absolutní adresa net.tcp://localhost:90/CR/Praha)
    binding="netTcpBinding"
    contract="ISluzba" />
  <endpoint
    address="net.msmq://localhost/private/Evropa/Viden"
    binding="netMsmqBinding"
    contract="ISluzba" />
</client>
```

Příklad 10: Base address

3.6 Hostování WCF Služby

V lekci o hostování navazuji na předchozí příklad samotné služby měřič spotřeby a předkládám dvě ukázky, jak službu hostovat. Z časových důvodů nebylo možné ukázat všechny možnosti. Proto jsem se zaměřil jen na ty, které si myslím, že čtenář nejvíce využije: self-hosting a IIS hosting. Jako self hosting jsem zvolil WPF aplikaci, protože jsem chtěl držet krok s dobou a WPF je přece jenom modernější než Windows Forms nebo konzole.

3.6.1 Self Hosting

V případě self-hostingu kladu důraz na vlastní hostující kód. Grafické uživatelské rozhraní a zdrojový kód, co nesouvisí s hostováním, nejsou pro výuku WCF tolik důležité. Tak jako u všech ukázek samozřejmě popisuji od začátku bod po bodu jak daný hosting vytvořit. Nejdůležitější zůstávají body 10 a 12. Nebudu tu teď zmiňovat všechny body, můžete je nalézt v příloze ve vlastní výukové lekci.

Podívejme se ale na onu hlavní část (Příklad 11), kde je prostřednictvím ServiceHost instance služba hostována. Co je stěžejní, jsou 3 metody. Chtěl jsem na ně více upozornit, proto jsem je tučně zvýraznil a popsal jejich funkci. Mimochodem v celé příručce často zvýrazňuji části, na které chci upozornit. Je to lepší, než je znovu opisovat, protože je nechci vytrhávat z kontextu, ale nechávám je se zbylým kódem, na místě kde patří. Někdy je zvýrazňuji tučně, jindy jim dávám žlutý podklad.

```
private void start_Click(object sender, RoutedEventArgs e)
{
     try
     {
        MericSpotrebyServiceHost = new
        ServiceHost(typeof(PocitadloSpotreby));
        MericSpotrebyServiceHost.Open();
        start.IsEnabled = false;
        stop.IsEnabled = true;
        stav.Text = "Služba aktuálně běží";
     }
     catch
     {
        MericSpotrebyServiceHost.Close();
     }
}
```

Příklad 11: Self-hosting

3.6.2 IIS Hosting

V podkapitole o vytvoření IIS hostingu jsem se kromě vlastního nastavení zmínil i o instalaci a konfiguraci IIS. I když tato věc přímo nesouvisí s WCF, považoval jsem za rozumné, obětovat tomu pár odstavců.

Instalaci a konfiguraci IIS se autoři, v některých příručkách o WCF, vyhýbají. Spoléhají se na to, že čtenáři použijí buď některý profesionální nebo free hosting (kde podobné věci řešit nemusí), nebo využijí integrované součásti Visual Studia, který je okleštěným IIS, ale na ukázku postačuje. Já jsem se rozhodl jít cestou plnohodnotného IIS a ukázat čtenáři, jak ho nainstalovat a nakonfigurovat virtuální adresář. Samozřejmě jsem nezabíhal do detailů, to by bylo na jinou příručku, ale vysvětluji jen povrchně věci, které se hodí pro účely WCF. Návod jsem napsal a otestoval pro Windows XP a Vista.

3.6.2.1 Problémy s IIS

I když se na první pohled zdá, že nás v takto triviální záležitosti nemůže nic překvapit, opak je pravdou. Kromě vlastní instalace je totiž nutná i registrace WCF, která není samozřejmá a velmi často po instalaci chybí. Tento poznatek je sice zmiňován v msdn, ale vůbec ne tam, kde bych ho čekal, a proto jsem strávil nějaký čas prohledáváním fór. Stálo to ale za to a výsledek doufám bude čtenářům užitečný. Na ukázku přikládám výňatek z přílohy o registraci na Windows XP.

1) Windows XP a Windows Server 2003: Použijte ServiceModelReg.exe k registraci WCF s IIS: V příkazové řádce napište %SystemRoot%\Microsoft.Net\Framework\v3.0\Windows Communication Foundation**ServiceModelReg.exe /i /x** .

Někdy se může stát, že .svc soubor není namapován na aspanet_asapi.dll. Je to třeba provést ručně. Ani tento drobný fakt jsem nemohl opomenout, neboť by mohl prakticky znemožnit jakékoli zkoušení mého příkladu. Proto jsem zájemcům přidal i odkaz na článek, kde se o tomto problému píše.

3.6.2.2 Vytvoření IIS hostingu

V této části kladu důraz na 3 základní elementy pro správnou funkčnost IIS. Ve skutečnosti jsou totiž potřeba pouze 2 soubory (.svc se službou nebo odkazem na službu a web.config s konfigurací), umístěné ve virtuálním adresáři IIS. Způsoby, jak vytvořit virtuální adresář, zmiňuji dva. Buď ručně pomocí IIS konfigurace anebo pomocí Visual Studia 2008 ve vlastnostech projektu. Pro programátory je pohodlnější druhý způsob, ale na druhou stranu z hlediska profesionálního použití, se může hodit i první, neboť skrývá širší možnosti nastavení. Proto jsem raději zájemcům představil oba.

3.6.3 Rozdíly mezi konfiguracemi IIS a self-hosting

V části o hostingu jsem nemohl opomenout konfiguraci služby, kterou jsem záměrně neprobíral v první lekci (Vytvoření WCF služby). V první lekci jsem sice vysvětloval některé části z konfigurace jako takové, například element endpoint, ale jelikož se konfigurace váže nejen ke službě, ale i k hostingu (element host), nechal jsem ji na tuto kapitolu o hostování. U IIS je konfiguračním souborem Web.config, u self-hostingu obvykle App.config. Nastavení je podobné, ale přesto určité odlišnosti existují. Takže ve svém příkladu ukazuji obě konfigurace a vysvětluji na nich rozdíly, aby některé čtenáře nezmýlilo, že není konfigurace stejná. Rozdílů není mnoho. V podkapitole vystihuji podstatnou odlišnost, že ve Web.config se musí adresy endpointů uvádět vždy jako relativní a také zdůrazňuji fakt, že oproti konfiguraci u self-hostingu není nikde element host s base Address. Base Address je totiž brána pro IIS jako virtuální cesta k .svc souboru. To by si měl každý programátor uvědomit a nesnažit se výchozí adresu nastavovat ručně.

```
<system.serviceModel>
  <services>
    <service name="MericSpotreby.PocitadloSpotreby"
        behaviorConfiguration="MyBehavior">
      <endpoint
      address=""
      binding="wsHttpBinding"
      contract="MericSpotreby.IPocitadloSpotreby" />
    </service>
  </services>
  <behaviors>
    <serviceBehaviors>
      <behavior name="MyBehavior">
        <serviceMetadata httpGetEnabled="true"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
```

Příklad 12: Ukázka konfigurace ve Web.Config souboru pro IIS hosting

3.6.4 Přidání služby formou reference

Výstupem první výukové lekce (Vytvoření WCF služby) byla knihovna se službou bez vlastní konfigurace. Až ve druhé kapitole se služba dostala do hostujícího prostředí. To je jedna možnost, kdy je služba oddělena od hostujícího prostředí, ke kterému je přidána přes volbu přidat reference. Není to ale jediný způsob. Možná ještě používanější je vytvořit službu rovnou jako součást hosta. Tak, že kód služby bude ve stejném souboru, jako je hostující kód, tudíž ve stejné assembly. V případě IIS, kód služby píšeme rovnou do .svc souboru. Každý přístup má své výhody a nevýhody. Výhody jednoho přístupu jsou nevýhodami druhého:

- **Služba v oddělené knihovně**
  + Přenositelnost, jedna služba může být hostována více hosty, aniž bychom museli duplikovat kód.
  + Omezení duplicity, máme-li službu hostovánu na více místech a změníme-li něco ve službě, změna se projeví ve všech hostech.
- **Služba s hostem v jednom projektu**
  + Jednoduchost, nemusíme vytvářet 2 assembly.
  + Můžeme upravovat kód na jednom místě, všechno máme „po ruce“.

Rozhodl jsem se pro první způsob z důvodu, abych snadněji oddělil lekci s vytvořením služby od hosta a také, aby čtenář nemusel kopírovat kód služby do druhého hosta, a nakonec i proto, že je to způsob profesionálnější a více v praxi používaný.

3.7 Vytvoření WCF Klienta

Část pro začátečníky uzavírám kapitolou o WCF klientech. Tato práce má název Služby WCF a opravdu je především o službách. Nesnažil jsem se tedy kapitolu o klientech příliš rozepisovat. Přesto jsem ji musel do práce začlenit, protože bez klienta není možné vyzkoušet, jak služba funguje.

3.7.1 Komunikace služby a klienta

Aby si čtenář udělal představu, jak vlastně služby a klienti komunikují, jakým způsobem mezi nimi zprávy putují, přidávám do této lekce i část o komunikaci mezi službou a klientem. Velmi hezky jí popsal Juval Löwy v ([2, kap. 1.11). Proto jsem jeho výklad včetně obrázku z kapitoly 1.11 použil ve své příručce. Mimochodem, to samé používám i jako teoretický podklad v této práci pro kapitolu 2.6.2.

3.7.2 Postup při návrhu klienta

[20] uvádí, že nejběžnější postup při návrhu klienta je ve 4 bodech. Jsem téhož názoru a stejně vysvětluji, jak vytvořit i svého klienta. I tohoto klienta vytvářím ve WPF, tak jako všechny další v následujících kapitolách. Jak už jsem zmínil v kapitole o hostování, je to proto, že se jedná se o moderní technologii a také WPF spolu s WCF a jinými patří do skupiny příbuzných technologií, vydané ve Frameworku .NET 3.0. Klienta tedy navrhuji v následujících 4 bodech.

- Získáme Service Contract, Binding, a Address z endpoint služby.
- Vytvoříme WCF klienta použitím těchto informací.
- Voláme operace na klientovi.
- Zavřeme WCF klient objekt.

Jako první získávám endpoint informace z metadat služby. K tomuto bodu bych rád dodal, že existuje více způsobů, jak toho docílit. Protože od čtenářů vyžaduji Visual Studio, ukazuji to prostřednictvím něho, tedy přes volbu Add Service Reference. Tento způsob je pro začátečníky jednodušší a také i rychlejší. Osobně ho používám, když chci rychle přidat referenci na službu bez nějakých dodatečných procedur. Na druhou stranu je vhodné znát i standardní a běžný způsob, kterým je možno získávat endpoint informace bez nutnosti Visual Studia. Tím je utilita svcutil.exe, proto píši i o tomto způsobu.

Tak jako u prvního bodu i pro druhý bod není pouze jedna cesta realizace. Nejběžnější je vytvořit instanci proxy třídy, kterou jsme získali v předchozím bodu. Ve všech ukázkách v příručce takto klienta realizuji a je to natolik běžný a používaný způsob (setkal jsem se s ním ve všech publikacích, které jsem studoval), že nemohl chybět ani v mém výukovém materiálu.

```
using (PocitadloSpotrebyClient proxy = new
PocitadloSpotrebyClient("WSHttpBinding_IPocitadloSpotreby"))

{
    Stejné
```

Příklad 13: Vytvoření klienta prostřednictvím proxy třídy

Mnoho programátorů zná pouze první variantu vytvoření WCF klienta a s tou si vystačí na všechny scénáře. Svým způsobem to je pravda, ale v některých případech jde použít i jiná možnost, která není tolik běžná, a proto ji autoři v některých publikacích neuvádí. Já jsem ji však do své příručky zahrnul z důvodu vhodnosti využití v určitých případech. A to tehdy, kdy je klientu přístupný i kód rozhraní služby. Bývá to často, když jsou služba a klient programovány stejnou společností, týmem či programátorem. Potom lze klienta vytvořit přímo v kódu prostřednictvím CannelFactory objektu bez nutnosti generování proxy třídy. Dobré na tom je, že rozdíl mezi oběma přístupy je z programátorského hlediska pouze v pár řádcích kódu. V příkladu tyto řádky zvýrazňuji, aby měl čtenář snazší porovnání. Volba přístupu je již na něm.

```
Binding binding = new WSHttpBinding();
EndpointAddress address = new
EndpointAddress("http://localhost:8000/Pocitadlo ");
IPocitadloSpotreby proxy =
ChannelFactory<IMyContract>.CreateChannel(binding, address);
using (proxy as IDisposable)
{
 Stejné
```

Příklad 14: Vytvoření klienta prostřednictvím ChannelFactory v kódu

Celé vytvoření proxy instance je otázka jednoho řádku. Důležitá je ale skutečnost, že kromě vytvoření je nutné i korektní zavření proxy třídy. Jednoduché a užitečné pro daný příklad je použít blok using, protože

programátor se může soustředit na samotné volání metody a nemůže se stát, že nedojde k zavření proxy. Vhodná ale není pro většinu případů, například tehdy, když jsou odchytávány v aplikaci výjimky, neboť i samotné zavření proxy může způsobit výjimku, která pak překryje kód za uzavírací složenou závorkou bloku using (více se o tom zmiňuji v 6. kapitole Chyby a Výjimky).

3.8 Instance management a Concurrency

Čtvrtá lekce již patří do trojice lekcí pro čtenáře, kteří znají základy WCF. Bude užitečná, jak pro pokročilejší čtenáře, tak i pro začátečníky. V této a následující kapitole totiž píši o technikách pro zvýšení propustnosti služby. Doufal jsem tedy, že by mohly čtenáře zajímat, neboť je naučí dělat zásahy, které mají praktický dopad na vlastní výkonnost, kvalitu a dostupnost služby v reálném použití.

Hovořím zde o dvou aspektech chování služby (ServiceBehavior), kterým je InstanceContextMode a Concurrency. V teorii naučím čtenáře, jak prostřednictvím InstanceContextMode kontrolovat vytváření instancí služby (instance management). Jinak řečeno, jak službu nastavit, aby mohla určit vytváření svých instancí v závislosti na požadavcích od klienta. To je ovšem jen polovina teorie. Píši taktéž o Concurrency a vysvětluji, jak prostřednictvím ní řídit vytváření vláken uvnitř instance služby. Nebudu tu psát o teorii, spíše se chci zaměřit na to, jak jsem vymyslel výukový příklad, aby byl pro čtenáře co nejpřínosnější.

3.8.1 Tabulka chování služby

Ačkoli je možné vysvětlovat InstaceContextMode a Concurrency odděleně, myslím, že to není příliš vhodné, neboť obě nastavení spolu souvisí a až jejich vzájemná kombinace má konečný vliv na chování služby. Přidal jsem proto do příručky, podle mého mínění, velmi přínosnou tabulku, na kterou jsem náhodou narazil v ([4], str. 186). V této tabulce je popsáno chování pro všech devět případů společných kombinací jednotlivých parametrů (tři možnosti pro InstanceContextMode a tři Concurrency mode), se kterými může programátor WCF přijít do styku.

3.8.2 Příklad nastavení instancing

Pro tuto kapitolu jsem chtěl vytvořit vzorový příklad, na kterém budou názorně vidět všechny způsoby nastavení InstaceContextMode. Vymyslel jsem tedy aplikaci, obsahující službu a klienta. Pro kontrolu instancing je třeba nějak jednotlivé instance jednoznačně odlišit. Udělal jsem to tak, že jsem vytvořil proměnnou cisloInstance, která uchovává její pořadové číslo vytvoření. Hodnotu této proměnné získám tak, že při vytvoření instance služby

v konstruktoru služby, inkrementuji statickou proměnnou CelkovyPocetInstaci (shodnou pro všechny služby), která bude obsahovat celkový počet instancí služby. CisloInstance je v momentu vzniku instance stejná, jako toto číslo, až do okamžiku vytvoření další instance služby. Každá instance bude mít tedy cisloInstance jedinečné, a to je to, čeho jsem potřeboval docílit. Má to ale jeden háček. V případě likvidace instance služby, nedojde k snížení proměnné celkovyPocetInstanci. Ačkoli by nebyl problém i tuto funkcionalitu dodělat, neudělal jsem to z prostého důvodu, protože by se pak mohlo stát, že by cisloInstance pro instanci vytvořenou v budoucnu nebylo jedinečné.

Mám tedy proměnnou s jedinečným, chcete-li identifikačním číslem služby. Další věcí, co jsem pro příklad potřeboval, abychom mohli vidět, jaké instance služby jsou volány jakými relacemi, je nějaká metoda služby, kterou budou volat samotní klienti. Proto jsem jako OperationContract služby použil jedinou metodu s parametrem zprava. Prostřednictvím ní klient bude moci zavolat službu a předat zprávu, která se zobrazí na konzoli hostujícího prostředí služby a to s identifikačním číslem instance.

```
[ServiceBehavior
(InstanceContextMode=InstanceContextMode.PerrCall)]
class ZobrazovacZprav : IZobrazZpravy
{
   private static int celkovyPocetInstanci;
   private int cisloInstance;

   public ZobrazovacZprav()
   {
         cisloInstance = Interlocked.Increment(ref
         celkovyPocetInstanci);
   }
   public void TiskZpravy(string zprava)
   {
        Console.WriteLine("Instance číslo {0} zobrazuje zpravu:
        {1}",cisloInstance, zprava);
   }
}
```

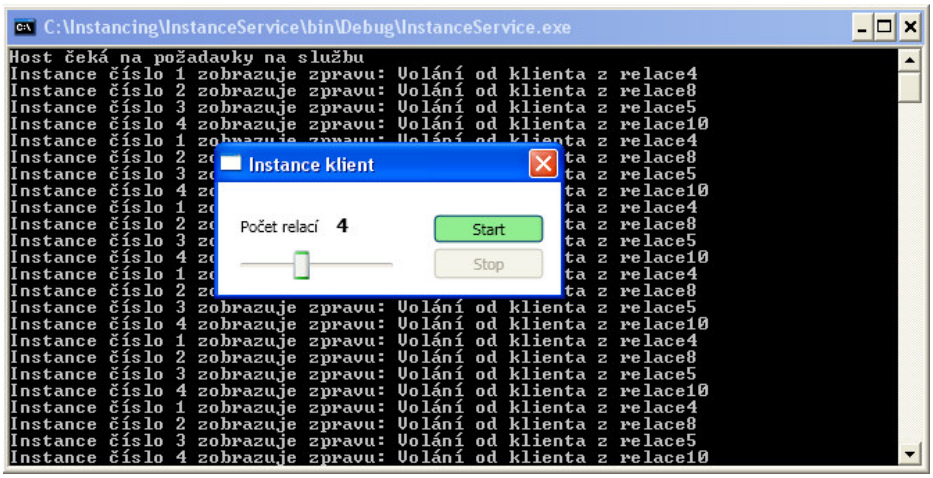
Příklad 15: Nastavení InstanceContextMode služby v příkladu Instancing

Proto pro demonstraci instancing budu jako poslední věc potřebovat klientskou aplikaci, která zavolá metodu služby, o které jsme mluvili, a předá do zprávy identifikační číslo relace (proxy třídy). Pak bude možné na konzoli odlišit, z jaké relace byla instance volána. Vytvořil jsem tedy jednoduché okno, kde si uživatel nastaví počet vláken (která odpovídají oněm relacím na službu). Tlačítkem Start je pak aktivován příslušný počet vláken. Každé vlákno, ve vlastní obslužní metodě, vytvoří jednu proxy instanci (relaci) a v intervalech 300 milisekund volá metodu TiskZpravy na službu, dokud není stisknuto tlačítko Stop. Jako parametr předává ID vlákna, čímž odliší, z jaké relace byla metoda služby volána.

```
InstanceService.ZobrazZpravyClient proxy = new
InstanceClient.InstanceService.ZobrazZpravyClient();
while (!zastaveno)
{
      proxy.TiskZpravy("Volání od klienta z relace" +
      Thread.CurrentThread.ManagedThreadId.ToString());
      Thread.Sleep(300);
}
```

Příklad 16: Volání služby z klienta v příkladu Instancing.

Tímto trikem jsem dosáhl toho, že budu moci kontrolovat, jaké relace budou obsluhovat jakou instanci třídy, a budu tedy moci sledovat Instance Management služby. Zájemce si pak může pouhou změnou jednoho slova nastavit InstanceKontextModu služby a výsledek vidět názorně na konzoli. Přikládám výpis pro nastavení PerrSession (Obrázek 7), kdy nová instance služby je vytvořena pro každé nové session(spojení) klienta a je udržována po celou dobu spojení. Na výpisu je na první pohled názorně vidět, že všechny požadavky z jedné relace jsou obsluhovány stejnou instací, což potvrzuje teorii o chování PerrSession modu. Zbylá dvě nastavení uvádět nebudu, ale v příručce samozřejmě nechybí.



Obrázek 8: Výpis konzole příkladu Instancing - nastavení PerrSession

Mimochodem právě k tomuto příkladu jsem vytvořil videoukázku, kde podobně, jako tady, vysvětluji funkci instancing na tomto příkladu. Výhoda videoukázky je ale v tom, že konzolové výstupy jsou ještě lépe patrnější, neboť jsou názorně vidět za chodu.

3.9 Throttling

Další možností, jak kontrolovat propustnost služby, konkrétně omezit její zatížení, je prostředek zvaný throttling (volně přeloženo škrcení). Jedná se o poměrně užitečný a snadno pochopitelný prostředek, jak omezovat počet volání, relací a instancí služby. Kapitolu jsem zařadil do příručky, protože jsem chtěl zájemcům vysvětlit, jak mohou ochránit službu proti přirozenému přetížení, ale i útokům, což by asi mohlo čtenáře zajímat.

Throttling je poměrně jednoduchý. V teorii toho v podstatě není nutné moc vysvětlovat, stačí krátce popsat jednotlivé 3 druhy omezení a na příkladu ukázat, jakým způsobem je nastavit Přínosné je také napsat, jaké jsou defaultní hodnoty a doporučit v jakém poměru je vhodné jednotlivé módy nastavovat. Já jsem to ukázal, tak jako u mnoha dalších příkladů, jak pomocí administrativní konfigurace v konfiguračním souboru (Příklad 17), tak i programově v kódu.

```
<serviceBehaviors>
  <behavior name = "ThrottledBehavior">
    <serviceThrottling
       maxConcurrentCalls     = "8"
       maxConcurrentSessions  = "40"
       maxConcurrentInstances = "200" />
  </behavior>
</serviceBehaviors>
```

Příklad 17: Konfigurace Throttling

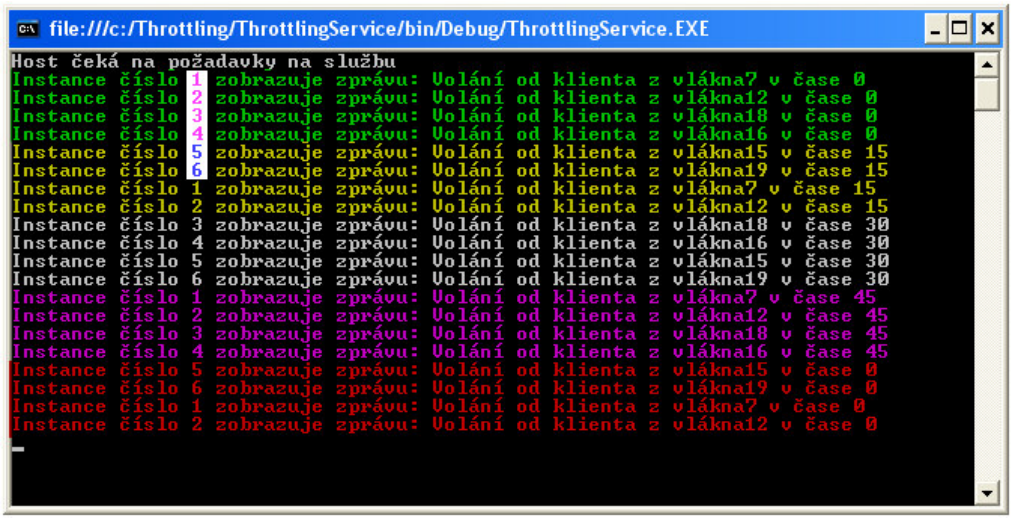
3.9.1 Příklad omezení počtu volání a relací služby

Jako příklad pro tuto lekci jsem vymyslel ukázku omezení počtu instancí a relací. Tudíž parametry maxConcurrentCall a maxConcurrentSessions. Tyto dva parametry jsem zvolil z toho důvodu, protože počet instancí u nastavení instancing PerrSession, které v tomto příkladu využívám, odpovídá počtu relací a nehraje tedy v tomto nastavení žádnou roli. U nastavení perrCall je to obdobné k počtu volání. Není tedy dost dobře možné ukázat na jednom příkladu současné omezení prostřednictvím všech tří parametrů, takže právě proto volím první dva. Navíc vzhledem k tomu, že se jedná o téma natolik přímočaré, jak funguje třetí parametr, čtenář pak snadno vydedukuje sám.

Výhodné, jak pro mě, tak i pro čtenáře na tomto příkladu je, že jsem mohl využít klienta a službu z předchozí lekce, které se na tuto ukázku hodily. Problematika je totiž značně příbuzná, že by bylo zbytečné vytvářet pro ni vlastní příklad. Samozřejmě, ale mírné změny nutné byly. Musel jsem změnit instancing služby na perrSession, protože bych jinak při původním nastavení perrCall nemohl demonstrovat funkci maxConcurrentSessions. Dále jsem poupravil metodu TiskZpravy. Do výpisu jsem kromě čísla instance služby a ID vlákna (relace) přidal ještě čas volání, a to proto, abych rozlišil stejné

požadavky. Kromě výpisu jsem do metody zahrnul i čekání 15s. Čekání je z toho důvodu, abych simuloval zátěž na službě, bez něho by se požadavky střídaly příliš rychle a ve stejné době by na službu nepřišlo tolik požadavků, aby se vůbec throttling mohl projevit. Samozřejmě poslední nejdůležitější změnou oproti předchozímu příkladu je nastavení samotného throttling v konfiguračním souboru služby. Volil jsem hodnoty raději nižší, aby bylo omezení lépe patrné. maxConcurrentCall ="4", maxConcurrentSessions ="6", maxConcurrentInstances ="10".

V příkladu dále vyzývám ke kompilaci a vyzkoušení na 9 relací. Výsledek je pro zájemce velmi zajímavý. V konzolové aplikaci se začne zobrazovat, jak vznikají instance služby. Po první čtveřici dojde k čekání, protože maximální souběžný počet volání je nastaven na 4. Další čtveřice se zobrazí následně a potom další atd.. Čtenář tak názorně vidí, že ačkoli nastavil počet vláken (relací) na 9, je ve výpisu obslouženo pouze 6. To je také důsledek omezení celkového počtu relací na 6 prostřednictvím maxConcurrentSessions. Kromě vlastního throttling má zájemce možnost vidět i jiný zajímavý rys WCF a tím je TimeoutException. Po 1 minutě totiž dojde k TimeoutException, protože u první neobsloužené relace vypršela defaultní čekací lhůta (timeout), proto po 60s nenásledují další řádky výpis končí a doc´hazí k výjimce. Na obrázku 8 je výpis z konzole, kde jsem pro názornost barevně zvýraznil důležité části.



Obrázek 9: Výpis konzole příkladu Throttling

3.10 Chyby a výjimky

V této lekci jsem chtěl vysvětlit, jak se obsluhují a vytvářejí výjimky v prostředí WCF. Kapitola je to důležitá, neboť každá aplikace by měla obsahovat mechanismus pro kontrolu chyb a výjimek. WCF a mnohé SOA technologie zpracovávají chybové informace prostřednictvím SOAP fault zpráv. Tato kapitola proto hovoří především o SOAP faultech. Je velmi obsáhlá, protože jsem chtěl vysvětlit různé scénáře, jejich definice a obsluhy. Tak jako každá jiná kapitola začíná teorií. Tu jsem koncipoval do tří částí, podle toho v jakém pořadí je bude i vývojář programovat:

- **Definování a specifikace Fault:** Krátká část o tom, jak definovat typ s detaily o chybě pro použití uvnitř generického faultu.
- **Posílání Fault:** Vysvětluje jak posílat declared (generické) a undeclared (negenerické) faulty a co ovlivňuje IncludeExceptionDetailInFaults.
- **Přijímání a obsluha Fault:** Jaké druhy výjimek mohou u klienta vyvolávat SOAP faulty.

3.10.1 Příklad definování fault a odchytávání výjimek

Stěžejní, pro pochopení této lekce není teorie, ale ukázkový příklad, kterému jsem věnoval více úsilí než ostatním příkladům v příručce. Vytvořil jsem službu DatabazeOsob, implementující tři ServiceContract metody pro přidávání osob do služby, databáze nebo souboru. Výhodou tohoto příkladu je také to, že Osoba je DataContract objekt, takže čtenář na ukázce uvidí i praktické použití tohoto aspektu WCF, které jsem neuvedl v první kapitole (Vytvoření WCF služby). Klientskou aplikaci jsem vytvořil v podobě WPF formuláře pro nadefinování osoby a tři tlačítek, volajících prostřednictvím proxy ony tři metody služby. Pak je simulováno přidání osoby pomocí jedné metody do služby, databáze nebo souboru. Důležité ale je, že klient může při definování osoby ve formuláři udělat chybu - například neuvede jméno, udá špatný věk, atd.. Tento objekt Osoby je pak zkontrolován na straně služby, a je-li zjištěna nějaká nesrovnalost, služba vyvolá výjimku, která je předána klientovi, odchycena a zobrazena prostřednictvím messageBoxu. Takže zájemce si na tomto příkladu má možnost vyzkoušet, jak funguje WCF model chyb a výjimek.

3.10.1.1 Různé druhy výjimek

Na příkladu jsem chtěl ukázat všechny možné druhy výjimek, které mohou u WCF služeb nastat. Druhů výjimek je opravdu hodně. To je také důvod, proč jsem použil více metod služby, abych je vůbec mohl nasimulovat. Pro každou nejpoužívanější skupinu výjimek jsem tedy zvolil jednoho či více zástupců.

- Pro deklarované faulty jsem vybral jeden, přejímající informace o chybě z vlastního serializovatelného objektu: FaultException<OsobaFault>, a druhý přijímající objekt .NET: FaultException<NotSupportedException>.
- Pro nedeklarované faulty vyvolávám jeden negenerický FaultException.
- Ukazuji též klasické .NET výjimky, nezahrnuté ve FaultContractu služby, například NotImplementException a vysvětluji na nich, jak funguje nastavení includeExceptionDetailInFaults pro true a false.
- U klienta je ještě nepřímo vyvolána, díky čekání v jedné z metod služby, i další specifická výjimku WCF, kterou je TimeoutException.

3.10.1.2 Obsluha WCF výjimek

Obsluze WCF výjimek u klientské aplikace věnuji velkou pozornost, protože zahrnuje několik důležitých částí, jejichž špatná aplikace by mohla mít za následek přinejmenším neobsloužení výjimky a přinejhorším neprovedení kódu programátora. V části služby jsou v závislosti na metodě vyvolávány mnohé druhy výjimek, které jsou na pozadí WCF mapované na SOAP faulty. U klienta jsou tyto faulty vyvolávány jako výjimky několika typů. Abych ukázal, jakým způsobem obsluhovat všechny, musel jsem blok try-cache velmi zvětšit. Ve skutečnosti není obsluha tolika druhů výjimek v naprosté většině případů potřebná, ale pro ukázku je užitečné ukázat všechny možnosti.

```
FaultsService.PridatOsobuClient proxy = new
FaultsClient.FaultsService.PridatOsobuClient();
try
{
    proxy.PridejOsobu(PredejOsobu());
    proxy.Close();
}
catch (FaultException<FaultsService.OsobaFault> osobaFault)
{
    MessageBox.Show("Došlo k tomuto problému u osoby " +
    osobaFault.Detail.Osoba + "."  + "\n" +
    osobaFault.Detail.Problem);
    proxy.Abort();
}
catch (FaultException<NotSupportedException> fault)
{...}
catch (FaultException fault)
{...}
catch (TimeoutException)
{...}
catch (CommunicationException)
{...}
catch (FormatException)
{...}
catch (Exception ex)
{...}
```

Příklad 18: Obsluha WCF výjimek

3.10.1.3 Pořadí obsluhy výjimek

Nemohl jsem zapomenout v této části napsat i odstavec o pořadí obsluhy výjimek. Jedná se sice o vlastnost spíše samotného jazyka C# a .NET, ale domnívám se, že bych to měl uvést v příručce o WCF, protože někteří programátoři nemusí znát hierarchii dědění WCF výjimek. Mohou se tak dostat do zbytečných a těžko odladitelných situací, kdy si zakryjí obsluhu výjimky jejím předkem. Vyzdvihl jsem především nutnost psát FaultException v bloku try-cache před CommunicationException, jak je správně vidět v předchozí ukázce. Je to z důvodu, že FaultException je od CommunicationException odvozena. Pokud by byla uvedena až za ní, byla by zastíněna, a nebyla by nikdy provedena. Obdobně to platí i pro generickou FaultException<*Type*>, která je zase odvozena od negenerické FaultException a samozřejmě totéž platí i pro obecnou Exception.

3.10.1.4 Korektní zavření WCF klienta

Do této části jsem také zahrnul už dříve nastíněný problém používání using bloku u WCF klienta. Tento problém nebývá mnohdy v publikacích o WCF zmiňován a přitom je velmi důležitý. Using totiž na konci bloku volá metodu Dispose, která odpovídá metodě Close na proxy objektu. Problém je, že tato metoda může způsobit výjimku, když dojde k chybě sítě během zavírání. Proto v příručce doporučuji, aby programátoři nejprve WCF klienta vytvořili, pak otevřeli, používali a nakonec zavřeli v tom samém try bloku. Chyby pak obsluhovali v blocích catch, kde ovšem nesmí zapomenout WCF klienta zavřít pomocí Abort metody, která výjimku nezpůsobí. Podle tohoto doporučení, jak vidíme v předchozí ukázce, je psán klient v příkladu.

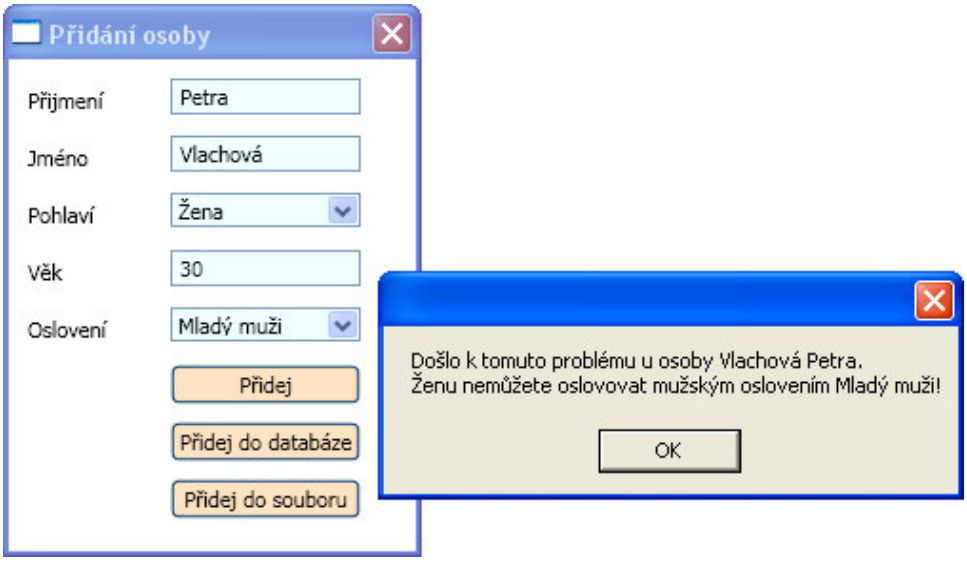
```
using (PridatOsobuClient client = new PridatOsobuClient())
{
 ...
} // <-- Tato část může způsobit výjimku
Console.WriteLine("k tomuto kódu nemusí dojít");
```

Příklad 19: Problém při použizí bloku using u WCF klienta

3.10.1.5 Ukázka výstupu z příkladu

Troufám si říci, že až při samém vyzkoušení projektu zájemce skutečně pochopí, jak WCF výjimky a chyby fungují. Bez konečného otestování by byl tento komplexní příklad jen holým (i když popsaným) kódem. Proto, jako u většiny lekcí, jsem i do této přidal pokyny k testování a jejich výstup. Čtenář na obrázcích a při zkoušení uvidí, že to, co se naučil v teorii a příkladu, skutečně funguje. Na konci kapitoly proto dávám pokyny jak otestovat všechny možné scénáře. Přidávám obrázek jednoho z nich, kdy uživatel zadá oslovení, které se neslučuje s pohlavím osoby. WCF model výjimek pak vyvolá declared

FaultException<OsobaFault>. Ten je obsloužen u klienta první catch klauzulí (příklad 15).



Obrázek 10: Ukázka výstupu obsluhy WCF výjimky z příkladu Faults

3.11 Zabezpečení

Jako poslední lekci jsem zařadil kapitolu o zabezpečení. Distribuované technologie potřebují mechanismy pro autentizaci a autorizaci přístupu, zaručení ochrany, konzistence přenosu dat a podobně. WCF není žádnou výjimkou a obsahuje množství technik pro jejich realizaci. Jedná se o komplexní a velmi rozsáhlé téma, které by vydalo na samostatnou knihu. Proto jsem nemohl probrat všechno a zaměřil jsem se jen na to nejvíce používané. I tak je tato kapitola největší ze všech, co do rozsahu i množství informací. Jako v každé jiné kapitole, jsem napsal teorii a výukový příklad.

3.11.1 Základní pojmy

Hodně jsem přemýšlel, jakým vhodným způsobem oblast zabezpečení nastínit. Bez teoretického podkladu a hlavně vysvětlení klíčových pojmů v této problematice, nelze podnikat žádné kroky, protože neznalý čtenář by mohl ze samotného příkladu snadno nabýt dojmu, že problematiku zná a může ji aplikovat na všechny scénáře, což je samozřejmě mylná domněnka. Chtěl jsem tedy vysvětlit některé základní informace a pak se teprve pustit do samotného příkladu. Zabezpečení přístupu k informačním systémům se obecně rozděluje do několika oblastí, což reflektují například v ([4] str. 316,317), ([2] kap. 10.1-10.3) a hlavně [8]. Shrnul jsem to do následujících bodů:

- Autentizace
- Autorizace
- Zabezpečení přenosu

Tyto pojmy by neměly být žádnému zájemci o zabezpečení cizí. V příručce vysvětluji nejen jejich význam, ale objasňuji i to, jak jsou implementovány ve WCF. Víceméně se pro každý prvek jedná o výčty různých technologických a systémových možností, které podporuje WCF, a které jsem získal ze zmíněných publikací [8]. Postupně tedy čtenáře provedu všemi fázemi zabezpečení WCF. Začnu s autentizací totožnosti uživatele, po jehož úspěšném provedení následuje autorizace pro přístup k operacím. Autentizace a autorizace jsou k ničemu, pokud nezabezpečíme i přenos dat. O tom, jaké nám k tomu dává WCF mechanismy, mluvím v podkapitole Zabezpečení přenosu.

3.11.1.1 Autentizace

Víceméně jsem pouze vysvětlil pojem a přeložil ([9] část Authentication), funkci jednotlivých autentizačních módů. Část je natolik přímočará, že jsem nepovažoval za nutné, ji probírat detailněji.

3.11.1.2 Autorizace

Stejně jako u Autentizace, vysvětluji nejprve to, co to autorizace je, a dále se zabývám jednotlivými autorizačními nastaveními. Co je ale nejdůležitější, je samotné aplikování těchto autentizačních mechanismů na jednotlivé metody služby. Toho se dá docílit dvěma způsoby, deklarativně a programově. Uvedl jsem oba přístupy. Vymyslel jsem pro ně jednoduchý ukázkový příklad, který je značně srozumitelný, což je v tak složité oblasti, jako je zabezpečení WCF, velmi vzácný jev. Přikládám tedy deklarativní ukázku definování práv z příručky.

```
[PrincipalPermission(SecurityAction.Demand, Role = "ucetni")]
[PrincipalPermission(SecurityAction.Demand, Role = "vedci")]
public double Nasob(double a, double b)
{
    return a * b;
}
```

Příklad 20: Nastavení autorizačních práv pro metodu služby

3.11.1.3 Zabezpečení přenosu

V této podkapitole mluvím, kromě toho nejdůležitějšího, což je vysvětlení jednotlivých módů, ještě o třech podmínkách, které musí každý zabezpečený přenos dodržet. Myslím si, že by o nich měl čtenář vědět, neboť je považuji za další klíčové poznatky oblasti zabezpečení.

- **Integrity:** Zabezpečuje integritu zpráv během přenosu. Musí zaručit, aby zpráva nebyla narušena a pozměněna někým během přenosu od služby ke klientovi a obráceně. Obvykle se toho dociluje hashovacími technikami.
- **Privacy (Confidentiality):** Zaručuje důvěrnost zpráv. Tedy, že nikdo další nemůže přečíst obsah zprávy. Toho je nejčastěji dosaženo prostřednictvím šifrování.
- **Authentication:** Stará se o autentizaci. Autentizace je zkontrolování a potvrzení správné identity příjemce. Díky tomu je pouze očekávaný, čili korektní příjemce, schopný číst obsah zpráv.

Model WCF je velmi rozšiřitelný a platí, že jde kombinovat různé vlastnosti. Ne všechny kombinace jsou ale funkční a ne všechny dávají smysl. U zabezpečení to platí dvojnásob. Uvedl jsem proto i dvě zajímavé tabulky, které jsem nalezl v knize ([2] kap. 10.12), a které pojednávají o kompatibilitě jednotlivých Binding s módy zabezpečení. Přikládám jednu na ukázku.

Name	None	Transport	Message	Mixed	Both
BasicHttpBinding	Ano (výchozí)	Ano	Ano	Ano	Ne
NetTcpBinding	Ano	Ano (výchozí)	Ano	Ano	Ne
NetPeerTcpBinding	Ano	Ano (výchozí)	Ano	Ano	Ne
NetNamedPipeBinding	Ano	Ano (výchozí)	Ne	Ne	Ne
WSHttpBinding	Ano	Ano	Ano (výchozí)	Ano	Ne
WSDualHttpBinding	Ano	Ne	Ano (výchozí)	Ne	Ne
NetMsmqBinding	Ano	Ano (výchozí)	Ano	Ne	Ano

Tabulka 2: Podpora transfer módů pro některé systémové Binding
([2], kap. 10.12)

3.11.2 Auditing

Po vysvětlení základních pojmů zabezpečení a jejich aplikace ve WCF, jsem se rozhodl do této lekce zahrnout pojednání o auditing, čili monitorování bezpečnostních událostí a jejich zápis do logu aplikace nebo systému. Jedná se o oblast často podceňovanou, ale osobně si myslím, že velmi důležitou. Zvláště v dnešní době častých bezpečnostních útoků je dobré logovat bezpečnostní záležitosti. Nastavení auditing přitom není vůbec složité. Proto mi pro výklad stačilo vysvětlit princip a popsat možnosti nastavení. Přidal jsem i ukázku její konfigurace, a to opět jak v kódu, tak v konfiguračním souboru aplikace.

```
<behaviors>
  <behavior name="mojeAuditBehavior">
    <serviceSecurityAudit
          auditLogLocation="Application"
          messageAuthenticationAuditLevel="SuccessOrFailure"
          serviceAuthorizationAuditLevel="None"
          suppressAuditFailure="true"/>
  </behavior>
</behaviors>
```

Příklad 21: Konfigurace Auditing

3.11.3 Certifikáty pro zabezpečenou komunikaci

Certifikáty patří do oblasti zabezpečení asi tak jako automobil do dopravních prostředků. Je to jedna z jejich důležitých podmnožin. Řekl bych, že více než polovina všech příkladů zabezpečených aplikací ve WCF, se kterými jsem se setkal, používala certifikáty, ať už na autentizaci klienta či serveru, nebo na komunikaci prostřednictvím HTTPS. Považoval jsem tedy za nezbytnost, seznámit s nimi blíže i čtenáře. Podkapitolu jsem použil především pro vysvětlení zabezpečení HTTPS komunikace, prostřednictvím SSL certifikátu, což je také v praxi velmi časté. Poznatky z ní jsou ale použitelné i pro autentizaci služby certifikátem, což předvedu ve výukovém příkladu. Nyní ale zpátky k vlastnímu rozčlenění podkapitoly, které jsem volil chronologicky, tak jak by postupoval samotný vývojář.

a) Získání testovacího certifikátu

Ačkoli v reálném použití se získává certifikát z nějaké důvěryhodné certifikační společnosti, v příručce ukazuji způsob jiný: získávání testovacího certifikátu s použitím utility makecert. Je to z toho důvodu, že ověřené certifikáty nejsou zdarma a těžko můžu vyžadovat, aby si je čtenář jen pro účely výuky pořídil. Navíc to, že se jedná o nepodepsaný certifikát nevadí, protože pokud čtenář provede určité zásahy, které uvádím, tak na samotné funkci certifikátu se nic radikálně nezmění.

b) Zobrazení a řízení certifikátů

Po vytvoření certifikátu je ho možné prohlížet, exportovat, mazat atd.. Lze to provádět několika způsoby. V této části popisuji dva. Asi nejjednodušší je použít snap-in modul pro certifikáty Microsoft Management Console. Proto tento způsob upřednostňuji. Mnoho zájemců se ale už asi setkalo s certifikáty v prohlížeči, proto jsem přidal i odstavec, kde je mohou nalézt v Internet Exploreru. Také můžeme spravovat certifikáty prostřednictvím utility certmgr, této možnosti se ještě letmo dotknu ve vlastním výukovém příkladu lekce.

c) Registrace SSL Certifikátu pro vlastní Server a pro IIS

Jak jsem již zmiňoval v úvodu, byla tato část určena především pro zabezpečenou HTTPS komunikaci, neboť pro ni se certifikáty v internetu velmi často používají. Na závěr tedy vysvětluji, jak je použít pro vlastní server (služba hostovaná ve vlastní kódu), ale i pro IIS. Konfigurace se liší v závislosti na operačním systému, a proto jsem uvedl postup pro uživatele obou nejpoužívanějších Windows systémů - XP a Vista.

3.11.4 Scénáře použití

WCF aplikace, pokud se bavíme o zabezpečení, bychom mohli shrnout do několika oblastí použití. Každá má své typické rysy a většinu WCF aplikaci bychom mohli zařadit do jedné či více z těchto skupin.

- Intranetové aplikace
- Internetové aplikace
- Business-to-business aplikace
- Anonymní aplikace
- Aplikace bez zabezpečení

V ([2] kap. 10.12) jsem nalezl přínosnou tabulku, která pomůže méně znalému vývojáři správně vybrat Binding a také způsob zabezpečení pro jeho aplikaci podle toho, do jakého scénáře použití patří. Samozřejmě, že nelze věc automatizovat a každá aplikace může mít svoje specifické prvky, které se nehodí do žádné skupiny. Popřípadě se může jednat o aplikaci zcela se vymykající těmto scénářům. Tabulka je ale určena pro rychlou orientaci a rozhodně to nemá být přesně daný návod.

Aspekt	Intranet	Internet	B2B	Aonymní	Žádný
Transport	Ano	Ne	Ne	Ne	Ne
Message	Ne	Ano	Ano	Ano	Ne
Service auentizace	Windows	Certificate	Certificate	Certificate	Ne
Client autentizace	Windows	ASP.NET	Certificate	Ne	Ne
Autorizace	Windows	ASP.NET	Ne/ASP.NET	Ne	Ne
Impersonation	Ano	Ne	Ne	Ne	Ne

Tabulka 3: Aspekty scénářů zabezpečení

([2], kap. 10.12)

3.11.5 Příklad zabezpečené internetové WCF aplikace

Volba příkladu mi dala velkou práci. Původně jsem chtěl vytvořit více příkladů. Abych ale ukázal většinu nejpoužívanějších aplikací, musel bych pro každý scénář (viz 3.11.4) vytvořit několik vzorových ukázek, což nebylo v mých silách a rozsah takového počinu by vydal na samostatnou práci. Od toho jsem tedy upustil. Místo toho jsem se rozhodl, že vytvořím pouze jednu ukázku, ale natolik komplexní, aby pokryla co nejvíce bezpečnostních aspektů WCF a také, aby byla dostatečně praktická. Vybral jsem zabezpečenou internetovou aplikaci hostovanou prostřednictvím IIS. To je příklad zabezpečení, který mě osobně zajímá nejvíce, a tak jsem také doufal, že by mohl být užitečný i pro většinu čtenářů.

Jedná se o WCF službu, hostovanou prostřednictvím IIS a klienta, který se ke službě připojuje prostřednictvím proxy třídy. Služba a klient používají message zabezpečení. IIS je kvůli obraně proti spoofingu autentizován pomocí

X.509 certifikátu. Klient se autentizuje pomocí jména a hesla. Autorizace klienta je provedena prostřednictvím autorizačních mechanismů, založených na ASP.NET, konkrétně SqlRoleProvider a SqlMembershipProvider. Také je použito auditování zpráv do aplikačního logu pro neplatné přístupy. Jako podklad jsem použil starší příklad z prvních třech lekcí, tedy službu PočitadlaSpotřeby, IIS hosta a WPF klienta. Na první pohled vypadají oba projekty stejně, jen teď u klienta přibila dvě textová pole pro zadání jména a hesla. Rozdíl je ovšem v kódu, který nyní navíc implementuje bezpečnostní politiku. Výhoda, využití staršího příkladu, je především pro ty čtenáře, které ho dříve zkoušeli. Ti nyní vidí, jaké zásahy musí v aplikaci udělat, aby fungovalo zabezpečení.

V ukázce se tedy zájemci naučí používat mnoho bezpečnostních aspektů, o kterých se psalo v teorii. Jako je vytváření a implementace certifikátu pro zabezpečení služby a konfigurace veřejné části certifikátu u klienta. Dále message úroveň zabezpečení s autentizací jménem a heslem. Definování rolí a uživatelů pomocí SQL providera. Nastavení autorizačních práv pro jednotlivé operace na službě. A nakonec monitorování bezpečnostních událostí. Samotný příklad je tedy značně rozsáhlý. V příručce vysvětluji bod za bodem, jak postupovat k jeho vytvoření. Čtenář má samozřejmě k dispozici zdrojový kód. Ten mu ovšem bez postupu moc neprozradí.

Pro představu, o jak rozsáhlý příklad se jedná, přidávám výpis jednotlivých bodů k jeho realizaci. Rád bych se poté ještě zastavil u některých bodů.

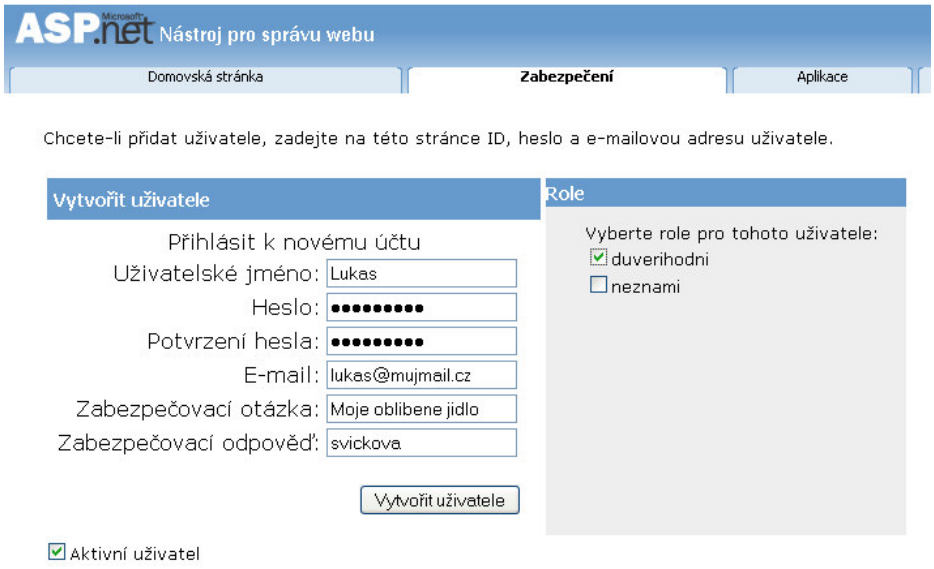
1) Vytvoření WCF služby: *probíráno v první lekci*
2) Nakonfigurování pravidel přístupu k jednotlivým metodám služby.
3) Vytvoření IIS hosting s virtuálním adresářem pro službu: *probíráno v druhé lekci.*
4) Vytvoření certifikátu pro autentizaci služby.
5) Exportování WCF certifikátu služby a importování ho do úložiště klientovi.
6) Zajištění ASP.NET účtu přístup k certifikátu.
7) Nakonfigurování Binding služby s message zabezpečení a username autentizací.
8) Přidání do konfigurace služby certifikát pro autentizaci služby.
9) Vytvoření SQL Server databáze pro použití SQL Membership a Role Providera.
10) Přiřazení práv k databázi pro WCF service proces.
11) Nakonfigurování ConnectionStringu, SqlRoleProvider a SqlMembershipProvider.
12) Vytvoření rolí a přidání uživatelů v ASP.NET konfiguraci.
13) Nastavení logování bezpečnostních událostí.
14) Vytvoření Klientské aplikace: *probíráno ve třetí lekci.*
15) Generování proxy třídy a doladění klientské konfigurace: *probíráno ve třetí lekci.*
16) Přidání informací o certifikátu služby do konfiguračního souboru klienta.
17) Nastavení klientských autentizací údajů: Username a password.
18) Otestování zabezpečení Klienta a WCF služby.

3.11.5.1 Zajištění přístupu účtu k certifikátu a SQL databázi

Jedna z důležitých věcí, co je potřeba pro správnou funkčnost zabezpečení služby hostované v IIS udělat, je zajistit přístup ASPNET účtu k databázi a také mu umožnit čtení certifikátu. Když jsem příklad připravoval, tak jsem si tyto, řekli bychom drobná fakta, neuvědomil a dostal jsem se do těžko odhalitelné situace, kdy jsem měl konfiguraci sice správně, ale přesto mi zabezpečení nefungovalo. Služba totiž běží v případě IIS hostingu pod ASPNET účtem, což je účet ASP.NET pracovního procesu. Tento účet nemá v první řadě defaultně nastaven přístup pro čtení certifikátu potřebného pro autentizaci služby certifikátem a za druhé nemá přístup k aspnetdb SQL databázi, kterou používá služba k autorizaci pomocí Sql providerů. Trvalo mi delší dobu, než jsem zjistil charakter problému a našel na [18] a [19] jeho řešení. Abych čtenáře ušetřil této situace, přidal jsem do návodu k příkladu i body 6 a 10, které tuto věc popisují.

3.11.5.2 Vytvoření rolí a uživatelů v ASP.NET konfiguraci

Pro tento příklad jsem volil způsob autorizace na základě ASP.NET rolí. Existují i jiné způsoby autorizace, na základě Windows skupin anebo vlastního úložiště. Já jsem ovšem vybral ASP.NET role z důvodu možnosti použití v prostředí internetu, kde nemají Windows skupiny význam. A také protože pro vytváření těchto rolí existuje pro čtenáře jednoduchý a intuitivní způsob, kterým je použití ASP.NET nástroje pro správu webu. Nástroj pro správu webu je webová stránka, kterou si uživatel může v prohlížeči vyvolat rovnou z Visual Studia, a pak si v ní nastavovat různé aspekty aplikace. Nám se hodila možnost vytváření rolí a uživatelů v záložce zabezpečení.



Obrázek 11: Vytvoření uživatele promocí Nástroje pro správu webu

3.11.5.3 Nastavení pravidel přístupu metodám služby

Vytvoření uživatelů a jejich přiřazení do rolí je jen část toho, co má vývojář udělat. Hlavně musí nakonfigurovat pravidla přístupu k jednotlivým metodám služby pro autorizaci uživatelů, což vysvětluji ve druhém bodě. Pro službu jsem vymyslel dvě role duverihodni a neznami (viz obrázek 11). Všem metodám služby jsem povolil přístup z role duverihodni a druhé metodě i z role neznámí, abych mohl otestovat více možností.

```
[PrincipalPermission(SecurityAction.Demand, Role =
"duverihodni" )]
public float prumSpotreba(int vzdalenost, int palivo)
{...}
[PrincipalPermission(SecurityAction.Demand, Role =
"duverihodni")]
[PrincipalPermission(SecurityAction.Demand, Role = "neznami")]
public float maxVzdalenost(int spotreba, int palivo)
{...}
[PrincipalPermission(SecurityAction.Demand, Role =
"duverihodni")]
public float potrebPalivo(int vzdalenost, int spotreba)
{...}
```

Příklad 22: Nastavení autorizačních práv pro službu PocitadloSpotreby

3.11.5.4 Nakonfigurování služby pro účely zabezpečení

Většinu zásahů v případě zabezpečení provádí uživatel v konfiguračním souboru. Zásahů je více, a proto konfiguraci ukazuji čtenáři postupně v bodech, v pořadí, jakém bych konfiguroval zabezpečení služby sám. Pro autorizaci přes ASP.NET role je potřeba nakonfigurovat ConnectionString, SqlRoleProvidera a SqlMembershipProvidera (bod 11). Pro umožnění logování bezpečnostních událostí se musí nastavit auditing (viz příklad 21). Uživatel nesmí zapomenout ani na konfiguraci certifikátu služby (bod 8), aby mohl službu autentizovat u klienta. Ale především je nezbytné určení způsobu autentizace klientů v konfiguraci Binding a mód zabezpečení, kterým je v našem případě Message. To je první věc, kterou by měl programátor konfigurovat, a proto ji uvádím již v bodě 7, hned za získáním certifikátu služby. Přikládám ukázku tohoto nastavení.

```
<bindings>
  <wsHttpBinding>
    <binding name="WSHttpBinding_MericSpotreby">
      <security mode="Message">
        <message clientCredentialType="UserName"/>
      </security>
    </binding>
  </wsHttpBinding>
</bindings>
```

Příklad 23: Nakonfigurování způsobu zabezpečení a autentizace uživatelů

3.11.5.5 Nastavení klienta pro zabezpečenou komunikaci

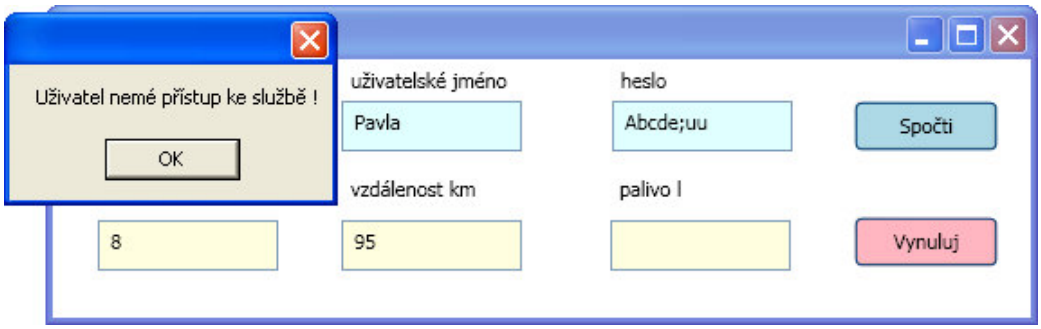
Klientská aplikace se oproti příkladu ze třetí lekce příliš nezměnila. Důležité je přidat před voláním jednotlivých metody proxy třídy klientské autentizační údaje: jméno a heslo. Na tento kód kladu důraz, aby si ho čtenář zapamatoval, neboť je pro klienta stěžejní.

```
PocitadloSpotrebyClient proxy = new PocitadloSpotrebyClient();
try
{
    proxy.ClientCredentials.UserName.UserName =
    jmenoTextBox.Text;
    proxy.ClientCredentials.UserName.Password =
    hesloTextBox.Text;
```

Příklad 24: Předání klientských autentizačních údajů

3.11.5.6 Otestování zabezpečení Klienta a Služby

Na úplný závěr lekce přidávám ukázky výstupů pro různé scénáře, které mohou v zabezpečené komunikaci nastat. Jako je úspěšná autentizace i autorizace, úspěšná autentizace a neúspěšná autorizace, neúspěšná autentizace. Na obrázku 12 je vidět úspěšná autentizace, ale neúspěšná autorizace uživatele Pavly, která patří do role neznami. Pavla se snaží přistupovat ke třetí metodě služby `potrebPalivo`, která má ovšem nastavena pravidla přístupu jen pro uživatele role duverihodni, čili tento pokus skončí bezpečnostní výjimkou.



Obrázek 12: Ukázka neúspěšné autorizace uživatele

4 Závěr

Cílem mojí bakalářské práce bylo seznámit čtenáře s novou komunikační technologií Windows Communication Foundation. Způsob, jakým jsem se toho zhostil, vycházel ze situace v oblasti českých zdrojů, týkajících se WCF. Z té vyplynulo, že materiálů je málo a jsou velmi povrchní. To mi bylo motivací k tvorbě vlastní práce, kterou jsem se snažil situaci částečně vylepšit.

Zaměřil jsem se na začínající zájemce, ale myslel jsem i na pokročilejší a podle toho jsem také volil koncepci práce. Vytvořil jsem příručku členěnou do lekcí, pro lepší orientaci a možnost vlastního rozvržení organizace výuky čtenáře. Volil jsem takový způsob tvorby lekcí, jaký nejvíce vyhovuje mně. A to vysvětlení problematiky nejen pomocí teorie, ale i na příkladu, který je bod po bodu rozebrán a popsán. Do práce jsem též zakomponoval videoukázku pro názorné předvedení jednoho z příkladů, který by se hůře popisoval slovně.

Co se týče počtu a volby probíraných rysů jazyka, tak mé počáteční požadavky nebyly naplněny v plné míře. Chtěl jsem původně projít většinu oblastí WCF, a to alespoň částečně. Avšak z časových a i rozsahových důvodů jsem to nemohl uskutečnit a raději jsem se tedy zaměřil jen na některé části technologie, které jsem probral podrobněji. Myslím, že pro začátečníky jsem tento úkol splnil dle předpokladů. V prvních třech lekcích, určených jim, jsem je postupně provedl vytvořením a hostováním jednoduché služby a klienta. Pro pokročilejší zájemce už bylo naplnění cílů složitější. Vybral jsem pro ně jen některá témata, především ta, která jsem považoval za nejvíce užitečná, například zachytávání výjimek a zvláště, v dnešní době frekventované téma, zabezpečení. Přesto příručka stále nechává prostor pro další rozšíření.

Volba výukových příkladů patří k stěžejním výstupům práce a hraje pro pochopení problematiky klíčovou roli. Dle mého názoru se mi podařilo vymyslet a navrhnout příklady dostatečně srozumitelné a ve velké míře i praktické. Neméně důležitý je způsob jejich popsání ve výukových kapitolách. Příklad jsem rozebíral postupně, tak jak vznikal. V každém bodu jsem pak vysvětloval příslušné funkce kódu. Tuto metodiku považuji za velmi zdařilou část práce. Také videoukázka byla zajímavým zpestřením příručky a dospěl jsem názoru, že je užitečná především pro ukázání postupů anebo výstupů, které se mění v čase. Její tvorba pro mě ale byla náročnější, než jsem původně očekával, a to je také důvod, proč jsem vytvořil jen jednu.

Na závěr bych chtěl dodat, že záměr práce byl splněn. Podařilo se mi ukázat jak programovat v mnoha oblastech technologie WCF. Ovšem konečné posouzení toho, jestli bude výuka srozumitelná a efektivní a hlavně zda přinese užitek, musí zhodnotit až samotní čtenáři.

Literatura

[1] KLEIN, Scott. *Professional WCF Programming : .NET Development with the Windows Communication Foundation*. Indianapolis (Indiana) : Wiley Publishing, Inc., 2007. 430 s. ISBN 978-0-470-08984-2.

[2] LÖWY, Juval. *Programming WCF Services : Building SOAs with Windows Communication Foundation*. Sebastopol (California) : O'Reilly Media, Inc., 2007. 634 s. ISBN 978-0-596-52699-3.

[3] PEIRIS, Chris, et al. *Pro WCF : Practical Microsoft SOA Implementation*. Berkeley (California) : Apress, Inc., 2007. 512 s. ISBN 978-1-59059-702-6.

[4] RESNICK, Steve, CRANE, Richard, BOWEN, Chris. *Essential Windows Communication Foundation : For .NET Framework 3.5*. Boston (Massachusetts) : Addison-Wesley Professional, 2008. 608 s. ISBN 978-0-321-44006-8.

[5] SHARP, John. *Microsoft Windows Communication Foundation Step by Step*. Redmond (Washington) : Microsoft Press, 2007. 448 s. ISBN 978-0-735-62336-1.

[6] SMITH, Justin. *Inside Microsoft Windows Communication Foundation*. Redmond (Washington) : Microsoft Press, 2007. 304 s. ISBN 978-0-735-62306-4.

[7] KOŠŤÁL, Marián. *WCF Instancing* [online]. 2007 [cit. 2008-12-20]. Dostupný z WWW: <http://www.vyvojar.cz/Articles/462-wcf-instancing.aspx>.

[8] Microsoft Corporation. *Windows Communication Foundation : MSDN* [online]. Microsoft Corporation, c2007 [cit. 2007-11-20]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms735119.aspx>.

[9] Microsoft. *WCF Security Fundamentals* [online]. 2009 [cit. 2009-03-20]. Dostupný z WWW:<http://msdn.microsoft.com/en-us/library/cc949042.aspx>.

[10] Microsoft. *WCF Security Overview* [online]. 2007 [cit. 2009-03-20]. Dostupný z WWW:<http://msdn.microsoft.com/en-us/library/ms735093.aspx>.

[11] Microsoft. *WCF Auditing Security Events* [online]. 2007 [cit. 2009-03-20]. Dostupný z WWW:<http://msdn.microsoft.com/en-us/library/ms731669.aspx>.

[12] Microsoft. *WCF Specifying and Handling Faults in Contracts and Services* [online]. 2007 [cit. 2009-03-20]. Dostupný z WWW:<http://msdn.microsoft.com/en-us/library/ms733721.aspx>.

[13] Microsoft. *Configuring System-Provided Bindings* [online]. 2007 [cit. 2008-12-10]. Dostupný z WWW:<http://msdn.microsoft.com/en-us/library/ms731092.aspx>.

[14] LARYŠ, Kryštof. *2. díl: contract, binding, service behavior : WCF pro začátečníky* [online]. 13. února 2009 [cit. 2009-03-10]. Dostupný z WWW: <http://www.netstudent.cz/Články/tabid/56/articleType/ArticleView/ArticleID/226/PageID/209/Default.aspx>.

[15] RIGSBY, Dan. *WCF Metadata* [online]. 27.5.2008 [cit. 2009-01-10]. Dostupný z WWW: <http://www.danrigsby.com/blog/index.php/2008/05/27/wcf-metadata>.

[16] RIGSBY, Dan. *Sessions, Instancing, and Concurrency* [online]. 2007 [cit. 2009-01-12]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/ms731193.aspx>.

[17] Microsoft. *WCF Sending and Receiving Faults* [online]. 2007 [cit. 2009-03-20]. Dostupný z WWW:<http://msdn.microsoft.com/en-us/library/ms732013.aspx>.

[18] Microsoft. *WCF FindPrivateKey* [online]. 2007 [cit. 2009-03-24]. Dostupný z WWW:<http://msdn.microsoft.com/en-us/library/aa717039.aspx>.

[19] Microsoft. *WCF How to: Use the SQL Server Role Provider with Username Authentication in WCF Calling from Windows Forms* [online]. 2009 [cit. 2009-03-24]. Dostupný z WWW:<http://msdn.microsoft.com/en-us/library/cc949027.aspx>.

[20] Microsoft. *WCF WCF Client Overview* [online]. 2007 [cit. 2009-01-22]. Dostupný z WWW:<http://msdn.microsoft.com/en-us/library/ms735103.aspx>.

Přílohy

A Obsah přiloženého CD

Příručka WCF

- Úvod (WCF.pdf)
- 1. Vytvoření WCF služby (WCF_1.pdf)
- 2. Hostování Služby (WCF_2.pdf)
- 3. Vytvoření WCF klienta (WCF_3.pdf)
- 4. Instance management a concurrency (WCF_4.pdf)
- 5. Throttling (WCF_5.pdf)
- 6. Chyby a výjimky (WCF_6.pdf)
- 7. Zabezpečení (WCF_7.pdf)

Příklady

- MericSpotrebyService (*pro 1. lekci*)
- MericSpotrebyHost *(pro 2. lekci)*
- MericSpotrebyHostIIS *(pro 2. lekci)*
- MericSpotrebyClient *(pro 3. lekci*)
- MericSpotrebyClientCode *(pro 3. lekci)*
- Instancing *(pro 4. lekci)*
  - InstanceClient
  - InstanceService
- Throttling *(pro 5. lekci)*
  - ThrottlingClient
  - ThrottlingService
- Faults *(pro 6. lekci)*
  - FaultsClient
  - FaultsService
- SecurityIIS *(pro 7. lekci)*
  - MericSpotrebyServiceSecurity
  - SecurityClient
  - SecurityIISHost

Videoukázka

- Instancing.avi *(pro 4. lekci)*