Tvorba aplikací typu klient/server pomocí Windows Communication Foundation

Bakalářská práce
Petr Kafka
Vedoucí bakalářské práce: Ing. Václav Novák, Csc.
Jihočeská univerzita v Českých Budějovicích
Pedagogická fakulta
Katedra informatiky
2010

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 22.4.2010

Anotace

Tato bakalářská práce se zabývá využitím technologie Windows Communication Foundation pro tvorbu aplikací typu klient/server. Cílem práce je vytvoření česky psaných výukových materiálů, pomocí kterých se čtenář seznámí s vytvářením klientů a služeb WCF. Jejich součástí je množství ukázkových příkladů použitých k vysvětlení problematiky.

Abstract

This bachelor work deals with usage of Windows Communication Foundation technology to create application of client/server type. The main goal is to create learning materials, which will familiarize reader with creating WCF clients and services, written in Czech language. These materials contain a number of samples used to explain the problems.

Obsah

Seznam obrázků 6
Seznam příkladů 7
Seznam tabulek 8
1 Úvod 9
1.1 Úvod 9
1.2 Cíle práce 10
1.3 Členění práce 10
2 Technologie Windows Communication Foundation 12
2.1 Distribuované programování v minulosti 12
2.2 Hlavní rysy WCF 13
2.2.1 Sjednocení předchozích technologií 14
2.2.2 Interoperabilita napříč platformami 15
2.2.3 Orientace na služby 16
2.2.4 Způsoby programování a nastavení ve WCF 19
2.3 Služby 20
2.3.1 Endpoint 22
2.3.2 Address 24
2.3.3 Binding 26
2.3.4 Contract 29
2.3.5 Metadata 35
2.4 Hostování služeb 37
2.4.1 Host architektura 38
2.4.2 Způsoby hostování 39
2.5 Klienti WCF 41
2.5.1 Architektura klienta 41
2.5.2 Komunikace mezi klientem a službou 42
3 Výukové materiály WCF 45
3.1 Současný stav problematiky 45
3.2 Použitá metodika 46
3.3 Úvodní kapitola 47
3.4 Vytvoření WCF služby 48
3.4.1 Endpoint 48
3.4.2 Výběr Binding 49
3.5 Hostování WCF služby 50

3.5.1 Výběr hostovacího prostředí 50
3.5.2 Reference na službu 52
3.6 Vytvoření WCF klienta 53
3.6.1 Vytvoření klientské aplikace 53
3.6.2 Postup při tvorbě WCF klienta 53
3.7 Práce s výjimkami a chybami 56
3.7.1 Ukázkový příklad práce s výjimkami a chybami 56
3.7.2 Služba ObjednávkaChyba 57
3.7.2 Klient služby ObjednávkaChyba 58
3.7.2.1 Správné uzavření klienta 59
4 Závěr 61
Přehled použité literatury 63
Přílohy 67
A Obsah přiloženého CD 67

Seznam obrázků

Obrázek 1: Sjednocení předchozích technologií 15
Obrázek 2: SOA 17
Obrázek 3: Služba 20
Obrázek 4: Možnosti komunikace služeb a klientů 21
Obrázek 5: Proces (převzato z [6]) 39
Obrázek 6: Komunikace klienta a služby (převzato z [2], kap 1.11) 43
Obrázek 7: Výběr ze System Binding 49

Seznam příkladů

Příklad 1: Konfigurace endpointu – administrativně 23
Příklad 2: Konfigurace endpointu – programově 23
Příklad 3: Ukázka Service Contract 31
Příklad 4: Ukázka Data Contract 32
Příklad 5: Ukázka Message Contract 33
Příklad 6: Ukázka objektu pro Fault Contract 34
Příklad 7: Ukázka Fault Contract 34
Příklad 8: MEX endpoint a HTTP-GET 37
Příklad 9: Self hosting, počátek hostování 51
Příklad 10: Vytvoření klienta pomocí vygenerované proxy 54
Příklad 11: Vytvoření klienta pomocí *ChannelFactory* 55
Příklad 12: Deklarace Fault Contract *ObjednavkaFault* 57
Příklad 13: Obsluha výjimek 58
Příklad 14: Korektní uzavření klienta 60

Seznam tabulek

Tabulka 1: Přehled systémových Binding..... 28

1 Úvod

1.1 Úvod

Service Oriented Architecture (SOA) patří k největším tématům v oblasti vývoje softwaru dnešní doby. Jedná se způsob návrhu a vývoje klient/server systémů, které se velmi často označují jako distribuované systémy. Nejnovější přírůstek společnosti Microsoft v oblasti vývoje distribuovaný systémů nese označení Windows Comunication Foundation (dále jen WCF).

WCF je programovací model pro vývoj a nasazení distribuovaných řešení pro platformu Windows. Ačkoliv je WCF primárně určeno pro platformu Windows, obsahuje implementaci souborů specifikací pro vzájemnou spolupráci služeb, typové konverze, logické uspořádání a rozmanitou správu protokolů. Díky tomu WCF poskytuje potřebnou interoperabilitu pro komunikaci s ostatními platformami.

Toto téma jsem si vybral jelikož jsem se o WCF již okrajově zajímal a proto jsem uvítal možnost prohloubit své znalosti, stejně jako možnost využití znalostí které jsem již měl.

1.2 Cíle práce

Cílem této práce je prozkoumání technologie WCF a následné vytvoření výukového materiálu této technologie. Výukový materiál má zájemcům pomoci proniknout do základů WCF a tvorby distribuovaných aplikací za využití prostředků, které WCF poskytuje. Výukový materiál, rozdělený do jednotlivých lekcí, obsahuje řadu ukázkových příkladů společně s teoretickým základem potřebným pro pochopení těchto příkladů. Stěžejní částí každé kapitoly jsou právě ony ukázkové příklady, jelikož jsem toho názor, že vysvětlení na příkladu a popis jak příklad vznikl jsou pro pochopení dané tématiky lepší než výklad strohé teorie.

Výukové materiály jsou určeny pro čtenáře, kteří mají s WCF malou, nebo žádnou zkušenost. Pro správné pochopení očekávám od čtenářů základní znalost programováni pod platformou .Net a programovacího jazyka C#. Jelikož jsou výukové materiály určeny pro nováčky v oblasti WCF, jsou zaměřeny na základní prvky WCF, jako je tvorba jednoduché služby a tvorba klientské aplikace konzumující tuto službu.

1.3 Členění práce

Samotná práce je členěna do několika částí. První část obsahuje popis technologie WCF, její základní vlastnosti a výhody. Dále následuje část popisující samotné výukové materiály. Zde je popsáno co obsahují

jednotlivé kapitoly, postup využitý při jejich sestavování a upozorněni na důležité, nebo zajímavé části těchto kapitol. Samotné kapitoly jsou umístěny na přiloženém CD. Na závěr shrnuji obsah práce a přikládám seznam literatury a příloh.

Výukové materiály jsou sestaveny formou příručky členěné na kapitoly. Každá kapitola je uceleným útvarem, který se zabývá jedním aspektem WCF. Součástí jednotlivých kapitol jsou ukázkové příklady.

2 Technologie Windows Communication Foundation

2.1 Distribuované programování v minulosti

Koncept distribuovaného programování není nikterak nový, popravdě řečeno tento koncept existuje již velmi dlouhou dobu. Během let vznikaly různé technologie, které se tímto konceptem zabývaly. Mezi těmito technologiemi je i několik od firmy Microsoft. V minulosti bylo možno pro tvorbu distribuovaných aplikací používat následující technologie této firmy :

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

Každá z těchto technologií s sebou nese jisté klady a zápory. Například ASMX a WSE poskytují možnost přístupu z rozdílných platforem. Enterprise Services a .Net Remoting vynikají výkonem, MSMQ zase spolehlivostí doručení. Všechny mají velké možnosti rozšíření ale jejich vzájemná nekompatibilita je velký problém. Pokaždé když chtěl vývojář začít s vývojem distribuovaného systému musel si klást otázku: „Která z těchto technologií nejjednodušeji a nejefektivněji vyřeší můj problém?“. Poté co si jednu z nich vybere, musí se jí držet po celou dobu vývoje, nemůže jednoduše přejí na jinou. Při změně požadavků na systém, změnu platformy, nebo transportního protokolu docházelo k velkým zásahům do aplikace a mnohdy i ke změně technologie a začínání od začátku.

Tyto nedostatky si firma Microsoft velmi dobře uvědomovala a proto v roce 2006 byla spolu s verzí 3.0 .Net Framework představena technologie Windows Communication Foundation jako jedna z novinek, které tato verze přinesla. WCF přináší nový pohled na tvorbu distribuovaných aplikací. Sjednocuje dříve zmíněné technologie, přebírá jejich silné stránky a odstraňuje jejich nedostatky.

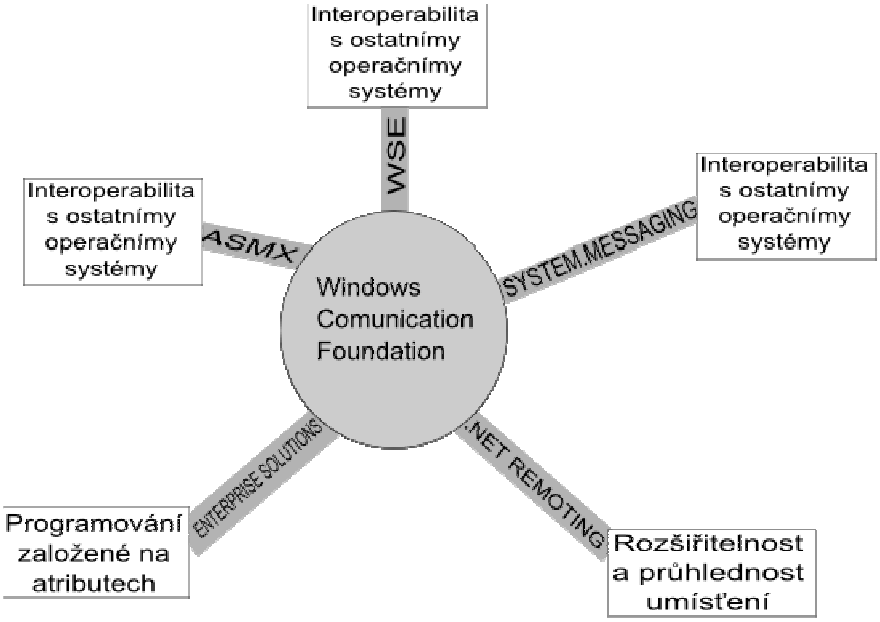
2.2 Hlavní rysy WCF

WCF není pouze další způsob tvorby distribuovaných aplikací, ale přináší řadu výhod oproti svým předchůdcům. Jako hlavní rysy jsou uváděny (viz [6], str. 29):

- Sjednocení předchozích technologií
- Interoperabilita napříč platformami
- Orientace na služby

2.2.1 Sjednocení předchozích technologií

Podnikové systémy v dnešní době využívají velké množství distribuovaných technologií. Každá z těchto technologií má vlastní způsob provádění specifických úkonů, vlastní roli a specifický význam. Nehledě na to, že každá z těchto technologií je založena na jiném programovacím modelu. Například, když je vytvářena aplikace komunikující pomocí http, bude zapotřebí změna aplikace při přechodu na TCP. Toto vytváří problémy pro vývojáře kteří se musejí neustále učit nová API pro tvorbu rozdílných distribuovaných komponent. Neustálý boj mezi distribuovanými technologiemi vedl k otázce, která technologie je z dlouhodobého hlediska pro tvorbu distribuovaných aplikací nejlepší. Jedna ze zajímavých otázek je, proč nemít pouze jednu technologii použitelnou pro každou situaci. WCF je možno označit jako řešení této otázky. WCF přináší souhrn existujících distribuovaných řešení pod jednou střechou a umožňuje využití jediného, čistého API .



Obrázek 1: Sjednocení předchozích technologií

2.2.2 Interoperabilita napříč platformami

Mnoho velkých softwarových společností používá vlastní protokoly, které jsou velmi těsně svázány se specifickou technologií. To může vést k problémům při spolupráci s jinými aplikacemi využívajícími jiné technologie. Tato nekompatibilita může nastávat i mezi vnitropodnikovými systémy, které se skládají z mnoha částí. Schopnost vzájemného propojení těchto částí se stala velmi důležitou potřebou pro mnoho společností.

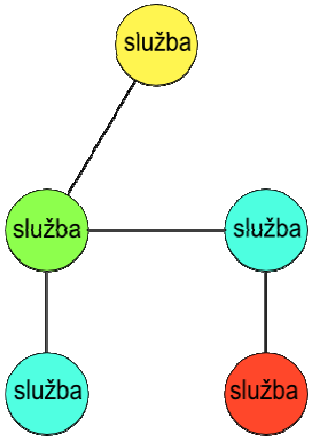
Tato potřeba vedla velké softwarové společnosti k vytvoření Web Service Interoperability organization (WS-I), která vyvinula řadu

specifikací pro vzájemnou komunikaci aplikací využívajících rozdílných technologií. WCF rozumí těmto specifikacím a proto umožňuje bezproblémovou komunikaci napříč platformami.

WCF využívá SOAP protokol, což je otevřený standard pro výměnu dat mezi aplikacemi využívajícímu různé technologie a operační systémy.

2.2.3 Orientace na služby

WCF je programovací model plně postavený na service oriented architecture (SOA). SOA není technologie, ale spíše návrhový vzor. Jedná se o souhrn nejlepších praktik pro tvorbu moderních distribuovaných aplikací. Nikoli o nový koncept, ale o myšlenku již několik let starou. Jednoduše řečeno SOA je způsob návrhu distribuovaných aplikací pomocí autonomních služeb, které mezi sebou komunikují pomocí zpráv a dobře známých rozhraní.



Obrázek 2: SOA

SOA má čtyři hlavní zásady :

1. Služby mají explicitní hranice

Každá služba je vždy uzavřena hranicemi jako je technologie a umístění. Služba vystavuje svoji funkcionalitu pomocí rozhraní, které popisují jednotlivé operace, datové členy, parametry a návratové hodnoty. Všechny ostatní věci týkající se implementace zůstávají skryty. Služba by neměla zveřejňovat své hranice vystavováním kontraktů a datových typů prozrazujících její technologii a umístění.

2. Služby jsou autonomní

Služba pro svojí existenci nepotřebuje nic od klientů a ostatních služeb. Každá služba musí být vyvíjena a spravována nezávisle na ostatních službách a klientech. Implementační detaily služeb se mohou měnit, ale jejich hranice a kontrakty by měly být zachovány, čímž se zajistí soudržnost se stávajícími klienty.

3. Služby sdílejí kontrakty a schémata, ne typově a technologicky specifická metadata

Všechny informace zveřejňované službou musejí být absolutně technologicky nezávislé a popsané v kontraktu. Služba musí být schopná převést své nativní i vlastní datové typy do neutrální reprezentace a ty popsat v kontraktu. Služba nesdílí technologicky a implementačně závislé informace. Služby nekomunikují pomocí tříd a datových typů, ale pomocí jasně definovaný rozhraní a datových schémat.

4. Služby jsou kompatibilní a založené na politice

Služba zveřejňuje politiku definující její chování a způsob, jak s ní mohou komunikovat klienti. Věci definované v politice (např. spolehlivost doručení, transakce, zabezpečení) by měly být odděleny od implementačních detailů služby. Ne všichni klienti mohou komunikovat se všemi službami. Politika je souborem pravidel podle kterých klient určuje zda může komunikovat se službou. Politika tedy určuje jejich

vzájemnou kompatibilitu. Politika také umožňuje přesunutí služby z jednoho prostředí do druhého bez změny chování služby.

2.2.4 Způsoby programování a nastavení ve WCF

WCF podporuje množství způsobů programování, které si s sebou nesou své klady i zápory. Dobrá vlastnost WCF je možnost dosažení cílů více způsoby. Dva základní způsoby nastavení jsou:

- **Administrativní** způsob nastavení pomocí konfiguračních souborů. Tento způsob je vhodný pro nastavení nebo změnu vlastností služby a klienta po jejich nasazení bez nutnosti rekompilace kódu. Tento způsob však není typově bezpečný a chyby v nastavení budou objeveny až za běhu aplikace.

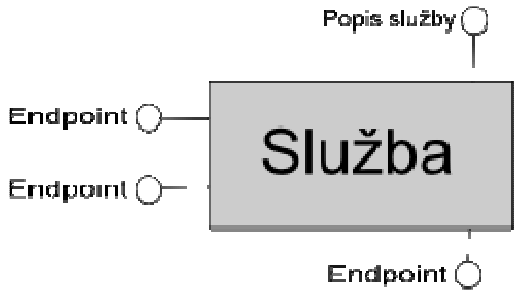
- **Programové nastavení** v samotném kódu programu se používá v případě, kdy je nastavení závislé na běhu programu, na momentálních vstupech, nebo podmínkách, nebo pokud je statické a nikdy se nemění. Po provedení změny v programovém nastavení je třeba program znovu zkompilovat, proto jsou chyby v tomto nastavení patrné již při kompilaci a nikoli až za běhu programu.

Tyto dva způsoby je možno libovolně kombinovat.

2.3 Služby

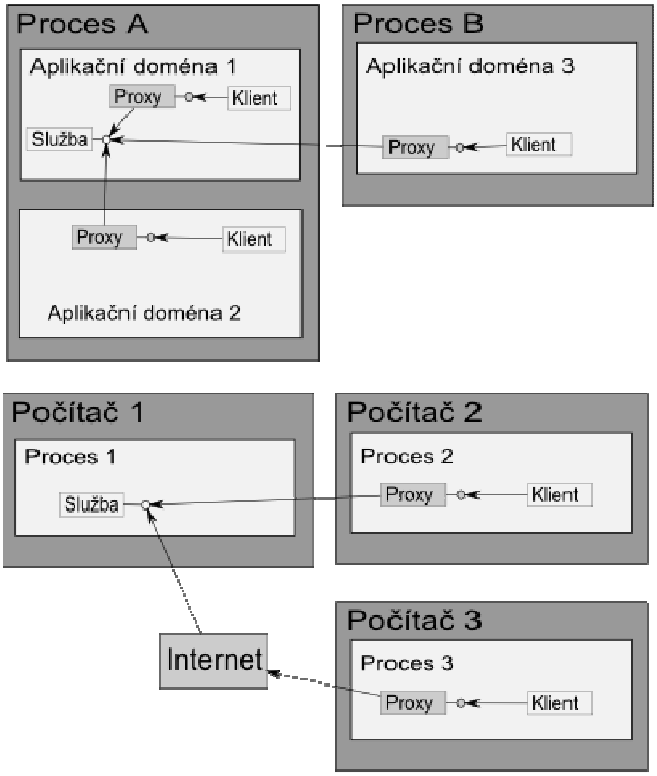
Služby jsou základním stavebním kamenem WCF aplikace. Ve své podstatě je služba systémem jednoho či více endpointů (viz obrázek 3), který poskytuje možnosti klientům. Služba se skládá ze tří základních částí: třídy služby, hostovacího prostředí a jednoho, či více endpointů . Třída služby je samotná implementace služby. Hostovací prostředí je místo, kde služba běží. Endpoint je místo ve službě, které slouží k přijímání a odesílání zpráv.

Klientem služby je v podstatě software konzumující její funkcionalitu. Může jím být doslova cokoliv: Windows Form aplikace, ASP.NET stránka, nebo jiná služba. Komunikace mezi klientem a službou probíhá pomocí SOAP zpráv, díky čemuž mohou WCF služby komunikovat s ne-WCF klienty a WCF klienti komunikovat s ne-WCF službami.



Obrázek 3: Služba

Komunikace mezi službou a klientem nikdy neprobíhá přímo, ani při použití lokálních entit. Klient vždy používá proxy pro předání volání službě. Proxy obsahuje stejné metody jako služba, ale navíc ještě obsahuje metody pro správu proxy. Jelikož je všechna komunikace vedena skrze proxy, služba a klient vždy komunikují stejným způsobem a nezáleží na tom, zda se jedná o lokální komunikaci v rámci jednoho procesu nebo počítače, či o vzdálenou komunikaci např. přes internet.



Obrázek 4: Možnosti komunikace služeb a klientů

2.3.1 Endpoint

Endpoint je jedna z nejdůležitějších částí služby. Veškerá komunikace mezi službou a klientem probíhá právě skrze endpoint. Endpoint je často popisován jako tzv. ABC (Address, Binding, Contract). ABC jsou tři části ze kterých se každý endpoint skládá.

- **Address** : je „Kde“ se endpoint nachází. Definuje kam na síti mají zprávy přicházet, aby je endpoint přijal.
- **Binding** : je „Jak“ endpopint komunikuje. Definuje použitý transportní protokol, kódování a komunikační protokol.
- **Contract** : je „Co“ endpoint umí. Definuje jaké endpoint poskytuje operace.

Endpoint je možno vytvořit dvěma způsoby. Administrativně v konfiguračním souboru, nebo programově v kódu.

```
<system.serviceModel>
    <services>
      <service name="ServiceWCF.SampleService">
        <endpoint
          address ="http://localhost:8000/httpEndpoint"
          binding="wsHttpBinding"
          contract="ServiceWCF.ISampleService">
        </endpoint>
        <endpoint
```

```
          address ="net.tcp://localhost:8001/tcpEndpoint"
          binding="netTcpBinding"
          contract="ServiceWCF.ISampleService">
        </endpoint>
      </service>
    </services>
  </system.serviceModel>
```

Příklad 1: Konfigurace endpointu – administrativně

```
ServiceHost host = new ServiceHost(typeof(SampleService));
Binding wsBinding = new WSHttpBinding();
Binding tcpBinding = new NetTcpBinding();
host.AddServiceEndpoint(typeof(ISampleService),
                  wsBinding,
                  "http://localhost:8000/SampleService");
host.AddServiceEndpoint(typeof(ISampleService),
                  tcpBinding,

"net.tcp://localhost:8001/SampleService");
host.Open();
```

Příklad 2: Konfigurace endpointu – programově

2.3.2 Address

Každý endpoint služby musí mít svojí adresu, která určuje umístění tohoto endpointu. Jedná se o unikátní URI endpointu, které umožňuje klientským aplikacím najít a identifikovat endpoint.

Každá adresa je složena z transportního protokolu/schématu, názvu počítače nebo domény na které služba běží a cesty specifikující endpoint. Jako nepovinný atribut může také obsahovat port, přes který je ke službě přistupováno. Číslo portu následuje za názvem počítače.

protokol//[název počítače|domény][:port]/cesta

WCF poskytuje několik transportních protokolů. Formátováni adresy je pro každý protokol malinko odlišné. Transportní protokoly, které WCF poskytuje jsou :

- TCP
- HTTP
- MSMQ
- Peer2Peer
- ICP

2.3.2.1. HTTP

HTTP adresa požívá jako transportní protokol HTTP. Jedná se asi o nejpoužívanější protokol. Je možno použít jeho zabezpečenou variantu HTTPS. Pokud v adrese neuvedeme port je implicitně použit port 80. Tvar adresy vypadá takto:

http|https://název počítače|domény[:port]/cesta
http://localhost:8080/sluzba
http://localhost:8081/sluzba

2.3.2.2 TCP

TCP adresa používá pro transport net.tcp schéma. Implicitně je port nastaven na port 808. Dvě TCP adresy mohou navzájem sdílet jeden port. Ukázka TCP adresy:

net.tcp://localhost:8082/sluzba
net.tcp://localhost:8082/dalsiSluzba

2.3.2.3 MSMQ

MSMQ adresa používá net.msmq schéma. Formát této adresy se mírně liší od ostatních. Adresa obsahuje transportní protokol, hostname, typ fronty pokud se jedná a privátní frontu a název fronty.

net.msmq://hostname/[private]/queueName
net.msmq://localhost/private/soukromaFronta
net.msmq://localhost/verejnaFronta

2.3.2.4 Peer2Peer

Peer2Peer adresa používá net.p2p schéma. Pro transport je použita peer2peer síť.

net.p2p://SluzbaMesh/SluzbaEndpointJmeno

2.3.2.5 ICP

ICP adresa používá net.pipe schéma. Služby využívající toto schéma mohou přijímat pouze volání z jiného procesu běžícího na stejném počítači. ICP adresa nepoužívá port.

net.pipe://localhost/sluzba

2.3.3 Binding

Binding definuje jak endpoint komunikuje s okolním světem. Každý endpoint musí mít binding. Binding je v podstatě sada vlastností popisujících transportní protokol, kódování, zabezpečení, transakce, spolehlivost a podobně.

WCF obsahuje devět před definovaných binding, jsou to :

- **BasicHttpBinding** : pro komunikaci webových služeb splňujících WS-Basic Profile
- **NetTcpBinding** : zabezpečený a optimalizovaný binding pro komunikaci WCF aplikací (velmi využívané)
- **NetPeerTcpBinding** : vícepočítačová komunikace
- **NetNamedPipeBinding** : komunikace WCF aplikací v rámci jednoho počítače (velmi rychlé)
- **WSHttpBinding** : zabezpečený a interoperabilní binding bez podpory duplex kontraktů
- **WSDualHttpBinding** : zabezpečený a interoperabilní binding s podporou duplex kontraktů
- **MetMsmqBinding** : komunikace pomocí MSMQ (message queue – velmi spolehlivé doručování zpráv, často používané)
- **MsmqIntegrationBinding** : komunikace mezi WCF aplikací a již existující MSMQ aplikací
- **WSFederationHttpBinding** : podpora protokolu WS-Federation (viz [4])

Binding	Interoperability	Security	Session	Transactions	Duplex	Encoding
BasicHttpBinding	Basic Profile 1.1	T,M,X	Ne	Ne		Text
WSHttpBinding	WS	T,M,X	T,RS	Ano		Text
WSDualHttpBinding	WS	M	RS	Ano	Ano	Text
WSFederationHttpBinding	WS-Federation	M,X	RS	Ano	Ne	Text
NetTcpBinding	.NET	T,M,X	T,RS	Ano	Ano	Binary
NetNamedPipeBinding	.NET	T	T,RS	Ano	Ano	Binary
NetMsmqBinding	.NET	T,M	Ne	Ano	Ne	
NetPeerTcpBinding	Peer	T	Ne	Ne	Ano	
MsmqIntegrationBinding	MSMQ	T	Ne	Ano		

Tabulka 1: Přehled systémových Binding

Tabulka 1 zobrazuje přehled systémových binding a jejich vlastností. Interoperability definuje protokol nebo technologii, kterou binding použije k zaručení výměny informací. Security popisuje zabezpečení informačního kanálu, session ukazuje podporu session contract, transaction zda binding podporuje transakce a duplex podporu duplex contract.

Ve většině případů je dostačující použít jeden ze systémových binding tak jak je, nebo po mírné úpravě. Pokud ani upravený binding není dostačující pro potřeby služby je možno vytvořit binding vlastní. Existují tři způsoby jak vytvořit vlastní binding :

- odvozením od třídy System.ServiceModel.Channels.CustomBinding
- odvozením od třídy reprezentující jeden ze systémových binding

- odvozením od třídy System.ServiceModel.Channels.Binding [5]

2.3.4 Contract

Kontrakty ve WCF poskytují interoperabilitu potřebnou ke komunikaci mezi službou a klientem. Contract je platformově nezávislý popis typů operací a struktur, které budou služba a klient používat během komunikace. WCF definuje čtyři základní druhy kontraktů:

- Service contract
- Data contract
- Message contract
- Fault contract

Kontrakty jsou v aplikaci definovány jako CLR typy, proto aby byla zajištěna spolupráce i s ne-WCF klienty, jsou v metadatech representovány jako XML data ve WSDL, SOAP, nebo XSD formátu.

2.3.4.1 Service contract

Service Contract popisuje operace vystavené službou na endpointu. Tyto operace mohou být využity klienty. Service Contract

vystavuje klientům specifické informace umožňující chápání toho co jim může služba poskytnout. Mezi tyto informace patří :

- Datové typy zpráv
- Umístění operací
- Serializační formát pro zajištění úspěšné komunikace
- Seskupování operací
- Message exchange pattern (MEP)

Service Contract je nadefinován aplikací *ServiceContract* atributu na rozhraní. Jednotlivé operace jsou poté označeny aplikací *OparationContract* atributu na jednotlivé metody. Vlastní kód služby poté implementuje toto rozhraní. Service Contract je mapován na Web Service Definition Language (WSDL).

```
[ServiceContract]
public interface IProduktySluzba
{
    [OperationContract]
    List<string> SeznamProduku();
    [OperationContract]
    Produkt GetProdukt(string cisloProduktu);
    [OperationContract]
    int PocetNaSklade(string cisloProduktu);
    [OperationContract]
    bool ObjednejProdukt(ObjednavkaProduktu
objednavkaProduktu);
}
```

Příklad 3: Ukázka Service Contract

2.3.4.2 Data contract

Data Contract popisuje data vyměňovaná mezi službou a klientem. Popisuje strukturu těchto dat. Primitivní typy jako *int*, nebo *string* definuje WCF automaticky a není třeba u nich používat Data Contract. Pro neprimitivní typy je třeba ho definovat. Data Contract se definuje pomocí atributu *DataContract* umístěného před třídou s datovým typem. Jednotlivé členy kontraktu poté označíme atributem *DataMember*.Data Contract je mapován na XML Schéma Definition (XSD).

```
[DataContract]
public class Produkt
{
    [DataMember]
    public string Nazev;
    [DataMember]
    public string CisloProduktu;
    [DataMember]
    public string Popis;
    [DataMember]
    public decimal Cena;
}
```

Příklad 4: Ukázka Data Contract

2.3.4.3 Message contract

Message Contract poskytuje absolutní kontrolu nad formátem SOAP zpráv posílaných a přijímaných službou. Umožňuje nastavovat jaká informace přijde do hlavičky a jaká do těla zprávy, jaká úroveň zabezpečení má být použita na jaký prvek zprávy a podobně. Message Contract, na rozdíl od Service Contract, je určen pro specifikaci operací služby a není určen pro znovu použití a sdílení. Použití Message Contract není ve většině případů zapotřebí, jelikož data kontrakty poskytují většinu kontroly, která je zapotřebí. Pokud je ovšem využití message contractu nutné, je definován umístěním atributu *MessageContract* před třídu s typem zprávy. Jednotlivé části zprávy jsou pak definovány

atributy *MessageHeader* jako hlavička a *MessageBody* jako tělo zprávy. Message Contract je mapován na Simple Object Access Protocol (SOAP) Message.

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

Příklad 5: Ukázka Message Contract

2.3.4.4 Fault contract

Fault Contract definuje, jaké chyby mohou být předávány službou a způsob jejich zasílání klientovi. Fault Contract je definován pomocí atributu *FaultContract*, tento atribut je aplikován na metodu, která může výjimku vyvolat. Je možno předávat klasické .Net vyjímky, nebo vlastní typy, které je ale nutné zahrnout do Data Contractu, aby byla možná jejich serializace. Fault Contract je mapován na Simple Object Access Protocol (SOAP) Faults.

```
[DataContract]
public class ProdujtFault
{
    [DataMember]
    public string NazevProduktu;
    [DataMember]
    public string Problem;
}
```

Příklad 6: Ukázka objektu pro Fault Contract

```
[ServiceContract]
public interface IProduktySluzba
{
    [OperationContract]
    List<string> SeznamProduku();
    [OperationContract]
    [FaultContract(typeof(ProdujtFault))]
    Produkt GetProdukt(string cisloProduktu);
    [OperationContract]
    [FaultContract(typeof(ProdujtFault))]
    int PocetNaSklade(string cisloProduktu);
    [OperationContract]
    [FaultContract(typeof(InvalidOperationException))]
    bool ObjednejProdukt(ObjednavkaProduktu
objednavkaProduktu);
}
```

Příklad 7: Ukázka Fault Contract

2.3.5 Metadata

Metadata jsou v podstatě standardizovaný popis jak spolupracovat se službou. Obsahují popis operací které může služba vykonávat, strukturu datových typů se kterými tyto operace pracují a které vracejí a definici všech endpointů implementovaných službou společně s jejich možnostmi a požadavky.

Metadata jsou publikována v XML formátu, což umožňuje vývojářům používajícím jiné technologie něž WCF vytvářet na jejich základě klienty pro komunikaci s WCF službami.

WCF poskytuje bohatou infrastrukturu pro exportování, zveřejňování, získávání a importování metadat. To znamená že vývojář nemusí sám metadata vytvářet, ale stačí pouze povolit jejich generování a o zbytek se postará WCF. Z bezpečnostních důvodů je generování metadat v základním nastavení vypnuto. Pokud chceme metadata zveřejňovat je třeba explicitně nakonfigurovat službu. Existují dva způsoby jak toho docílit.

2.3.5.1 HTTP-GET

Jeden ze způsobů získání metadat služby je pomocí http-get dotazu. WCF umožňuje využití tohoto dotaz automaticky, jediné co je

třeba je povolit ho v nastavení pomocí *httphetEnabled="true"*. Služba poté publikuje svá metadata na adrese v tomto tvaru *baseAddress+?wsdl* (pokud by base Address byla http://localhost:8080/ tak adresa pro přístup k metadatům bude http://localhost:8080/?wsdl).

2.3.5.2 Metadata Exchange Endpoint

Druhý způsob pro publikování metadat je použití speciálního endpointu nazývaného Metadata Exchange Endpoint (MEX endpoint). Jako každý endpoint i MEX endpoint se skládá z ABC.

Jako adresu MEX endpointu je možno zvolit libovolnou adresu, nic méně je třeba dodržet pravidla tvorby adresu pro zvolený Binding. Standardní způsob tvorby adresy je za pomoci přípony *mex*, například *http://localhost:8080/Sluzba/mex*.

Binding je možno vybrat ze čtyř které WCF obsahuje. Jedná se o *mexHttpBinding*, *mexHttpsBinding*, *mexNamedPipeBinding* a *mexTcpBinding*.

Jako Contract MEX endpoint využívá rozhraní *IMetadataExchange.* Toto rozhraní a jeho implementace je opět poskytována WCF automaticky.

```
<system.serviceModel>
    <services>
      <service name = "Sluzba" >
        <endpoint
address="http://localhost:8080/Sluzba/mex"
        binding="mexHttpBinding"
        contract="IMetadataExchange" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name = "metadata">
          <serviceMetadata httpGetEnabled = "true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
```

Příklad 8: MEX endpoint a HTTP-GET

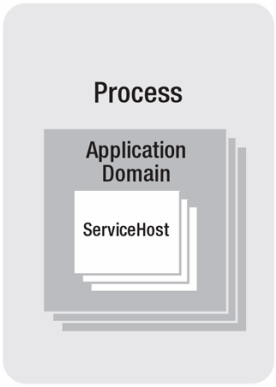
2.4 Hostování služeb

Aby se služba stala aktivní, musí být hostována v prostředí, které ji vytváří a ovládá její životní cyklus. WCF služby jsou navrženy aby bylo možno jejich hostování v jakémkoliv Windows procesu který podporuje spravovaný kód (managed code). Proces ve kterém je služba hostována se nazývá hostující proces (host process). Jeden hostující proces může hostovat více služeb a jedna a ta samá služba může být hostována ve více hostujících procesech. Existují tři základní způsoby jak

získat hostující proces: Pomocí Internet Information Server (ISS), Windows Activation Service (WAS), nebo jako self hosting.

2.4.1 Host architektura

Pro běh spravovaného .Net kódu jsou vytvořeny assembly. Assembly nejsou hostovány přímo ve Windows procesu. Namísto toho je tento spravovaný kód pomocí CLR isolován, vytvořením oddělených logických částí v procesu, nazývaných aplikační doména (application domain). Jeden proces může obsahovat více aplikačních domén a každá aplikační doména může hostovat kus kódu zapouzdřeného v assembly. CLR automaticky vytváří jednu základní aplikační doménu při vytvoření procesu. Tato základní aplikační existuje po celou dobu existence procesu. Ve většině případů základní aplikační doména neobsahuje žádný kód. Místo toho proces vytvoří novou aplikační doménu, která může být ukončena nezávisle na procesu. Ve velkém množství aplikací je žádoucí, aby serverová i klientská část byla umístěna ve vlastní aplikační doméně. Vztah mezi aplikací a aplikační doménou je velmi podobný vztahu mezi aplikační doménou a instancí třídy *ServiceHost*. Třída *ServiceHost* implementuje hosta používaného WCF k nakonfigurování, vystavení a kontrole životního cyklu služby. Jeden proces má minimálně jednu aplikační doménu a každá aplikační doména má nula, nebo více instancí třídy *ServiceHost* (viz Obrázek 5).



Obrázek 5: Proces (převzato z [6])

2.4.2 Způsoby hostování

Jak již bylo zmíněno dříve existují tři základní způsoby hostování WCF služby:

- **Microsoft Internet Information Server (ISS):** Hostování v ISS přináší řadu výhod. ISS samo vytvoří hostující proces poté co služba obdrží zprávu (pokud proces již neexistuje), který se stará o ovládání služby a její životní cyklus. ISS monitoruje stav procesu služby a dokáže jednat na základě tohoto stavu (například pokud služba dlouho neodpovídá ISS automaticky proces recykluje). Nevýhodou ISS je, že podporuje pouze HTTP protokol.

- **Windows Activation Service (WAS):** Jedná se o nový mechanizmus aktivace procesů dostupný na systémech Windows Server 2008 a Windows Vista. Vychází z modelu ISS a proto přebírá výhody tohoto modelu jako například sdružování aplikací, monitoring, nebo recyklaci. Na rozdíl od ISS není omezen pouze na http protokol.

- **Self hosting:** Jedná se o nejjednodušší způsob hostování služby. Vývojář musí sám vytvořit prostředí ve kterém bude služba existovat. Jako hostovací prostředí je možno použít jakýkoliv Windows proces: například WinForm aplikaci, WPF aplikaci nebo konzolovou aplikaci. Tato aplikace je poté odpovědná za ovládání a životní cyklus služby. Výhodami self hostingu jsou jednoduchost použití, flexibilita, snadný deployment, podpora všech Binding a transportních protokolů. Nevýhodou je omezená dostupnost služby. Self hosting také neposkytuje implicitně vlastnosti jako je vysoká dostupnost, jednoduchá zpráva, robustnost, verzování a podobně. Pokud je nějaké této vlastnosti zapotřebí musí ji vývojář implementovat sám.

2.5 Klienti WCF

WCF klient je aplikace která využívá funkcionality služby. Komunikace klienta a služby probíhá pomocí endpointu služby. Aby tohoto bylo dosaženo klient musí znát několik základních informací o službě jako adresu kterou endpoint používá ke komunikaci, Binding a Contract služby.

2.5.1 Architektura klienta

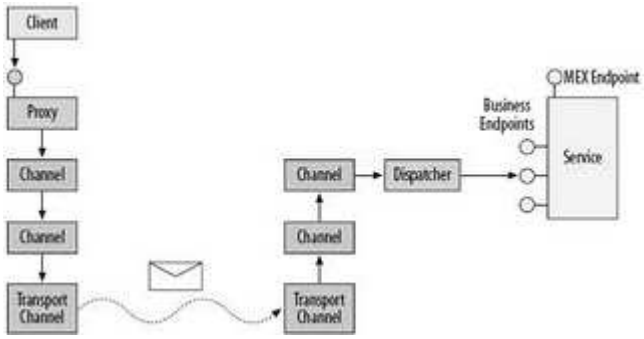
WCF poskytuje bohaté API pro komunikaci klientských aplikací se službou. Toto API, implementované *System.ServiceModel* assembly, se stará o serializaci typů do XML a odesílání zpráv z klienta na službu. Je možno programovat přímo za použití tohoto API, nebo využít nástroj *svcutil.exe* k vygenerování proxy třídy a konfiguračního souboru. Existují situace vhodné pro použití prvního způsobu a situace vhodné pro požití toho druhého.

Proxy je v podstatě vygenerovaný Service Contract služby, který umožňuje převod metod volaných klientem na odchozí zprávy a převod příchozích zpráv na informace, které může klient použít ve formě návratových hodnot a výstupních parametrů.

Další důležitou součástí klienta je implementace rozhraní *IClientChanel*. Toto rozhraní definuje opera umožňující vývojářům kontrolovat funkcionalitu kanálů, jako je zavírání klientských relací a uvolnění kanálu. Poslední části je klientský kanál postavený na Binding specifikovaném v konfiguračním souboru.

2.5.2 Komunikace mezi klientem a službou

Komunikaci mezi službou a klientem velmi dobře zachycuje Obrázek 6. Komunikace začíná na straně klienta, kdy proxy serializuje volání na zprávu. Ta poté projde řetězem kanálů. Kanál je ve své podstatě pouhý zachytávač, jehož účelem je provést specifický úkon. Každý kanál na klientské straně provede předzpracování zprávy. Přesná struktura a složení řetězu kanálů záleží hlavně na zvoleném Binding. Například jeden kanál může být zodpovědný za kódování zprávy, jiný za zabezpečení, jiný za zpracování transakcí a podobně. Poslední kanál na klientské straně je transportní kanál, který pošle zprávu přes nakonfigurovaný transportní protokol na stranu hosta.



Obrázek 6: Komunikace klienta a služby (převzato z [2], kap 1.11)

Na straně hosta zpráva opět prochází řetězem kanálů. Tyto kanály také provádějí předzpracování zprávy. První kanál na straně hosta je transportní kanál, který přejímá zprávu od transportního protokolu. Kanály následující po něm provádějí různé úkony jako například rozšifrování těla zprávy, dekódování zprávy, pospojování transakcí a podobně. Od posledního kanálu klientské strany je zpráva předána na Dispatcher. Dispatcher zajišťuje převod zprávy na volání metod instance služby.

Služba nemá žádný způsob jak zjistit, že nebyla volána lokálním klientem. Ve skutečnosti je vždy volána lokálně dispatcherem. To znamená, že nezáleží zda se klient nachází ve stejném procesu, stejném počítači, či na druhé straně světa, služba se bude vždy chovat stejně.

Poté co instance služby provede volání, předá kontrolu zpět dispatcheru. Ten převede navrácené hodnoty a chyby (jsou-li nějaké) na návratovou zprávu. Proces se nyní obrátí. Dispatcher pošle zprávu

řetězem kanálů, ty zprávu zakódují, zašifrují a podobně. Nakonec je zpráva přes transportní kanál odeslána na stranu klienta kde opět prochází řetězem kanálů. Zpráva je dekódována, rozšifrována, transakce jsou schváleny, nebo odmítnuty a podobně. Poslední kanál předá zprávu proxy. Proxy převede zprávu například na navrácené hodnoty, nebo výjimky a předá kontrolu klientovi.

3 Výukové materiály WCF

3.1 Současný stav problematiky

O technologii WCF bylo sepsáno již nemalé množství článků, několik publikací a online návodů. Oficiální stánky této technologie také obsahují řadu užitečných informací. Když jsem se začínal zajímat o tuto technologii přečetl jsem velké množství těchto článků a byl jsem velmi zklamán když jsem zjistil, že skoro všechny obsahují velmi podobné informace. Většina článků obsahovala informace velmi povrchní, v podstatě se jednalo o nefunkční kusy zdrojového kódu ze kterého nebylo mnohdy jasné proč a jakým způsobem tento kód vznikl. Po nějakém čase jsem se dostal k ucelenějším publikacím, které obsahovaly mnohem více informací. Tyto informace byly členěny do logických celků a výuka z nich byla mnohem snazší. Jedinou nevýhodou těchto publikací byly, že byly napsány v anglickém jazyce. V českém jazyce, co jsem si vědom, nebyla vydána žádná z těchto publikací. Co se týče internetových článků těch v českém jazyce také není moc a kromě jednoho velmi kvalitního seriálu na serveru *netstudent.cz.* Jejich kvalita také není o nic lepší, než u jejich anglicky psaných protějšků.

3.2 Použitá metodika

Před tím něž jsem mohl přistoupit k samotné tvorbě příručky, musel jsem se dopodrobna seznámit s problematikou WCF. Přečetl jsem tedy velké množství publikací a článků o této technologii. Po nastudování potřebných znalostí jsem za účelem osvojení těchto znalostí začal vytvářet jednoduché aplikace. V těchto aplikacích jsem experimentoval s technologií WCF.

Také jsem se musel rozhodnout, jaký způsobem budu při tvorbě příručky postupovat a jak seznámím čtenáře s WCF. Po dlouhém přemýšlení jsem nakonec zvolil postup výuky, který nejvíce vyhovuje mě samotnému.

Příručka je rozdělena do jednotlivých lekcí a pro každou lekci jsem vytvořil ukázkový příklad. Začátek lekce je vždy uvozen teorií. Po teorii následuje popis tvorby ukázkového přikladu. Právě tento popis, je společně s vlastním ukázkovým příkladem nejdůležitější částí lekce. Díky němu čtenář hned vidí jak používat teorii ze začátku lekce v praxi. Z popisu tvorby příkladu čtenář zjistí jak příklad krok za krokem vznikal. Nejedná se tedy o pouhý komentář ke zdrojovému kódu příkladu, ale o skutečný postup jeho tvorby.

Pří tvorbě ukázkových příkladů jsem kladl velký důraz na funkčnost příkladu. Všechny jsou proto plně funkční a čtenář si je může sám otestovat. Popisu jak jednotlivé příklady otestovat je většinou věnovaná závěrečná část jednotlivých kapitol.

Příručku jsem rozdělil do čtyř kapitol. Jelikož je příručka určena pro čtenáře nemající žádné zkušenosti s WCF. Při výběru obsahu těchto kapitol jsem vybíral témata skutečně určená začátečníkům, která jim umožní rychlé a snadné proniknutí do základů WCF. Jednotlivé kapitoly tedy osahují tyto oblasti WCF : tvorba služeb, hostování služeb, tvorba klientů a obsluha výjimek.

3.3 Úvodní kapitola

Úvodní kapitola seznamuje čtenáře se stručním obsahem příručky a rozvržením jednotlivých kapitol. Čtenář je dále seznámen s vývojovými nástroji, které používám.

Pro používání technologie WCF je nutný .Net Framework ve verzi 3.0 nebo vyšší. Jelikož čtenář nemusí mít tento framework nainstalován součástí úvodní kapitoly je jednoduchý popis jak si jej obstarat a nainstalovat. Já v příručce používám .Net Framework 3.5 SP1 a proto čtenáři doporučuji nainstalování právě této verze.

Jako operační systém používám Windows XP. Naštěstí verze operačního systému nemá na funkčnost WCF velký vliv a proto na čtenáře nekladu v tomto ohledu žádné nároky.

Pří tvorbě ukázkových příkladů používám vývojové prostředí Microsoft Visual Studio 2008. Použití tohoto vývojového prostředí vyžaduji i po čtenářích. Pro čtenáře kteří Visual Studio nemají proto přikládám způsob jeho obstarání.

3.4 Vytvoření WCF služby

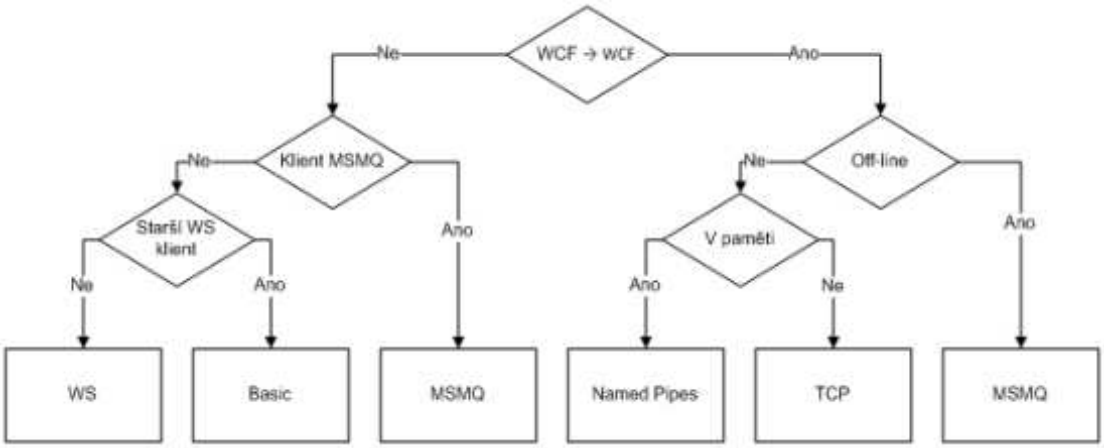
Základem WCF jsou služby. První kapitola proto pojednává právě o nich. Mým cílem nebylo zahrnout čtenáře nepřeberným množstvím informací o službách, ale pouze ho seznámit se základními aspekty návrhu a vytváření služeb. Hlavním tématem první lekce je tedy endpoint a jeho jednotlivé části (Address, Binding, Contract). Na konci kapitoly ještě ukazuji způsob jak je možno službu otestovat bez přítomnosti hostovacího prostředí a klienta. Ukázkový příklad k první lekci je služba objednávka knih. Tento příklad je dále rozvíjen v dalších lekcích.

3.4.1 Endpoint

Endpoint je nejdůležitější část WCF služby. Je popisován pomocí tří částí ze kterých se skládá jako tzv. ABC (Address, Binding, Contract). Je tomu hlavně díky jednoduchosti a snadné zapamatovatelnosti. Je důležité, aby se čtenář s těmito částmi seznámil. Proto po popisu endpointu následuje popis ABC.

3.4.2 Výběr Binding

Výběr ze systémových Binding může být pro někoho kdo s WCF teprve začíná velmi matoucí. Výběr se řídí tím co od komunikace očekáváme. Pro snazší orientaci v systémových Binding jsem vytvořil tabulku (viz Tabulka 1) shrnující vlastnosti jednotlivých Binding. Uvědomuji si že ani tato tabulka nemusí být pro nováčka dostatečně srozumitelná. Naštěstí existuje způsob jak výběr více zjednodušit, tento způsob je popsán na Obrázku 7.



Obrázek 7: Výběr ze System Binding

3.5 Hostování WCF služby

Druhá kapitola navazuje na první a je v ní vysvětleno jak hostovat WCF službu. Možností hostování není málo, ale já se dopodrobna rozebírám pouze self hosting. Ukázkový příklad z první lekce je v této lekci rozšířen o dvě hostovací aplikace. První je WinForm aplikace na které je ukázáno hostování služby ve Windows procesu, druhá je aplikace pro hostování služby ve Windows Service.

3.5.1 Výběr hostovacího prostředí

Pro hostování pomocí self hostingu jsem se rozhodl z několika důvodů. Hlavním důvodem byla složitost instalace a nastavení ISS. Jsem přesvědčen, že čtenáře kteří s ISS nemají žádné zkušenosti by to mohlo odradit a nakonec by si tento způsob hostování ani nevyzkoušeli. Dalším důvodem bylo zaměření ISS více na podnikové služby. Domnívám se že čtenáři pro pochopení hostování více ocení flexibilitu a jednoduchost nasazení, kterou přináší self hosting.

3.5.1.1 Self hosting

Popisuji vytvoření kompletní aplikace s grafickým uživatelským rozhraním, která má za úkol spuštění a zastavení hostování služby.

Hlavní důraz je kladen na kód zabývající se samotným hostováním. Hostování služby probíhá ve třech jednoduchých krocích (viz Příklad 9):

- Vytvoření instance třídy *ServiceHost*, té se jako parametr předává typ hostované služby
- Zavolání metody *Open* pro počátek hostováni
- Zavolání metody *Close* pro ukončení hostování

```
try
{
    using (knizniOjednavkaSluzbaHost = new
        ServiceHost(typeof(KnizniObjednavkaSluzba)))
    {
        knizniOjednavkaSluzbaHost.Open();
        labelStav.Text = "Služba běží.";
        buttonStart.Enabled = false;
        buttonStop.Enabled = true;
    }
}
catch  {
    knizniOjednavkaSluzbaHost.Close();
}
```

Příklad 9: Self hosting, počátek hostování

3.5.1.1 Self hosting pomocí Windows Service

Hostování služby pomocí Windows Service je velmi zajímavá myšlenka. Jedná se o takovou chudší alternativu oproti hostování pomocí

ISS. Takto hostovanou službu je možno řídit jako každou jinou službu v systému. Je možno ji spustit manuálně nebo automaticky, pozastavit, zastavit či restartovat. Události odehrávající se na službě jsou zaznamenávány v log souboru do kterého je možno kdykoli nahlédnout.

Součástí tohoto příkladu je i vytvoření instalátoru pro instalaci služby do systému.

3.5.2 Reference na službu

Služba vytvořená v první lekci je v podstatě obyčejná dynamická knihovna. To je jeden ze způsobů jak službu vytvořit. Druhým způsobem je mít kód služby přímo jako součást hostovací aplikace. Oba tyto způsoby jsou hojně používané, mají své výhody a nevýhody. První způsob poskytuje přenositelnost, což znamená že služba může být hostována více hosty bez nutnosti duplikovat kód a jelikož je kód služby na jednom místě, je jednoduší provádět ve službě změny. Toto jsou také důvody proč jsem umístil službu do knihovny. V neposlední řadě mi to pomohlo ucelit první lekci, která by jinak přesahovala do lekce druhé

3.6 Vytvoření WCF klienta

Třetí kapitola pojednává o vytváření a nastavování WCF klientů. Moje služba objednávky knih je zde doplněna o klientskou aplikaci, takže čtenář má konečně pohromadě službu i klienta, který tuto službu používá. Ukázkovým příkladem je klient vytvořený jako WinForm aplikace. Tento typ aplikace jsem vybral z důvod, že se jedná o starší technologii takže s ní bude více čtenářů obeznámeno.

3.6.1 Vytvoření klientské aplikace

Vytvoření klientské aplikace probíhá v jednotlivých krocích. Jako první vytvářím grafické uživatelské rozhraní. Tato část tvorby není s ohledem na WCF příliš důležitá a proto nezabíhám do přílišných podrobností. Více pozornosti věnuji vytváření WCF a používání klienta pro komunikaci se službou.

3.6.2 Postup při tvorbě WCF klienta

Klientská aplikace je spravovaná aplikace používající WCF klienta ke komunikaci se službou. Tvorba WCF klienta probíhá ve čtyřech hlavních krocích:

- Získání Service Contract, Binding a Address z endpointu služby
- Vytvoření WCF klienta pomocí těchto informací
- Volání operací
- Uzavření objektu WCF klienta

Získání metadat služby provádím pomocí Visual Studia a volby *Add Service Reference*. Tento způsob je velmi rychlý a jednoduchý. Nejedná se ale o standardní způsob získání metadat a vyžaduje aby programátor používal Visual Studio. Standardní způsob je pomocí utility *svcutil.exe* a proto popisuji i jak získat metadata tímto způsobem.

Příklad 10 ukazuje nejběžněji používaný postup při vytvoření klienta, za využití třídy klienta vygenerované pomocí přidání reference na službu, nebo *svcutil.exe*.

```
using (KniznıObjednavkaClient proxy = new
KniznıObjednavkaClient("NetTcpBinding_IKniznıObjednavka"))
{
   proxy.Open();
   kniha = proxy.GetKnihaPodleIsbn(textBoxVyhledat.Text);

}
```

Příklad 10: Vytvoření klienta pomocí vygenerované proxy

Existuje ovšem i druhý způsob pro vytvoření klienta a to sice pomocí třídy *ChannelFactory* (viz Příklad 11). Tuto třídu je možno

využít k vytvoření klienta bez nutnosti generování třídy WCF klienta přidáním reference na službu, nebo použitím svcutil.exe. Tento způsob je možno využít pouze v případě, že má programátor přístup k rozhraní služby.

```
EndpointAddress address = new
EndpointAddress("net.tcp://localhost:9000/KnizniObjednavka"
);
NetTcpBinding binding = new NetTcpBinding();
ChannelFactory<SluzbaWCF.IKnizniObjednavka> factory = new
ChannelFactory<SluzbaWCF.IKnizniObjednavka>(binding,
address);
SluzbaWCF.IKnizniObjednavka proxy =
factory.CreateChannel();
using (proxy as IDisposable)
{
    kniha = proxy.GetKnihaPodleIsbn(textBoxVyhledat.Text);
}
```

Příklad 11: Vytvoření klienta pomocí *ChannelFactory*

Z posledních dvou příkladů je jasně vidět, že rozdíl mezi těmito způsoby je minimální. Samotné vytvoření instance klienta je záležitost jednoho řádku kódu. Po zavolání operací klientem je nutné korektní uzavření klienta. Toho lze jednoduše docílit použitím bloku *using*. Použití tohoto bloku s sebou ale nese jeden problém. Na konci tohoto bloku dochází automaticky k uzavření klienta a to samo o sobě může

vyvolat výjimku, i v případě že nic uvnitř *using* bloku žádnou výjimku negeneruje. Tato výjimka poté zamezí provedení kódu následujícího za *using* blokem. Tomuto problému se více věnuji v následující kapitole o výjimkách.

3.7 Práce s výjimkami a chybami

Závěrečná čtvrtá kapitola se zabývá vyvoláváním a obsluhou výjimek. Používání výjimek je velmi důležitou součástí práce programátora a proto jsem mu věnoval celou jednu kapitolu. WCF pro prácí s chybami využívá SOAP Fault zprávy. Kapitola se tedy z velké části zabývá tímto tématem. Teoretická část kapitoly se skládá z popisu typu Fault a způsobu jeho definice. Dále popisuji jak vytvářet a posílat deklarované i nedeklarované chyby. A je uzavřena popisem způsobů odchytávání a zpracovávání chyb.

3.7.1 Ukázkový příklad práce s výjimkami a chybami

I takto kapitola obsahuje vytvoření ukázkového příkladu. Pro potřeby kapitoly jsem opustil službu objednávka knih z minulých kapitol a použil novou službu a klienta. Vytvořil jsem službu *ObjednávkaChyba*. Dále jsem vytvořil klientskou WinForms aplikaci, pro komunikaci s touto službou.

3.7.2 Služba ObjednávkaChyba

Při tvorbě služby jsem postupoval jiným způsobem než v první kapitole. Služba *ObjednávkaChyba* není oddělena v samostatné knihovně, ale je součástí hostovací aplikace, tím čtenáře seznamuji i s tímto způsobem tvorby služby. Service Contract služby definuje pouze jednu metodu, tato metodata deklaruje Fault Contract typu *ObjednavkaFault* (viz Příklad 12), takže je čtenáři ukázáno použití tohoto typu kontraktu.

```
[OperationContract]
[FaultContract(typeof(ObjednavkaFault))]
string OdeslatObjednavku(Objednavka objednavka);
```

Příklad 12: Deklarace Fault Contract *ObjednavkaFault*

Samotné vytváření výjimek je poté umístěno v implementaci této metody. Čtenáře je seznámen se třemi typy chyb. Těmito typy jsou:

- deklarované chyby vytvořené pomocí generické třídy F*aultException<ObjednavkaFault>*
- nedeklarované chyby vytvořené pomocí negenerické třídy *FaultException*
- klasické .Net výjimky

3.7.2 Klient služby ObjednávkaChyba

Klientská aplikace je jednoduchý WinForms formulář ve kterém je možno zadat objednávku a odeslat ji na službu. Služba ji vyhodnotí a klient zobrazí jak objednávka dopadla. Pokud bylo s objednávkou něco v nepořádku služba vytvoří chybu, ta je na klientu zachycena jako výjimka a zpracována (viz Příklad 13).

```
try
{
    ServiceReference.VyjimkySluzbaClient proxy =
 new VyjimkyKlient.ServiceReference.VyjimkySluzbaClient();
    proxy.Open();
    string ret = proxy.OdeslatObjednavku(objednavka);
}
catch (FaultException<ServiceReference.ObjednavkaFault>
chyba)
{
    MessageBox.Show(chyba.Detail.Problem);
}
catch (FaultException neznamaChyba)
{ MessageBox.Show(neznamaChyba.Message); }
catch (CommunicationException)
{ MessageBox.Show("Došlo k chybě v komunikaci."); }
catch (Exception ex) { MessageBox.Show(ex.Message); }
```

Příklad 13: Obsluha výjimek

K zachytávání výjimek dochází v try/catch bloku a jak vidíme klient může obsluhovat libovolné množství výjimek.

3.7.2.1 Správné uzavření klienta

Nakonec této kapitoly jsem zařadil popis a řešení problému, který vzniká při použití bloku *using* k uzavření WCF klienta. Bloku *using* na svém konci volá metodu *Dispose*. Tato metoda je shodná s metodou *Close* na WCF klientu a může vyvolat výjimku pokud během uzavírání dojde k síťové chybě. Proto se doporučuje klienta otevřít, používat a zavřít v jedno bloku try a v blocích catch poté obsluhovat chyby, nesmíme ale zapomenout zde také klienta uzavřít pomocí metody *Abort* (viz Příklad 14).

```
try
{
    client.Close();
}
catch (CommunicationException e)
{
    client.Abort();
}
catch (TimeoutException e)
{
    client.Abort();
}
```

```
catch (Exception e)
{
    client.Abort();
    throw;
}
```

Příklad 14: Korektní uzavření klienta

Tento způsob uzavření klienta je také použit při tvorbě aplikace klienta v této lekci.

4 Závěr

Cílem mé bakalářské práce bylo prozkoumání technologie WCF jako nového způsobu tvorby distribuovaných aplikací. A následné sestavení výukových materiálů pro rychlé proniknutí do základů této technologie. V oblasti českých zdrojů pro tuto technologii je situace velmi špatná, je jich velmi málo. To mě navedlo na myšlenku vytvoření této práce.

Vytvořil jsem příručku technologie WCF v českém jazyce, určenou pro zájemce, kteří nemají s WCF žádné zkušenosti. Příručka je členěna do kapitol. Každá kapitola obsahuje ukázkový příklad, který jsem pro ni také vytvořil. Při vykládání látky v jednotlivých kapitolách postupuji způsobem, který by nejvíce vyhovoval mě samotnému. Problematika tedy není vysvětlována pouhou teorií, ale z hlavní části pomoci ukázkových příkladů a zdrojových kódů.

Vytvoření jednotlivých ukázkových příkladů a popis jejich tvorby v jednotlivých kapitolách považuji za nejdůležitější část práce. Zvolil jsem postup krok za krokem, abych mohl čtenáře seznámit jakým způsobem postupovat a ne mu pouze předvést finální zdrojový kód. Všechny ukázkové příklady jsou plně funkční aplikace.

Témata jednotlivých kapitol jsem vybíral s ohledem na to, že příručka je určena začátečníkům. Vybral jsem tedy pouze základní

vlastnosti WCF. Tím jsem dosáhl snadného a rychlého proniknutí do základů této technologie.

Technologii WCF je možno označit za velmi zdařilého nástupce předchozích technologií pro tvorbu distribuovaných, klient/server aplikací. Vytváření těchto aplikací je s její pomocí velmi rychlé a efektivní. Programátor je odstíněn od velkého množství problémů, které s sebou vývoj aplikací tohoto typu nese. Může se tak plně soustředit na implementaci business logiky a jeho práce je proto usnadněna a urychlena.

Dle mého názoru byl cíl mé práce naplněn. Podařilo se mi ukázat využití WCF pro tvorbu klient/server aplikací. A poskytnout zájemců rychlý způsob proniknutí do této technologie.

Přehled použité literatury

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

[2] LOWY, Juval. *Programming WCF Services*. Sebastopol : O'Reilly, 2007. 640 s. ISBN 978-0596526993.

[3] KLEIN, Scott. *Professional WCF Programming : .NET Development with the Windows Communication Foundation*. Indianapolis, Indiana : Wiley Publishing, Inc., 2007. 430 s. ISBN 978-0470089842.

[4] LARYŠ , Kryštof . *WCF pro začátečníky : 2. díl: contract, binding, service behavior* [online]. 2009 [cit. 2010-04-22]. WCF pro začátečníky. Dostupné z WWW: <http://www.netstudent.cz/%C4%8Cl%C3%A1nky/tabid/56/articleType/ArticleView/ArticleID/226/PageID/209/Default.aspx>.

[5] KOŠŤÁL, Marián. *Vyvojar.cz* [online]. 2007 [cit. 2010-04-22]. WCF - Bindings. Dostupné z WWW: <http://www.vyvojar.cz/Articles/459-wcf-bindings.aspx>.

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

[7] RIGSBY, Dan. *Dan Rigsby* [online]. 2008 [cit. 2010-04-22]. WCF Metadata. Dostupné z WWW: <http://www.danrigsby.com/blog/index.php/2008/05/27/wcf-metadata>.

[8] SHARP, John. *Microsoft Windows Communication Foundation Step by Step*. Redmond, Washington : Microsoft Press, 2007. 448 s. 978-0735623361.

[9] MCMURTRY, Craig; MERCURI, Marc; WATLING, Nigel. *Microsoft Windows Communication Foundation : Hands-on*. [s.l.] : Sams Publishing, 2006. 560 s. ISBN 978-0-672-32877-0.

[10] Microsoft. *Basic WCF Programming* [online]. 2010 [cit. 2010-04-22]. Dostupné z WWW: <http://msdn.microsoft.com/enus/library/ms731067.aspx>.

[11] Microsoft Corporation. *Windows Communication Foundation : MSDN*

[online]. Microsoft Corporation, c2007 [cit. 2010-04-22]. Dostupný z WWW:
<http://msdn.microsoft.com/en-us/library/ms735119.aspx>.

[12] Microsoft. *Building Clients* [online]. 2010 [cit. 2010-04-22]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms730825.aspx>.

[13] Microsoft. *Endpoints: Addresses, Bindings, and Contracts* [online]. 2010 [cit. 2010-04-22]. Dostupné z WWW:
<http://msdn.microsoft.com/en-us/library/ms733107.aspx>.

[14] Microsoft. *Metadata* [online]. 2010 [cit. 2010-04-22]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms731823.aspx>.

[15] Microsoft. *Specifying and Handling Faults in Contracts and Services* [online]. 2010 [cit. 2010-04-22]. Dostupné z WWW:
<http://msdn.microsoft.com/en-us/library/ms733721.aspx>.

[16] *ExtremeExperts* [online]. 2007 [cit. 2010-04-22]. Exception Handling in Windows Communication Framework and Best Practices . Dostupné z WWW:
<http://www.extremeexperts.com/Net/Articles/ExceptionHandlingInWCF.aspx>.

[17] Microsoft. *Services* [online]. 2010 [cit. 2010-04-22]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/dd699762.aspx>.

[18] Microsoft. *Introduction to Windows Communication Foundation* [online]. 2010 [cit. 2010-04-22]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/dd936243.aspx>.

[19] GAURAV, . *Exploring Windows Communication Foundation* [online]. 2007 [cit. 2010-04-22]. Dostupné z WWW: <http://www.codeproject.com/KB/WCF/edujini_wcf_scart_01.aspx>.

[20] Microsoft. *Clients* [online]. 2010 [cit. 2010-04-22]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms734653.aspx>.

[21] ALBERTO, Al. *Communication options with WCF - Part 1* [online]. 2006 [cit. 2010-04-22]. Dostupné z WWW: <http://www.codeproject.com/KB/WCF/WCF_CommOptions_part1.aspx>.

Přílohy

A Obsah přiloženého CD

- Příručka WCF rozdělená do kapitol
- Projekty se zdrojovými kódy ukázkových příkladů