Konfigurátor počítačových sestav a spotřební elektroniky

Configurator of PC and consumer electronics

Bc. Radek Zaťko

Diplomová práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Radek ZAŤKO**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Konfigurátor počítačových sestav a spotřební elektroniky.**

Zásady pro vypracování:

1. **Proveďte průzkum informačních zdrojů a literární rešerši z oblasti konfigurátorů PC sestav a spotřební elektroniky.**
2. **Analyzujte současný stav a porovnejte různá řešení.**
3. **Navrhněte možnosti implementace a vhodné řešení pro distribuovaný eshop (forma projektu).**
4. **Zvolená řešení prakticky realizujte včetně uživatelského manuálu a diskutujte jejich pozitiva a negativa.**
5. **Stanovte závěry k dané problematice včetně dalších možných doporučení.**

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **MACDONALD, Matthew; SZPUSZTA, Mario. ASP.NET 3.5 a C-SHARP 2008 – tvorba dynamických stránek PROFESIONÁLNĚ. Brno: Zoner Press, 2008. 1584 s. ISBN 978-80-7413-008-3.**
2. **EVJEN, Bill; HANSELMAN Scott; RADER, Devon. ASP.NET 3.5 v jazycích C-SHARP a Visual Basic – Programujeme profesionálně. Brno: Computer Press, 2009. 1600 s. ISBN 978-80-251-2069-9.**
3. **WALTERS, Robert E.; COLES, Michael; RAE, Robert; FERRACCHIATI, Fabio; FARMER Donald. Mistrovství v Microsoft SQL Server 2008. Brno: Computer Press, 2009. 864 s. ISBN: 978-80-251-2329-4.**
4. **PIALORSI, Paolo; RUSSO, Marco. Microsoft LINQ – Kompletní průvodce programátora. Brno: Computer Press, 2009. 616 s. ISBN: 978-80-251-2735-3.**
5. **CONOLLY, Thomas; BEGG, Carolyn; HOLOWCZAK, Richard. Mistrovství – Databáze – Profesionální průvodce tvorbou efektivních databází. Brno: Computer Press, 2009. 584 s. ISBN: 978-80-251-2328-7.**
6. **ERL, Thomas. SOA Servisně orientovaná architektura – Kompletní průvodce. Brno: Computer Press, 2009. 672 s. ISBN: 978-80-251-1886-3.**
7. **BUDD, Andy; MOLL, Cameron; COLLISON Simon. CSS – filtry, hacky a pokročilé postupy. Brno: Zoner Press, 2007. 272 s. ISBN: 978-80-86815-54-1.**
8. **SCHMULLER, Joseph. Myslíme v jazyku UML. Knihovna programátora. Praha : Grada, 2001. 359 s. ISBN 80-247-0029-8**

Vedoucí diplomové práce: **Ing. Petr Šilhavý, Ph.D.**
Ústav počítačových a komunikačních systémů

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

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

Ve Zlíně dne 19. února 2010

prof. Ing. Vladimír Vašek, CSc.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

12. 03. 2010

ABSTRAKT

Diplomová práce se zabývá problematikou konfigurátoru počítačových sestav a spotřební elektroniky. V teoretické části je rozebrán současný stav v ČR. Dále jsou vysvětleny pojmy z oblasti e-komerce, servisně orientovaných služeb, technologie, která určuje současný trend v této oblasti. V praktické části je realizována webová aplikace konfigurátoru sestav dle požadavků zadání.

Klíčová slova: konfigurator, eshop, B2C, SOA, XML, webové služby, ASP.NET, Ajax, Entity Framework, LINQ pro entity,

ABSTRACT

This diploma thesis deals with configurator of PC and consumer electronics. The theoretical chapter describes the current situation in the Czech Republic. The next part explains the basic terms about e-commerce, service-oriented aplication... There is described the technology, that determines the current trend in this area. In the practical chapter there is the implementation of web application configurator groups according to the award.

Keywords: configurator, eshop, B2C, SOA, XML, web services, ASP.NET, Ajax

Rád bych poděkoval vedoucí mé diplomové práce Ing. Petru Šilhavému, Ph.d. za ochotu, cenné rady a věcné připomínky, které mě během práce poskytla.

Prohlašuji, že

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

Prohlašuji,

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

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD

Konfigurátory sestav počítačů nejsou zcela běžnou součástí internetových obchodů, z důvodu náročnosti implementace a přípravě dat. Zaměstnanec, který sestavy připravuje, musí umět zhodnotit možnosti obchodního i technického charakteru. Je nutné sledovat kompatibilitu a pohyb cen, a to klade na tvůrce těchto konfigurací vysoké nároky v oblasti technických, obchodních znalosti a zkušeností z výroby počítačů. Proto je nutné sledovat nejnovější trendy ve vývoji výpočetní techniky.

Konfigurátor je většinou součástí e-shopů výrobců s delší tradicí, majících vlastní značku a odborné technické zázemí. S možností nasadit konfigurátor na u obchodníka s principy B2C jeho význam roste. S klientem tak může obchodník navrhnout počítač na míru přímo za jeho účasti.

Ve své diplomové práci předkládám řešení na základě principů servisně orientované architektury, za využití webových služeb. V teoretické části jsou rozebrány jednotlivé vlastnosti SOA, webové služby, které jsou základním stavebním kamenem servisně orientovaných návrhů. Jsou zde popsány standardy specifikace WS-I Basic Profile: WSDL, SOAP a UDDI. Důležitou součástí je kapitola věnovaná podpoře SOA na platformě .NET, ve které byla aplikace vytvořena. Podrobněji jsem se věnoval ASP.NET AJAX Controll Toolkitu, který přináší nový způsob použití Ajaxu při tvorbě webových stránek v platformě .NET. Součástí diplomové práce je i uživatelská příručka pro aplikaci konfigurátoru.

I. TEORETICKÁ ČÁST

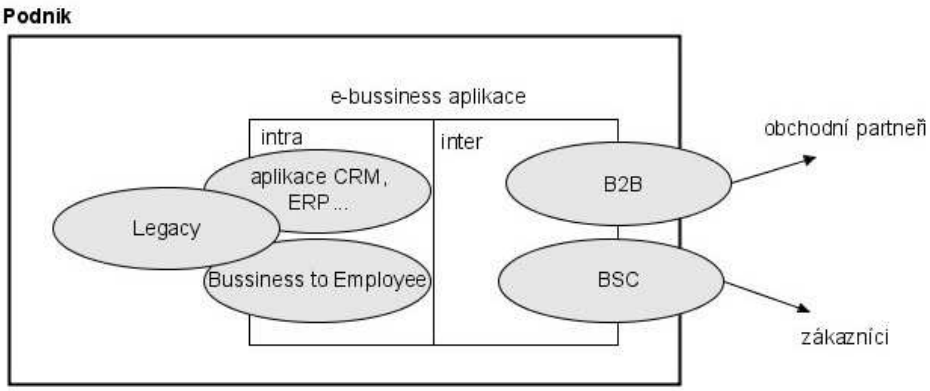
1 VYUŽITÍ KOFIGURÁTORŮ SESTAV V INTERNETOVÉM OBCHODĚ

V důsledku rozvoje obchodu po internetu vznikly a rychle se ujaly, dva nové termíny převzaté z angličtiny, kterými se na úvod seznámíme.

E-business - jakýkoli informační systém nebo aplikace, prostřednictvím které se uskutečňují obchodní transakce nebo jiné procesy související s obchodování a řízením podnikových činností. Tyto systémy jsou dnes převážně založeny na webových technologiích.

E-commerce - část e-business aplikací zaměřených na obchod B2B , B2C, C2C a C2B.

E-business je velmi široký pojem a můžeme ho chápat jako souhrn všech typů elektronické komunikace a obchodování, při kterých jsou využívány již zmíněné počítačové sítě. Je mnoho způsobů jak klasifikovat aktivity e-businessu. Nejprve se tedy podíváme na e-business obecnějším pohledem.



Obr. 1. Rozvrstvení e-bussiness aplikací

Při zkoumání oblastí použití e-business aplikací budeme brát v úvahu subjekty na obou koncích obchodní transakce a rozdělíme si je na dvě hlavní třídy:

Intra-business - aplikace typu Intra-business, zahrnuje všechny e-business systémy, které firma nebo organizace používá uvnitř sebe sama, a které nepřesahují hranice podniku. Mohou to být například intranetové webové stránky pro zaměstnance.

Inter-business - obsahuje všechny aplikace, které vyžadují jakýkoli druh interakce mezi společností (firmou, podnikem) a nějakou externí entitou (zákazníkem, obchodním

partnerem nebo finanční institucí). Jako příklad můžeme uvést aplikaci typu e-komerce, která modeluje nákupně-prodejní činnosti mezi firmou a spotřebitelem přes Internet. Je třeba ale dodat, že z infrastrukturního a implementačního úhlu pohledu, se tyto dvě třídy hodně prolínají a striktní hranice se jen těžko určuje.

Druhy e-komerce podle zúčastněných stran:

B2B (business-to-business) - jedná se o obchodní vztah, realizovaný automatizovanými procesy a robustními softwarovými balíky, provozovaný převážně v prodejních a distribučních sítích a to mezi výrobci, pobočkami, velkoobchody, distributory, dealery nebo obchodními zástupci. Základní rozdíl mezi B2B a B2C spočívá v tom, že prodávající (firma, distributor, dealer) nakupujícího předem zná. Obvykle jde o dlouhodobější vztah, který je ošetřen kupní smlouvou. Nejedná se tedy o klasické nakupování, ale o uzavírání propojení mezi společnostmi. Software používaný pro provoz B2B je mnohem rozsáhlejší než se používá v obchodování typu B2C.

B2C (business-to-consumer) - obchod se specializuje na prodej zboží a služeb konečnému spotřebiteli. Výrobci a distributoři nabízejí své výrobky z větší části prostřednictvím Internetu kdy k přímému kontaktu prodávajícího a nakupujícího nemusí nikdy dojít a také většinou nedochází.

C2C (consumer-to-consumer) - představuje vzájemný obchod mezi jednotlivými osobami, spotřebiteli. Jakýsi druh komerce ve formě směnného obchodu, burzy nebo blešího trhu.

C2B (consumer-to-business) je sice méně obvyklý druh e-komerce, při níž individuální spotřebitelé nabízejí převážně své služby (práci) firmám a společnostem, ale jistě má svůj význam například na trhu pracovní síly.

Po úvodu do problematiky e-komerce se již zaměříme na charakteristiku on-line obchodů s výpočetní technikou a na využití aplikace konfigurátoru v nich. Současné B2B eshopy jsou řešeny dvěma způsoby:

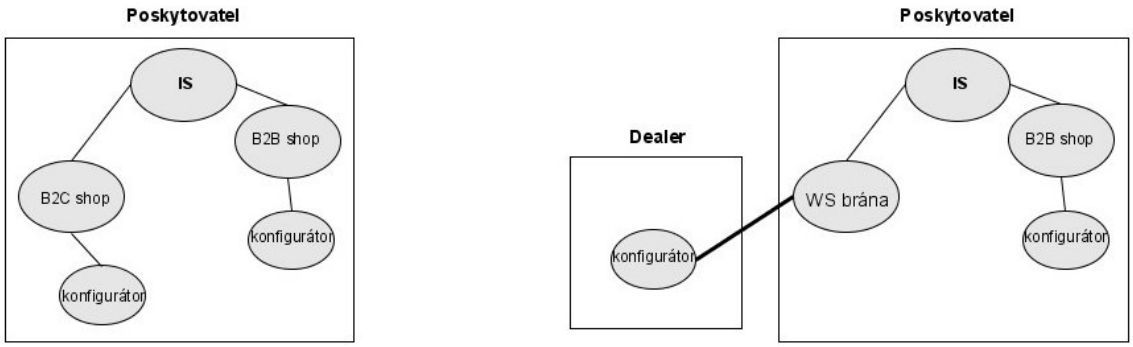
- modul lokálního informačního systému,
- externí produkt firmy zabývající se e-komercí napojený na lokální informační systém.

Větší distributoři využívají řešení B2C eshopů pro své dealery. Tyto eshopy jsou umístěny na serverech poskytovatele dat. Výhodou tohoto řešení je snadná implementace a údržba, nevýhodou je uniformita daných řešení, společná IP adresa pro skupinu B2C eshopů a minimální možnosti změny vzhledu. Výkonnost těchto aplikací klesá s rostoucím počtem eshopů a je zcela na poskytovateli kolik zdrojů dá k dispozici.

Za zmínku ještě stojí řešení pomocí exportu a importu dat pro externí eshop dealera. Pro data se ve většině případů používá formátu XML tzv. feed.

Dále se nabízí řešení pomocí distribuovaných eshopů, které by měli být jednoduché, se snadnou údržbou a úzce orientované. Tak aby pro jejich provoz bylo zapotřebí co nejméně vstupních dat. Jedním z takových řešení je i distribuovaný konfigurátor založený na architektuře SOA, která je popsána v následující kapitole.

Pojmem konfigurátor rozumíme specializovanou část eshopu, která svou funkcionalitou umožňuje zaměnit, přidat nebo vyřadit prvky v sestavě dle požadavků zákazníka. Jako vstupní data mu slouží předkonfigurované sestavy. Konfigurátor musí umět vypočítat cenu změněné sestavy i při omezené funkčnosti prohlížeče např. vypnutém javascriptu. V současnosti jsou konfigurátory součástí eshopů nebo samostatné aplikace běžící v iframu na stránkách dealera.



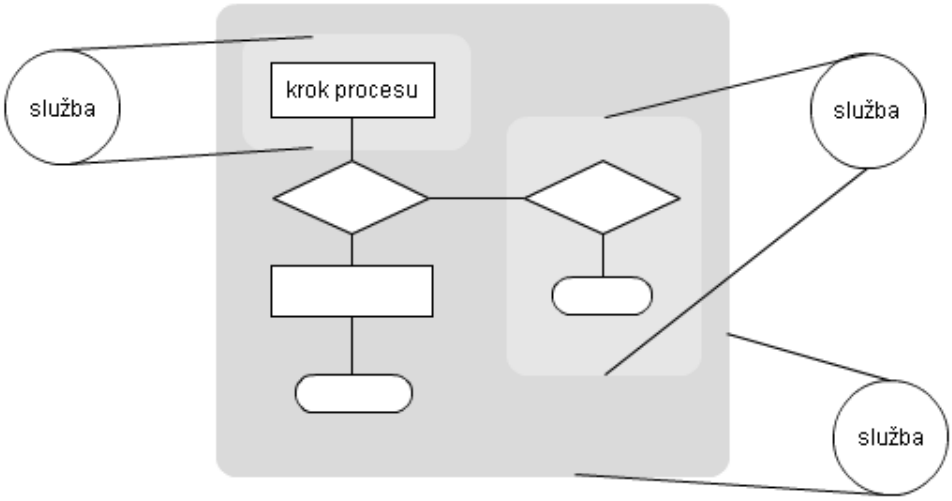
Obr. 2. Současná a navrhovaná podoba základu konfigurátoru

2 SERVISNĚ ORIENTOVANÁ ARCHITEKTURA

Analýza i další vývoj projektu konfigurátoru výpočetní techniky je založen na principu servisně orientované architektury, dále SOA.

SOA je termín, který stále prochází vývojem a i v minulosti byl používán v různých kontextech. Z technického pohledu můžeme obecně SOA definovat jako model, ve kterém je logika automatizace rozdělena na menší jednotky logiky, které mohou existovat samostatně a dohromady tvoří větší kus logiky. Tyto jednotky musí dodržovat principy dovolující nezávislý vývoj při uchování standardizace. Označujeme je jako služby.

Jednotlivé služby mohou zapouzdřovat úlohu vykonávanou v jednom kroku nebo v podprocesu složeném ze skupiny kroků. Služba může dokonce obsáhnout celou logiku procesu. Jednotlivé typy jsou znázorněny na obrázku (Obr. 3).



Obr. 3. Služby zapouzdřují různé množství logiky

Aby spolu mohly služby komunikovat, musí o sobě vzájemně vědět a k tomu slouží popis služeb. Obsahuje název, umístění a požadavky na výměnu dat.

2.1 Základní principy SOA

Na úvod uvedeme základní vlastnosti, které byly formulovány ještě před příchodem webových služeb a tvoří model tzv. tradiční, prvotní SOA.

Volná vazba – služby udržují vztahy, které minimalizují závislosti a vyžadují pouze, aby o sobě navzájem věděly.

Kontrakt služeb – udržují dohodu o komunikaci, definovanou jedním nebo více popisy služeb.

Samospráva – služby mají kontrolu nad logikou, kterou zapouzdřují.

Abstrakce – kromě vlastností popisovaných v dohodě služeb, skrývají logiku před okolím.

Znovupoužitelnost – logika je dělena do služeb za účelem podpory opětovného použití.

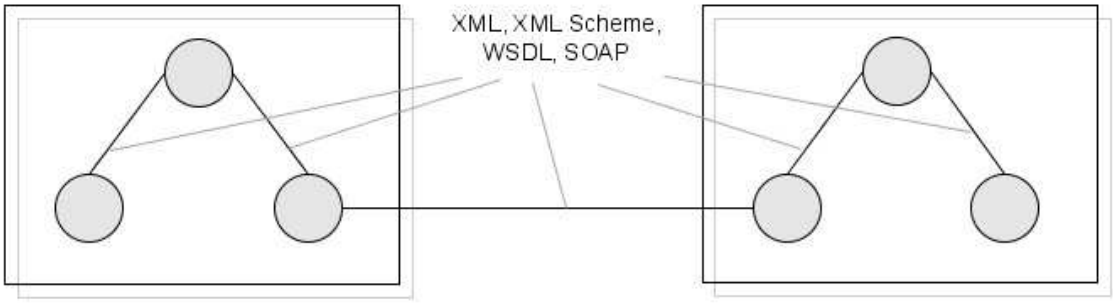
Kompozice – kolekce služeb lze uspořádat a složit, aby vyvořily spojené služby.

Bezstavovost – minimalizace množství uchovávaných informací určených pro další aktivitu.

Zjistitelnost – služby jsou navrženy jako navenek popisné, lze je vyhledat a zhodnotit dostupnými vyhledávacími algoritmy. Existují registry služeb nebo adresář pro správu popisů.

V současnosti je působením průmyslu, technologického vývoje, rozvojem XML a jeho implementací model rozšiřován o další specifikace SOA a servisní orientace jsou vzory nezávislé na implementaci, které lze realizovat na libovolné vhodné technologické platformě.

Výměna dat mezi webovými službami je řízena otevřenými standardy, zprávy mezi nimi jsou posílány prostřednictvím obecně přijímaných a standardizovaných protokolů, to platí i o formátu zprávy a způsobu, jakým popisuje svůj obsah. Tento způsob komunikace dovoluje spojovat neslučitelné platformy (aplikace založené na .NET a J2EE).



Obr. 4. Otevřenost vně i uvnitř řešení

SOA podporuje:

Vnitřní spolupráci – služby jsou přirozeně vybaveny vlastnostmi, které ji podporují, což umožňuje snadnou integraci. Služby s vnitřní spoluprací se stávají potencionálními koncovými body sjednocení.

Federace – schopnost zavést jednotnost do dříve nesourodých aplikací. Při nasazení SOA není nutné nahrazovat původní řešení, jen zapouzdřit starší a novou aplikační logiku a umožnit jim komunikovat.

Architektonickou kompozici – druhá generace webových služeb na základě modulárního charakteru specifikací WS- umožňuje použít jen ty funkční stavební bloky, které potřebuje.

Rozšiřitelnost – bez změny stávajícího rozhraní služby ji můžeme dosáhnout přidáním služby nebo sloučením s jinými aplikacemi. Volná vazba mezi službami umožňuje tento proces s minimálním negativním dopadem.

Implementaci vrstvy abstrakce – běžné servisně orientované architektury nastavují služby jako výhradní přístupové body k prostředkům a procesní logice nebo abstrakce směřuje k řídící a aplikační logice, např. zavedením vrstvy koncových bodů.

Organizační hbitost – díky řídící reprezentaci a abstrakci služeb, volné vazbě mezi řídící a aplikační logikou, které poskytuje vrstva služeb, je se zvyšuje schopnost systému reagovat na neplánované události (reorganizace, změna obchodního zaměření, změna technologické platformy).

2.2 Vývoj SOA

Historický přehled začneme stručným popisem vývojových trendů, které daly podobu platformě SOA, která pak zpětně mění úlohy technologií XML a webových služeb.

2.2.1 Historie XML

Podobně jako HTML i XML (Extensible Markup Language) vznikl z jazyka SGML, většího použití dosáhl na konci 90. let s rozvojem elektronického obchodu. Jazyk XML nesloužil jen pro standardizovanou reprezentaci dat, ale je i základem dalších specifikací např. XSD (XML Schema Definition Language) a XSLT (XSL Transformation Language), které se staly klíčovými částmi jádra technologické sady XML.

Reprezentace dat XML představuje základní vrstvu SOA. Určuje formát a strukturu zpráv přenášených mezi službami. XSD chrání integritu a platnost obsahu zprávy a XSLT umožňuje komunikaci mezi neslučitelnými reprezentacemi pomocí mapování schémat.

2.2.2 WS

V roce 2000 konsorcium W3C přijalo návrh na specifikaci SOAP (Simple Object Access Protocol), založené na ideji, že parametry přenášených dat budou převedeny do XML, přeneseny a zpět navráceny do původního formátu.

Opět rozvoj technologií internetového obchodování vedl, ke vzniku čistě webové technologie standardizované komunikace, která odstraňuje nejednotnost mezi společnostmi i uvnitř nich. Označují se jako webové služby dále jen WS. Jejich nejdůležitější součástí je veřejné rozhraní, jedná se o centrální díl informace, který přiřazuje službě identitu a umožňuje ji volat. Prvním projektem na podporu WS byl proto logicky vznik jazyka WSDL (Web Service description Language). W3C obdrželo první návrh v roce 2001.

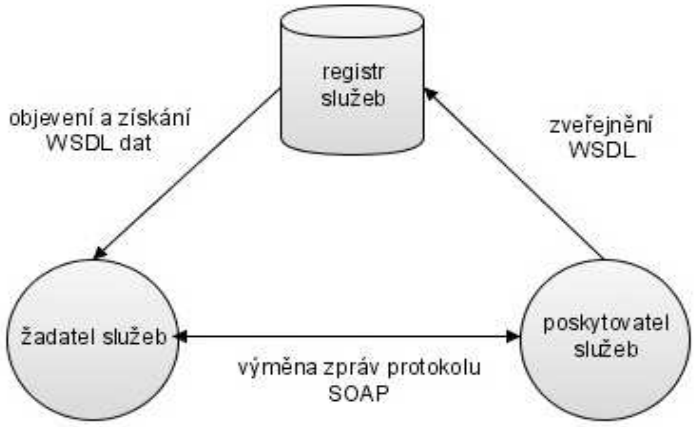
WS požadovaly komunikační formát, který by byl kompatibilní s XML a mohl vytvořit standardní systém komunikace. Byl vybrán protokol SOAP, který W3C v nové verzi specifikací doplnilo o podporu zpráv typu RPC a dokument.

První generaci standardů WS uzavřela specifikace UDDI, která podporuje tvorbu registrů popisů služeb a umožňuje tím centrálně registrovat služby, tak aby byly přístupny

žadatelům služeb uvnitř i vně společnosti. Na rozdíl od WSDL a SOAP nebyla UDDI zatím plně přijata za standard, zůstává tedy volitelným rozhraním SOA.

Na vytváření standardů se podílí celá řada institucí, v následující tabulce uvedu porovnání třech hlavních standardizačních organizací. Samozřejmě i další společnosti, mezi ně patří Microsoft, IBM, Sun Microsystems, sehrály významnou roli nejen při formalizaci specifikací WS, ale také při urychlení implementace těchto specifikací.

Brzy se zjistilo, že WS mohou tvořit základ samostatné architektury, která realizuje systém služeb v podniku. A tak vznikl první model SOA založený na třech součástech: žadatel, poskytovatel a registr služeb.



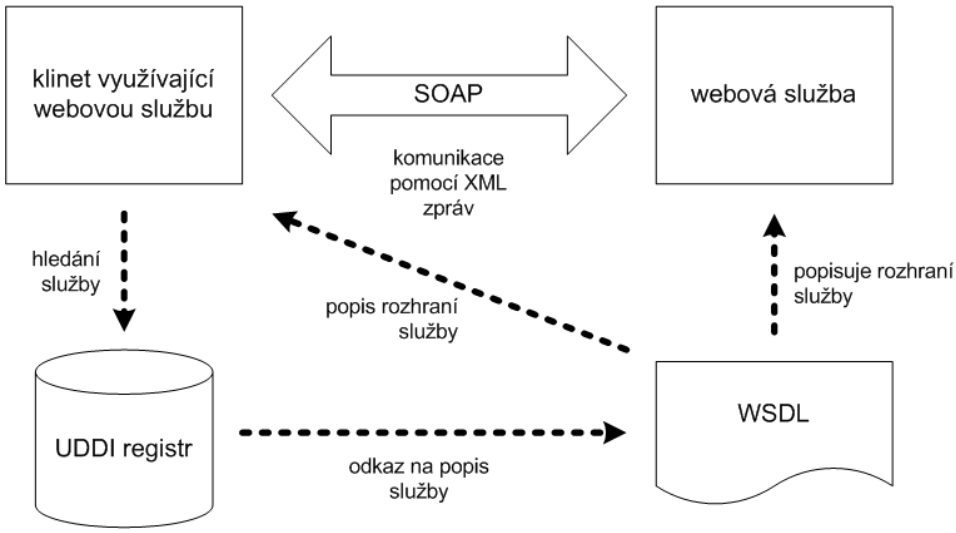
Obr. 5. Raná podoba SOA

WS představují standardní formát propojení mezi různými aplikacemi, které mohou běžet na odlišných platformách. Jsou charakterizovány vysokou součinností a rozšiřitelností, mohou být vzájemně volně kombinovány a vytvářet tak komplexní operace.

Celá infrastruktura webových služeb je založena na třech základních technologiích:

- SOAP (Simple Object Access Protocol) – protokol používaný pro komunikaci
- WSDL (Web Services Description Language) – standardní formát pro popis rozhraní webové služby
- UDDI (Universal Description, Discovery and Integration) – standardní mechanismus umožňující registraci a vyhledávání webov:ých služeb.

Vzájemné vztahy mezi těmito třemi technologiemi jsou zachycené na obrázku (Obr.6).



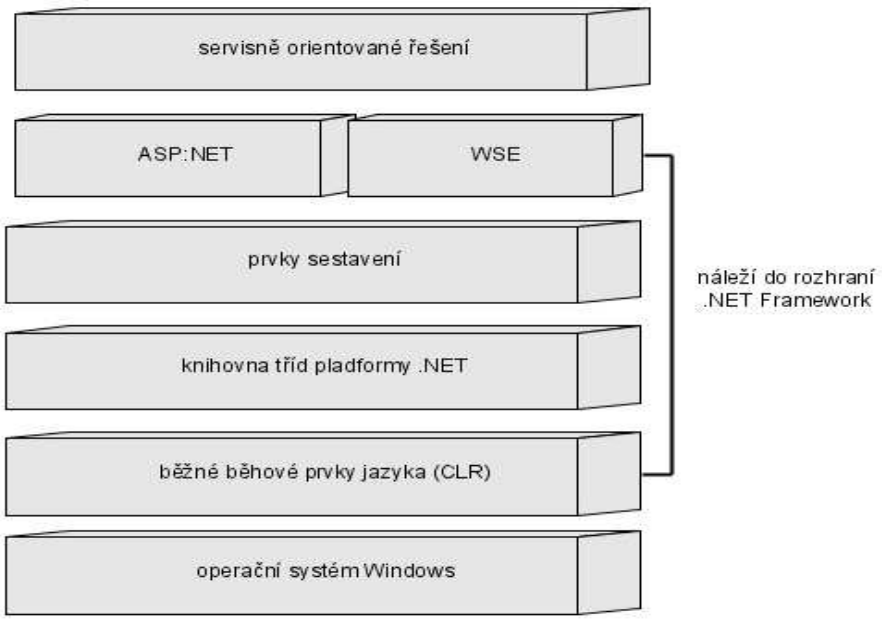
Obr. 6. Vztahy mezi SOAP, WSDL a UDDI

Ke každé webové službě by měl být k dispozici její formální popis v jazyce WSDL. Z tohoto popisu již jde automaticky vygenerovat soapový požadavek. Ve větších systémech nebo přímo v otevřeném prostředí Internetu se popis služby může zaregistrovat do UDDI registru. Ten umožňuje vyhledávání služeb s určitými parametry.

Klient, který chce využít webovou službu, získá buď přes UDDI, nebo přímo její popis. Z něj je jasné, jakou strukturu má mít soapová zpráva a kam se má webové službě poslat, aby ji rozpoznala.

2.3 Podpora SOA na platformě .NET

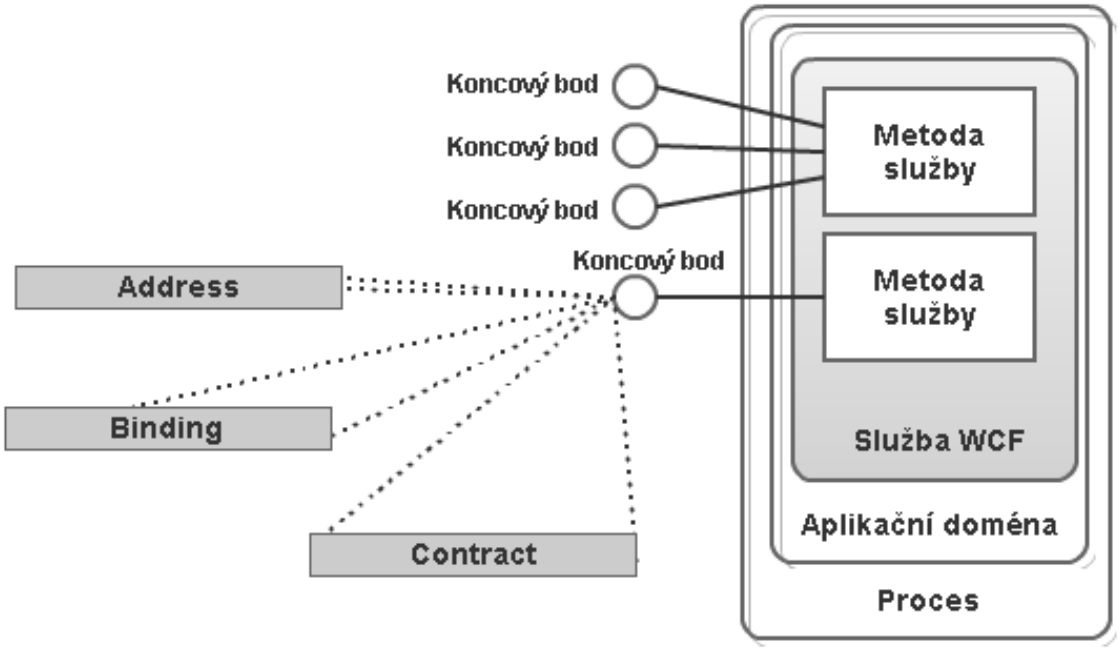
V minulosti jsme pro vytváření SOA aplikací měli několik možnost, příklady takových technologií mohou být ASP.NET Web Services, Web Services Enhancements (WSE), .NET Remoting, Enterprise Services, Microsoft Message Queuing (MSMQ, dodnes hodně využívané hlavně díky spolehlivému doručování zpráv) a další. Velkou nevýhodou těchto technologií je fakt, že pokud budeme chtít psát nějakou aplikaci, bude její vývoj závislý na použité technologii. Dalším faktem je to, že tyto technologie nejsou vzájemně kompatibilní, což nám znesnadňuje práci s nimi.



Obr. 7. Vrstvy rozhraní .NET Framework související se SOA

Od uvedení .NET Frameworku 3.0 v roce 2006 se nabízí nový způsob tvorby webových služeb WCF (Windows Communication Foundation). Hlavním přínosem WCF (známým také jako ServiceModel podle názvu příslušného jmenného prostoru system.ServiceModel) je sjednocený programovací model, jeho návrh a architektura. Služba WCF je tvořena třemi základními součástmi:

- **služba** – třída napsaná v jednom z jazyků .NET, obsahuje jednu či více metod
- **koncové body** – místo, které slouží k přijímání a odesílání zpráv. Tvoří jej tři části: **Address** (udává kde služba běží, tedy kam budou zasílány zprávy), **Binding**(způsob, jakým bude služba komunikovat, tedy jaký komunikační protokol je zvolen, jaké kódování, ale také výběr bezpečnosti, sessions, transakcí atd.) a **Contract**(specifikuje rozhraní, které služba poskytuje, metody a další. Contract je nezávislý na volbě adresy a bindingu.).
- **hostitel** - místo, kde služba poběží, kde bude hostována. WCF služba může být hostována například v IIS (Internetové Informační Službě), klasickém Windows procesu anebo také jako tzv. self-hosting, což může být prakticky jakákoliv aplikace (konzolová, WinForm, WPF).



Obr. 8. Součásti služby WCF

Prvním krokem při tvorbě služby WCF je založení kontraktu, jedná se o třídu s metodami, které bude služba nabízet. Komunikace je založena na zprávách, který je implementován pomocí typu *System.ServiceModel.Chanels.Message* jako informační sadu XML a poskytuje vlastnosti a metody pro navigaci v uzlech SOAP. Každá operace WCF implementuje specifický návrhový vzor pro výměnu zpráv MEP (massage exchange pattern), rozlišujeme typy:

- one way – klient odešle zprávu službě a neočekává odpověď
- request-response – klient pošle požadavek a čeká na odpověď
- duplex – obousměrná komunikace mezi klientem i službou probíhající asynchronně (služba může vynutit spuštění metody na straně klienta)

Datový kontrakt je třída pro stanovení struktury, kterou chceme na rozhraní vystavit. Kontrakty pracují se serializovanými typy. Existuje řada typů serializace, pokud bude probíhat v rámci .NETu stačí pouze přidat *SerializableAttribute*, v případě XML kontraktu začneme definicí XSD. Dále můžeme využít serializační stroj SOAP (používaný ve službách ASMX) ze třídy *XmlSerializer* jmenného prostoru systém.Xml.

Dalším krokem je nadefinování hostitele služby a jeho koncové body. Objektový model WFC poskytuje třídu *ServiceHost* určenou pro implementaci hostitele, definice konfigurace může být zahrnuta přímo v kódu nebo lze podrobnosti koncového bodu zadat do souboru .config a tím oddělit kód od detailů nasazení. K využití vytvořené služby můžete ve Visual Studiu využít hotového klienta s názvem WCF *Test Client*. Kromě této utility je možné nadefinovat klienta (spotřebitele) dvěma dalšími způsoby. První zahrnuje abstrakci od služby samotné, jedná se o definici kontraktu WSDL nezávislého na platformě. Pokud se budeme pohybovat pouze v prostředí WCF a není vyžadována interoperabilitu, nemusíme oddělovat službu a klienta pomocí WSDL a XSD. K tomu slouží typ *ChannelFactory* na straně klienta, který přímo sdílí typy kontraktů a odkazuje se na sestavení definovaná na straně služby.

2.3.1 LINQ pro entity a WCF

Entity ADO.NET Entity Frameworku mají atributy SerializableAttribute a *DataContractAttribute*, je tedy možné je serializovat. Výsledek dotazu LINQ pro entity lze použít jako obsah komunikace WCF.

2.3.2 Podpora základních principů

Zapouzdření služeb – je platformou podporováno. Rozhraní .NET Framework podporuje rozdělení aplikační logiky na atomické jednotky. Tyto nezávislé prvky sestavení mohou být za podpory WS skládány a zapouzdřovány.

Volná vazba - díky společnému použití veřejného rozhraní vytvářeného komponentami a rámce pro předávání zpráv, jako je např. MSMQ, lze mezi aplikacemi dosáhnout volně vázaných vztahů. Kromě toho je využívané rozhraní služeb reprezentované jako WSDL popisy podporované rámcem pro předávání SOAP zpráv.

Předávání zpráv – platforma podporuje jak primární SOAP formát pro předávání zpráv, tak zprávy MSMQ.

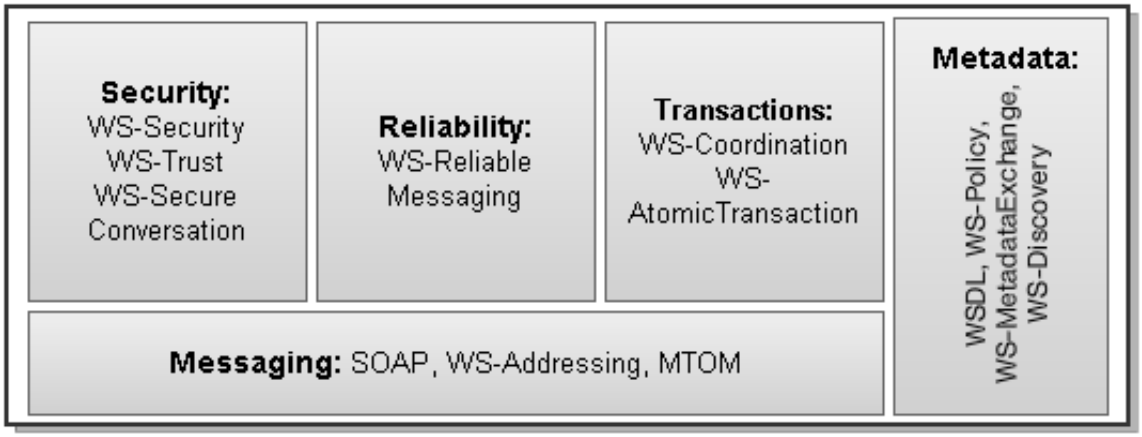
Autonomie – služby, které nesdílí procesní logiku s dalšími prvky sestavení, jsou přirozeně autonomní, jsou plně řízeny vlastní logikou a hostujícími prostředími. Pokud je

nutné zapouzdřit aplikační logiku, která byla navržena pomocí COM komponent, je dosažení autonomie obtížnější.

Znovupoužitelnost – také ji snadněji dosáhneme při tvorbě nového návrhu aplikační logiky.

Bezstavovost – webové ASP.NET služby jsou standardně bezstavové,. Platforma ale umožňuje nastavením atributu *EnableSession* na danou operaci služby vytvořit při vyvolání této operace objekt typu *HttpSessionState*.

Zjistitelnost – je podporována prostřednictvím služeb z oboru názvů System.Services.Discorery nebo sadou nástrojů UDDI SDK.



Obr. 9. WCF implementace Web services standards[1]

[1] http://msdn.microsoft.com/en-us/library/ee958158%28v=MSDN.10%29.aspx

3 ENTITY FRAMEWORK

Dodržování principů SOA vede k vícevrstvému řešení softwarových projektů a tomu neodpovídá standardní způsob dotazování na data v .NET aplikaci. Jedná se o přístup s připojením (*SqlCommand* a *SqlDataReader*) nebo odpojený přístup (*SqlDataAdapter* a *DataSet).* Oba typy mohou ale vést k problémům jednak s dotazováním přímo z kódu a dále s omezením na použití konkrétního databázového stroje. To řeší použití *PbProviderFactoies,* ale stále zůstává slučování různých typů kódu a syntaxí, práce blízko databáze. Řešením může být použití ADO.NET Entity Frameworku, který umožňuje modelovat entity na konceptuální úrovni abstrakce. První verze Entity Framework dodáván v. NET Framework 3.5 SP1 a Visual Studio 2008.

3.1 Datové modelování entit

Model entit je tvořen 3 soubory s definicí konceptuálního schématu a jeho mapováním na schéma fyzické. Je založen na množině schémat XML, ta jsou dále analyzována v nástroji, který generuje odpovídající kód implementace v .NETu. Souhrnně se tyto procesy označují datové mapování entit EDM (Entity Data Modeling).

První soubor, zapsaný v definičním jazyce konceptuálního schématu **CSDL** (Conceptual Schema Definition Language), popisuje entity bez ohledu na databázi a atributy jsou definovány pouze pomocí typů .NETu. Obsahuje pravidla pro ověření a omezení. Jsou zde použity položky typu:

- EntityType – konkrétní entita
- EntitySet – skupina entit
- Assotiation – vztah mezi entitami.

Druhý soubor, v definičním jazyce schématu úložiště **SSDL** (Storage Scheme Definition Language), popisuje fyzickou datovou vrstvu. Obsahuje primární klíče, pravidla pro povolení nul, identity, cizí klíče a datové typy ze strany databázového stroje (SQL Serveru,...).

Poslední soubor s metadaty je napsán v definičním jazyce schématu mapování **MSL** (Mapping Schema Language). Zde dochází k mapování mezi konceptuálním modelem

(CdmEntityContainer) a fyzickým úložištěm (StorageEntityContainer). Všem skalárním vlastnostem (Name) každé entity (TypeName) je přiřazena příslušná tabulka (TableName) a sloupec (ColumnName).

Na kód se převádí pouze soubor CSDL, zbývající dva SSDL a MSL zůstávají v XML, což umožňuje měnit fyzickou vrstvu bez změny v konceptuálním modelu.

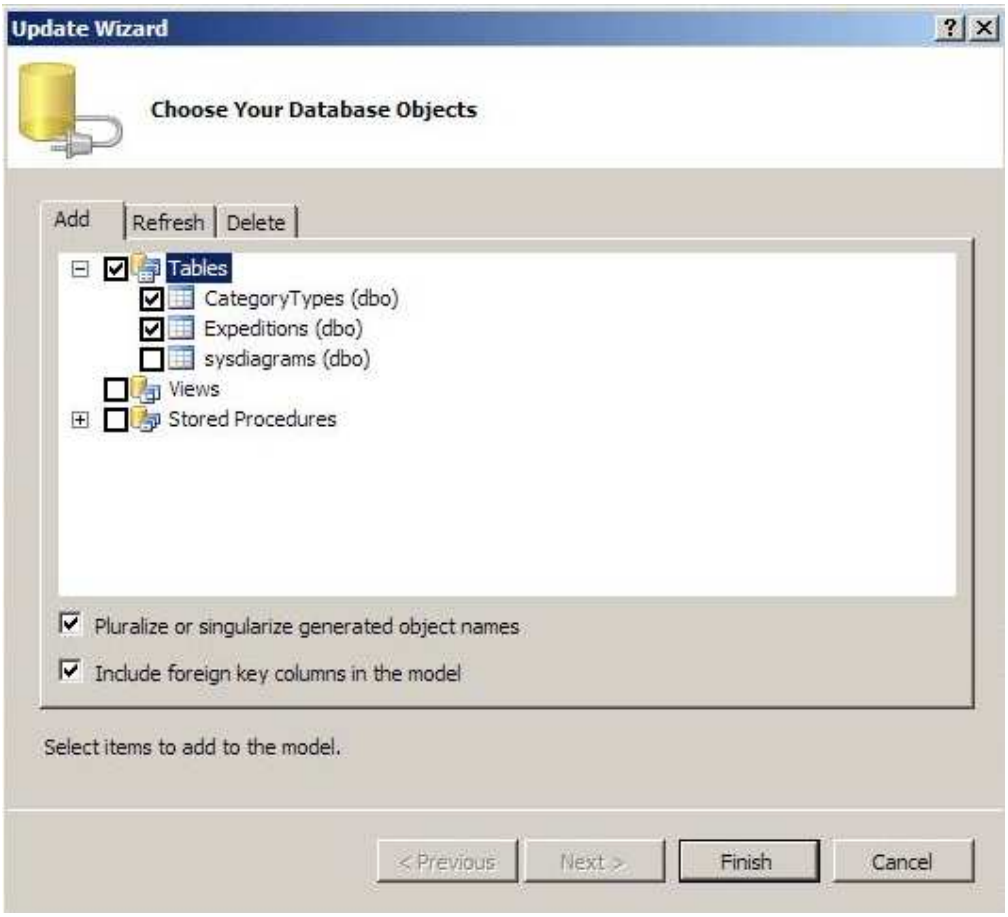
3.1.1 Návrhář a průvodce pro datový model entit

Visual Studio 2010 nabízí integrovaný návrhář, který umožňuje tvorbu konceptuálních schémat od samého začátku nebo z existující databázové vrstvy. Výsledkem průvodce je XML soubor .EDMX vnitřně reprezentující všechny tři soubory modelu. Webový projekt ASP.NET nadefinuje .EDMX soubor jako EntityDataModelItemTemplate, bude umístěn ve složce App.Code a do do sestavování se jako zdroje zahrnou soubory CSDL, SSDL, MSL.

Entity Data Model designer, který je součástí Visual Studia 2010 nabízí, kromě možnosti vygenerovat entitní model z již existující databáze, Model-first. Kdy první vytvoříme model a z něho vygenerujeme databázi. Další novinkou je možnost při generování modelu entit z tabulek již existující databáze využít k tvorbě názvů pluralizace. Jedná se o jednoduchou pluralizační službu pro anglické názvy, kterou lze dále upravovat. V singuláru vytvoří názvy EntityType a NavigationProperty 0..1, v plurálu názvy pro EntitySet a NavigationProperty 0..n. Výsledný model se tak stává srozumitelnější a vyžaduje menší následné úpravy.

Tak jako celé Visual Studio 2010 podporuje EF4 a EDM Multi-Targeting capability. Můžeme pokračovat v tvorbě aplikace EF3.5 nebo ji převést do EF4. Verze EDMX je automaticky upravena.

Do EF4 byl implementován nový typ vazby ForeignKey. Při generování modelu entit v EDM je nutné ve stromu vybrat Blogs, Poststables položky Tables a dál zvolit Include ForeignKey Columns in the Model. Ve vytvořeném modelu bude přednostně použita asociace cizím klíčem před volnou. Ta je samozřejmě dále přístupná a typ vztahů je možné měnit v dialogovém okně vlastnosti vazby. Obě asociace lze vytvořit i ručně.



Obr. 10. Možnosti generování modelu entit

Entity Data Model Designer ve VS 2010 umožňuje definovat komplexní typy, zobrazit je v model explorer tree a refaktorovat stávajících vlastnosti do komplexního typu.

Mapování entit lze měnit v okně podrobností mapování, jak ukazuje následující obrázek.

Obr. 11. Mapování entit na fyzické úložiště dat

Lze nastavit mapování každé vlastnosti entity na odpovídající tabulku a sloupec v databázi. V samostatném okně můžete procházet graf modelu, jeho uzly jsou sdruženy do dvou

hlavních množin popisujících konceptuální model a model cílového úložiště dat. Pomocí návrháře lze spravovat vztahy mezi entitami a jejich odpovídající kardinalitu. Pokud importujeme do modelu tabulky se vztahy, návrhář tyto vazby odvozuje a spojuje tak entity. Jsou-li dvě tabulky propojeny pomocí tabulky se vztahem m:n, návrhář tuto tabulku vynechá a ve schématu entit zobrazí příslušný vztah.

ADO.NET Entity Framework vyžaduje, aby všechny entity měly identifikační klíče.

3.2 POCO, IPOCO

ADO.NET Entity Framework podporuje mnoho poskytovatelů dotazů, a to prostřednictvím otevřeného a veřejného modelu poskytovatelů dotazů. Nabízí dva odlišné scénáře:

normativní třídy (prescriptive classes) – typy, které dědí ze specifické základní třídy, pracující přímo s trvalým úložištěm. Obvykle jsou tyto třídy vygenerovány nějakým podpůrným nástrojem.

IPOCO (Interface-Based Plain Old CLR Objects) – typy entit, které musejí implementovat určitá specifická rozhraní, aby mohla podporovat funkce jako perzistence, sledování změn, vztahy, identifikace klíčů atd.

EF4 umožňuje vytvářet třídy bez nutnosti implementovat specifická rozhraní, podporující perzistence ignorance. Dynamické proxy servery podporují lazy loading a efektivní sledování změn. Umožňuje to šablona POCO Template dostupná ve Visual Studio Extensions Gallery, více na:

http://blogs.msdn.com/b/adonet/archive/2010/01/25/walkthrough-poco-template-for-the-entity-framework.aspx

Tato šablona umožňuje dodržet jeden z hlavních požadavků abstrakce datové vrstvy: neznalost perzistentního modelu (perzistence ignorance). Šablona je rozdělena na dvě části, první generuje Entity Types a Complex Types, druhá strongly typed context.

3.2.1 Šablonovací systém T4

Šablonovací systém T4(Text Templating Transformation Toolkit) jednoduše umožňuje vytvářet soubory na základě, i velmi komplexních, šablon. Visual Studio používá tento

nástroj pro generování kódu. Ve Visual Studia 2010 je lehce dostupný v okně Add New Item, kde lze vybrat T4 šablony. Nyní můžete napsat vlastní šablony, které určují přesně, jak chcete upravit generování kódu, nebo můžete upravit vestavěné šablony, jako je Entity Object Code Generator atd.

Šablony T4 můžete využívat v jakémkoli produktu. Vlastní provádění zajišťuje Visual Studio (ne runtime) a pokud např. generujete soubor se zdrojovým textem, je do výsledné kompilace přidán pouze tento výsledek transformace.

Šablona je soubor s příponou .tt, vše co obsahuje, bude ve výsledném, vygenerovaném, souboru. Aby měla šablona nějaký význam, potřebuje speciální symboly, které nám umožní tvořit výsledek pomocí běžných konstruktů jako for, if apod. V šablonách T4 jsou tyto speciální elementy uzavřeny v <# a #>. Uvnitř je pak možné používat např. C# nebo VB.NET.

T4 šablony lze použít dvojím způsobem:

- Generování kódu šablonou během kompilace projektu - vlastní šablonu, která z Entity modelu vygeneruje datové třídy.
- Generování za běhu programu - šablona se zkompiluje a za běhu aplikace je možné ji naplnit daty a nechat z ní vygenerovat výsledný text.

3.3 Lazy loading

Další novinkou EF4 je automatická podpora lazy loading, které je dostupné jako výchozí nastavení. Pro získávání dat z SQL databází existují různé typy načítaní.

Lazy loading - při tomto přístupu jsou data získávána tak, jak programátor přistupuje k proměnným a prochází grafem. Výhodou tohoto přístupu je jednoduchost pro programátora. Nemusí se starat o to, zdali má data načtena atp. Na druhou stranu nemusí být vždy jasné, v jaké části programu skutečně k načtení dojde. Tento přistup je vhodný pro několik málo dotazů do třeba číselníků. Jakmile iterujeme přes větší kolekci a dotahujeme další údaje, plýtváme zbytečně prostředky a časem, neboť se dotazy do databáze posílají jednotlivě.

Eager loading - data zde načteme ihned při spuštění prvního příkazu, v Entity Frameworku využíváme metody „Include“, která nám zajistí vytvoření spojení (join) tabulek a získání všech dat přímo.

```
foreach(Customer customer in entities.Customers.Include(„Orders“))
{
  foreach(Order o in customer.Orders)
  { ... }
}
```

Díky tomu máme všechna data dostupná na klientovi a při procházení již není třeba dělat další dotazy do databáze. V případě procházení celé kolekce a přidružených objektů je tento přístup více než vhodný. Na druhou stranu je třeba velmi opatrně další tabulky přidávat, neboť je možné velmi jednoduše vytvořit „velký“ dotaz, který vrátí enormní množství dat a dojde k zahlcení linky nebo klienta.

Explicit loading - standardní v předchozí verzi Entity Frameworku a LINQ to Entities. Načítání dat si řídíte sami, podobně jako u eager loading, nicméně získáváme pouze menší kousky dat podobně jako u lazy loading.

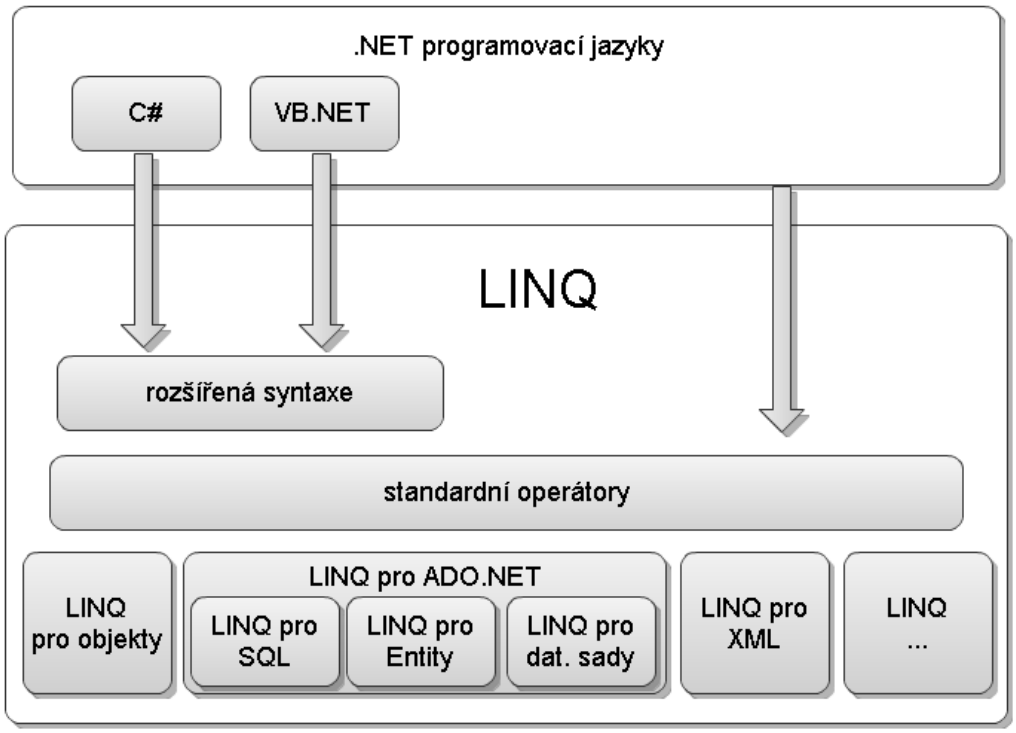
```
foreach(Customer customer in entities.Customers)
{
  if (!customer.Orders.IsLoaded)
    customer.Orders.Load();
  foreach(Order o in customer.Orders)
  { ... }
}
```

Při procházení kolekce, chceme-li dostat objednávky konkrétního (právě zpracovávaného) zákazníka, musíme před zpracováním této kolekce zavolat metodu *Load*. Díky schopnosti Entity Frameworku cacheovat načtené objekty není třeba data získávat znova, byla-li již jednou načtena (property *IsLoaded*). Výhodou tohoto postupu je jasná viditelnost, kdy k načítání dat dochází (resp. kdy se o to kód pokusí). Nevýhodou je naopak, při procházení velkých kolekcí, nutnost poslat velký počet dotazů (zde je vhodnější použít eager loading

4 LINQ

Integrovaný jazyk pro dotazování LINQ (Languafe Integrated Query) umožňuje vývojářům dotazovat se a spravovat sekvence položek (objekty, entity, databázové záznamy, uzly XML atd.) v jejich softwarovém řešení pomocí běžné syntaxe a jednoho programovacího jazyka bez ohledu na charakter zpracovávaných položek. Klíčovou vlastností LINQ je jeho integrace v běžně používaných programovaných jazycích, kterou umožňuje používání společné syntaxe pro všechny druhy obsahu.

Technologie LINQ je navržena tak, že je možné tvořit její implementace pro jednotlivé datové zdroje. Je to podobné jako v ADO .NET, kde je možné implementací rozhraní, vytvořit .NET provider pro specifický typ databáze, avšak v LINQ tato rozšiřitelnost nezůstává jenom u relačních databází a je mnohem abstraktnější. Této volnosti je dosaženo hlavně díky Expression Trees, jako novinka implementováno v .NET 3.5, jež umožňují pracovat s kódem jako z daty a tím pádem může být LINQ dotaz konkrétním providerem přeložen pro adekvátní datový zdroj a je jedno jestli se jedná o data relační, hierarchická či nějaká jiná.



Obr. 12. Implementace LINQ

Všechna dotazovací rozšíření vycházejí specializovaných metod a sdílejí stejnou sadu klíčových slov pro dotazovací výrazy. LINQ vychází z množiny dotazovacích operátorů definovaných jako rozšiřující metody, které pracují s jakýmkoliv objektem iplementujícím rozhraním *IEnemerable<T>* nebo *IQueryable<T>*.

Tab. 1. Metody LINQ[2]

Klíčové slovo	Metoda na IEnumerable<T>	Popis
from	žádná	První klíčové slovo v dotazu. Slouží ke specifikaci datového zdroje, nad kterým je prováděn dotaz.
where	Where	Slouží k definici podmínky restrikce pro výsledek. Pokud podmínka vrátí true, prvek je zahrnut do výsledku.
select	Select	Používá se k implementaci projekce, kde jsou pouze některé datové složky objektu použity ve výsledku. Uvádí se vždy na konci dotazu.
group	GroupBy	Slouží k seskupování prvků ve výsledku podle určitého klíče. K použití toho slova se váže nové klíčové slova by. Může jím být zakončen dotaz.
into	žádná	Používá se v kombinaci se slovem group, join nebo select k uložení jeho výsledku a další možné práci s tímto výsledkem.
orderby	OrderBy, OrderByDescending	Slouží k řazení prvků ve výsledku podle definovaných kritérií. Pro sestupné řazení je možné použít s novým klíčovým slovem descending.
join	Join, GroupJoin	Používá se k propojení prvků z různých datových zdrojů na základě definované podmínky ekvivalence. V kombinaci s tímto slovem se používají nová klíčová slova equals a on.
let	žádná	Slouží k definici lokální proměnné v rámci dotazu, do které může být přiřazena jak sekvence elementů, tak jednoduchá hodnota.

[2] http://www.vyvojar.cz/Articles/563-uvod-do-linq.aspx

Obecná forma jednoduchého LINQ dotazu tedy vypadá nějak takto:

```
from [typ] proměnná in datový_zdroj
[where] podmínka_restrikce
[orderby] klíč_řazení [descending]
select výraz_projekce
```

Rozhraní *IQueryable<T>*, které je od *IEnumerable<T>* odvozené a které přidává především property Provider typu *IQueryProvider*. Pro *IQueryable<T>* také existuje implementace všech LINQ metod – a to ve třídě *System.Linq.Queryable* (opět z assembly *System.Core*). Tato implementace vypadá tak, že přijímá povětšinou expression tree a předává ho k vyhodnocení konkrétnímu providerovi (*IQueryProvider*). Zjednodušeně řečeno můžeme implementace LINQ metod nad rozhraním *IQueryable<T>* rozdělit na dvě skupiny – jedny volají ve finále na provideru metodu *CreateQuery*, která opět vrací *IQueryable<T>* a tyto metody tedy často vůbec nekomunikují s datovým zdrojem a slouží jen pro sestavení dotazu. Příkladem takových metod je například Select, Where nebo Join. Druhou skupinou metod jsou ty, které na providerovi volají metodu Execute – takové metody už vyžadují komunikaci s datovým zdrojem a vrací přímo nějaký výsledek. Typicky jsou to metody, které něco počítají nad všemi výslednými "řádky", tedy např. metody Min, Max, Sum nebo Average.

U první skupiny metod je tedy dobré si uvědomit, že slouží jen k vytvoření dotazu, ne přímo k jeho vykonání. Dotaz je pak skutečně vykonán až tehdy, kdy je to skutečně potřeba, tedy např. při volání metody *GetEnumerator* (volá ji sám foreach). Pokud potřebujeme ale vyhodnotit dotaz ihned (např. proto, že nechceme nebo nemůžeme držet otevřené připojení do databáze), ale ještě nechceme enumerovat přes výsledek, musíme nějak donutit *IQueryable<T>* výsledek vrátit, tedy zavolat metodu Execute příslušného providera, která nám vrátí konkrétní výsledek (typicky dotazem do datového zdroje). To můžeme udělat např. pomocí extension metody ToList nebo ToArray, které vyenumerují všechny položky výsledku a uloží je do Listu, resp. pole.

Abychom mohli provádět dotazy nad vlastní třídou (to bude ale asi v praxi řešit málokdo), musí tedy tato třída implementovat rozhraní *IQueryable<T>*, což je *IEnumerable<T>* rozšířená především o property typu *IQueryProvider*. Musíme tedy implementovat ještě

toto rozhraní, které je zodpovědné za vlastní vytváření a spouštění dotazů (metody *CreateQuery* a *Execute*).

4.1 LINQ pro Entity

LINQ se poprvé jako technický náhled objevil v září 2005. Od listopadu 2007 se stal integrální součástí .NET Frameworku 3.5 a Visual Studia 2008. LINQ přímo podporoval několik datových zdrojů, ale až v průběhu roku 2008 byl doplněn o LINQ pro Entity.

Implementace LINQ umožňuje dotazovat se do datového modelu entit ADO.NET Entity Framework pomocí syntaxe LINQ. Obrázek ukazuje architekturu, která přímo vychází z infrastruktury navrstvených komponent ADO.NET Entity Framework.

Obr. 13. Architektura LINQ pro Entity (12)

Na LINQ pro Entity jsou přímo napojeny *ObjectService,* který zpracovává identitu a stav objektů, a EntitySQL, nabízející dotazovací možnosti nad datovým modelem entit.

Příklad jednoduchého dotazu nad ukázkovým datovým modelem entit Northwind, který vybere všechny zákazníky z Itálie.

```
var northwind = new NorthwindModel.NorthwindEntities();
var italianCustomers =
      from c in northwind.Customers
      where c.country == “Italy”
      select c;
```

Syntaxe dotazů je stejná jako v LINQ pro objekty či LINQ pro SQL. Dotaz je zde převeden na dotaz v příkazovém stromu Entity Framework a pak se provádí v objektu typu *ObjektContext.* V porovnání s metodami LINQ pro objekty tu nejsou dostupné projekční a omezující metody, které přebírají parametr pozice, vlastní metody Aggregate, omezení pokročilého stránkování (TakeWhile, SkipWhile). Důvodem je, že nelze převést každou operaci exekutivního a vývojového prostředí CLR () na odpovídající příkaz SQL a každý DBMS nemá stejné možnosti a u Entity Framework je zásadní vlastností nezávislost na použitém databázovém stroji.

V EF4 byla doplněna podpora dalších operátorů pro LINQ např. Contains, Single, SingleOrDefault a DefaultIfEmpty.

LINQ pro Entity dává možnost využívat při dotazování v modelu entit vztahy. Dotaz se nenačítá do paměti celou množinu entit, ale pouze potřebné entity. Využívá se kolekcí typu *EntityCollection*<> s pravidlem odloženého načítaní, k načtení z DBMS dojde až po zavolání metody Load. Metoda Include umožňuje vkládání další entity do hlavní množiny entit, na kterou se dotazujeme. Metoda UnionAll nepracuje při použit v dotazu, ale pouze s instancemi třídy *ObjectQuery*<*T*>.

4.1.1 Správa dat

Pro změny a aktualizace entit se využívá komponenta *ObjectService* ADO.NET Entity Framework. Pro uložení dat natrvalo používáme metodu *SaveChanges*, pokud je nutné provádět transakční operace, můžeme kód vložit do obálky *TransactionScope*. Tento postup je použit v následující ukázce.

```
using (NorthwindEntities db = new NorthwindEntities()) {
      using (TransactionScope scope = new ransactionScope()) {
            var customer = (from c in db.Customers
                        where c.CustomerID ==“ALFKI“
                        select c.).AsEnumerable().First();
      customer.ContactName = “X Y“;
      db.SaveChanges();
      scope.Complete();
      }
}
```

Provádění dotazu LINQ pro Entity vede k vytvoření dotazovacího stromu. Vnitřně existuje poskytovatel pro dotazy LINQ, implementovaný pro třídu OQ, který prochází dotazovací strom LINQ a převádí ho na strom příkazů zděděných ze třídy *DbCommandTree*. Strom příkazů se provádí proti objektu typu *ObjectContext* a vrací množinu dat podle pravidel odloženého provádění dotazu v LINQ. Každý strom příkazů lze vyhodnotit na klientské i serverové straně. Klientské vyhodnocení umožňuje nahradit výrazy, které je potřeba vypočítat na klientské straně, odpovídajícími hodnotami. Serverové vyčíslení odpovídá provádění dotazu na fyzickém DBMS pomocí standardních komponent ADO.NET.

Po provedení dotazu jsou výsledky převedeny na typy CLR, tuto fázi nazýváme materializace. Když je výsledkem dotazu množina entit z datového modelu, sleduje entity, materializované a navrácené do výkonného kódu, objekt typu *ObjectContext* a entity mají plnou podporu Entity Frameworku. To znamená, že se pro tyto entity budou sledovat změny, identita atd. Jestliže je výsledkem dotazu množina instancí anonymního typu či množina primitivních výrazů CLR, nedochází k jejich sledování objektem *ObjectContext*.

V LINQ pro Entity se všechny plány dotazů ukládají do mezi paměti automaticky, tím lze urychlit jejich provádění, používají-li se v kódu opakovaně. Toto chování je samozřejmě vypnout pomocí syntaxe

```
((ObjectQuery<T>)query).EnablePlanCaching = false;
```

Dalším ze způsobů optimalizace v LINQ pro Entity je použití kompilovaných dotazů. Jedná se o delegáta předkompilované instance typu *ObjectQuery<T>*, který vrací výsledek *IQueryable<T>*. Používá se při opakovaném volání dotazu, kdy se mění pouze parametry. Metoda Compile, dostupná v typu *CompiledQuery*, je generická metoda nabízející čtyři přetížení. Umožňuje definovat předkompilované dotazy až se třemi parametry. Prvním je

samotný *ObjectContext*, jenž je povinný, jako další typ nepovinných parametrů a na závěr typ výsledku. Pokud je nutné definovat dotaz s více než třemi parametry, je možné vložit parametr typu pole nebo strukturovaný typ (třídu atd.).

4.1.2 Výrazy lambda

LINQ umožňuje používat ve své syntaxi lambda výrazy, které zavádí již C# 3.0 a běžně jsou používány ve funkcionálním programování. Pomocí lambda výrazů je možné tvořit anonymní metody, které obsahují jeden výraz nebo několik příkazů a použít je kdekoli kde je očekávána instance delegáta. S lambda výrazy přichází do jazyku C# nový operátor =>, který umožňuje stručnější syntaxi. To je jedním z důvodů jejich používání v LINQ, dalším je tvorba stromu výrazů.

Obecný tvar lambda výrazu je v této podobě:

```
(vstupní argumenty) => výraz
```

Závorky nejsou nutné pouze v případě, že vstupní argument je jediný. Datové typy argumentů není nutné uvádět, protože jsou stejně jako v případě použití klíčového slova var odvozeny v době kompilace na základě použitého typu delegáta.

Jelikož lze lambda výrazy použit všude, kde je očekávána instance delegáta, můžeme použit i nový generický delegát Func<> z .NET Frameworku 3.5, kde je formou generických argumentů specifikováno jaké jsou typy vstupních argumentů a také jaký je návratový typ.

Výrazy lambda mohou také případně odložit generování kódu, když vytvoří strom výrazů. Strom výrazů povoluje další operace (procházení a vytváření obdobných stromů) před skutečným generováním kódu, k němuž dochází za běhu. Strom výrazů lze generovat pouze pro konkrétní části kódu.

5 MICROSOFT SYNC FRAMEWORK

Jedná se o sadu objektů pro synchronizaci různých zdrojů. K dispozici jsou nyní:

- Database synchronization providers – synchronizace ADO.NET zdrojů
- File synchronization provider - pro soubory a adresáře
- Web synchronization components - jednoduché zdroje (RSS, Atom, …)

Je možné vytvořit vlastní 'provider' pro výměnu informací mezi libovolnými zařízeními nebo aplikacemi.

Základním prvkem je tzv. účastník (participant). Účastníky rozdělujeme do tří kategorií, neboť ne všechna zařízení nebo zdroje mají stejné schopnosti:

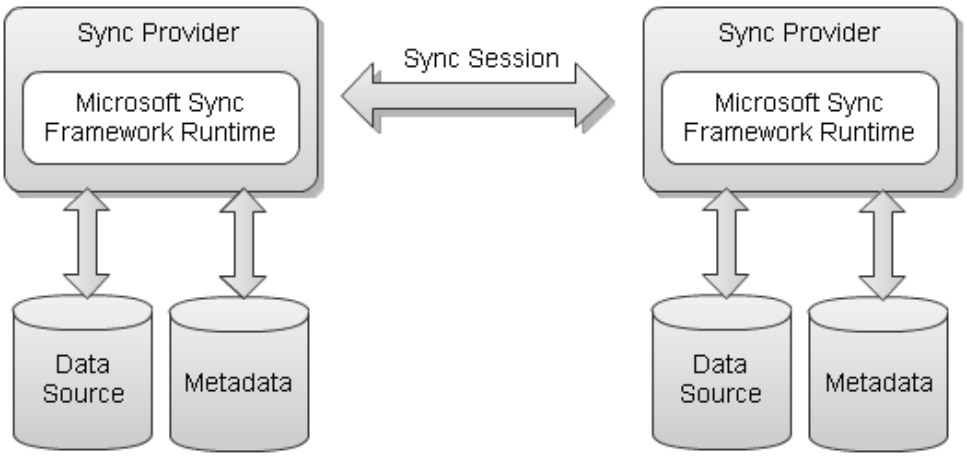
Full participant (úplný účastník) je zařízení, které dokáže ukládat přímo k sobě data a dovoluje spouštět aplikace (na sobě). Klasická ukázka je počítač, PDA atp.

Partial participant (částečný účastník) naproti tomu, je zařízení, které dokáže pouze data ukládat. Musí však umět ukládat i metadata potřebná k synchronizaci. Může se tedy jednat o flash disk, MP3 přehrávač.

Simple participant (jednoduchý účastník) je nejjednodušší forma. Dokáže pouze data poskytovat. Nejde na něj data ukládat (z pohledu aplikace). Typickou ukázkou je RSS feed či jednoduchá webová služba.

Do každého procesu synchronizace musí být zapojen alespoň jeden úplný účastník. Zbylé možnosti kombinace synchronizací jsou plně v rukou vývojáře.

Následující obrázek ukazuje komunikaci mezi dvěma providery přes synchronizační session, DataSource je vlastně úložiště informací, které budeme synchronizovat, resp. jeho identifikace. Může se jednat o relační databázi, složku na flash disku, webovou službu či jeden



Obr. 14. Synchronizace

Základní schopností providera je ukládání informací o uložených datech s ohledem na jejich stav a možnost měnit je. Tato metadata je možné ukládat pomocí souboru, databáze. Microsoft Sync Framework nabízí kompletní implementaci vytvoření a uložení metadat prostřednictvím databáze, ale možné napsat si vlastní. Metadata obsahují pět komponent:

version, tick count, replica ID, knowledge, tombstone.

Pro všechny synchronizované položky jsou ukládány dvě verze tzv. creation version a update version. Každý tento element se skládá ze dvou částí: tick count (logické hodiny) a „replica ID“ (identifikace obrazu). Obě verze jsou při vzniku shodné, ale creation version se už dále nikdy nemění. Existují dva typy sledování verzí:

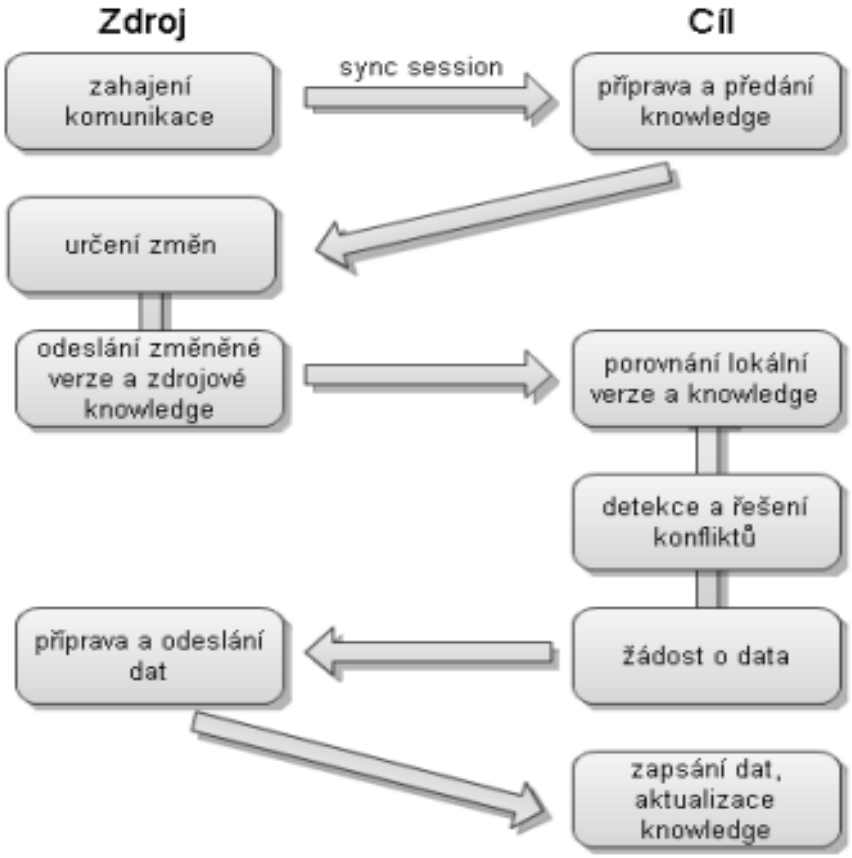
- **Inline tracking** - změny jsou aktualizovány hned, např. v databázi při uložení dat v řádku.
- **Asynchronouse tracking** - externí proces, který vyhledává změny a zaznamenává je do informací o verzi. Může být součástí procesu nebo být proveden před synchronizací, obvykle pomocí porovnání, např. u souboru velikost, čas uložení (last-write-time).

Knowledge je relativně úsporná informace o změnách v synchronizovaných položkách. Kromě přehledu změn, je využívána k rozpoznání konfliktů. Hlavní význam je ušetřit množství informací, které si vyměňují účastníci při synchronizaci a tím i zlepšit efektivitu.

Tombstone slouží k udržování informací o položkách, které byly odstraněny. Skládá se z Global ID, Deletion version, Creation version. Udržovány jsou jakoby kopie informací o položkách. Objem těchto informace stále roste a proto Sync Framework nabízí nástroje na jejich správu a mazání.

5.1 Průběh synchronizace

Synchronizace je popsána v následujícím schématu. Pro obousměrnou synchronizaci je celý proces proveden dvakrát, zamění se pouze zdroj a cíl.



Obr. 15. Průběh synchronizace

Zdroj zahájí relaci a ptá se cíle na knowledge. Jakmile ji dostane, zjistí, jaké položky bude třeba přenášet a tuto informaci pošle zpět. Cíl zjistí, jaké položky bude třeba poslat a také určí konflikty. Konflikt je detekován, když verze neobsahují knowledge o druhé, především neobsahuje-li zdrojová knowledge cílovou verzi. Jestliže je zdrojová verze obsažena

v cílové, nejsou změny akceptovány. Pokud konflikty vznikly, jsou zde vyřešeny či odloženy. Základní principy řešení jsou:

- Source Wins – prioritu mají změny provedené ve zdroji.
- Destination Wins – prioritu mají změny provedené ve zdroji.
- Specified Replica ID Always Wins – je vybrána verze s předem určeným Replica ID.
- Last-Writer Wins – vychází z předpokladu, že je vše synchronizované a vybrána změna provedená nejpozději
- Merge - ve speciálních případech je možné použít sloučení. Tento způsob ale přirozeně nejde použít vždy.
- Log Conflict – můžete konflikt odložit.

Následuje požadavek na zdroj, aby poslal dané položky, a cíl je poté zanese lokálně. Pokud se vyskytnou nějaké chyby během tohoto procesu, jako je například výpadek sítě, budou položky označeny jako výjimky a opravena v průběhu příští synchronizaci. Posledním krokem je úprava knowledge.

II. PRAKTICKÁ ČÁST

6 ANALÝZA

6.1 Požadavky

Cílem práce je vytvořit nástroj pro konfiguraci počítačových sestav, který bude dále použitelný i pro výběr volitelných vlastností spotřební elektroniky. Tato aplikace má být součástí B2C internetového obchodu. Dále uvedu seznam konkrétních požadavků:

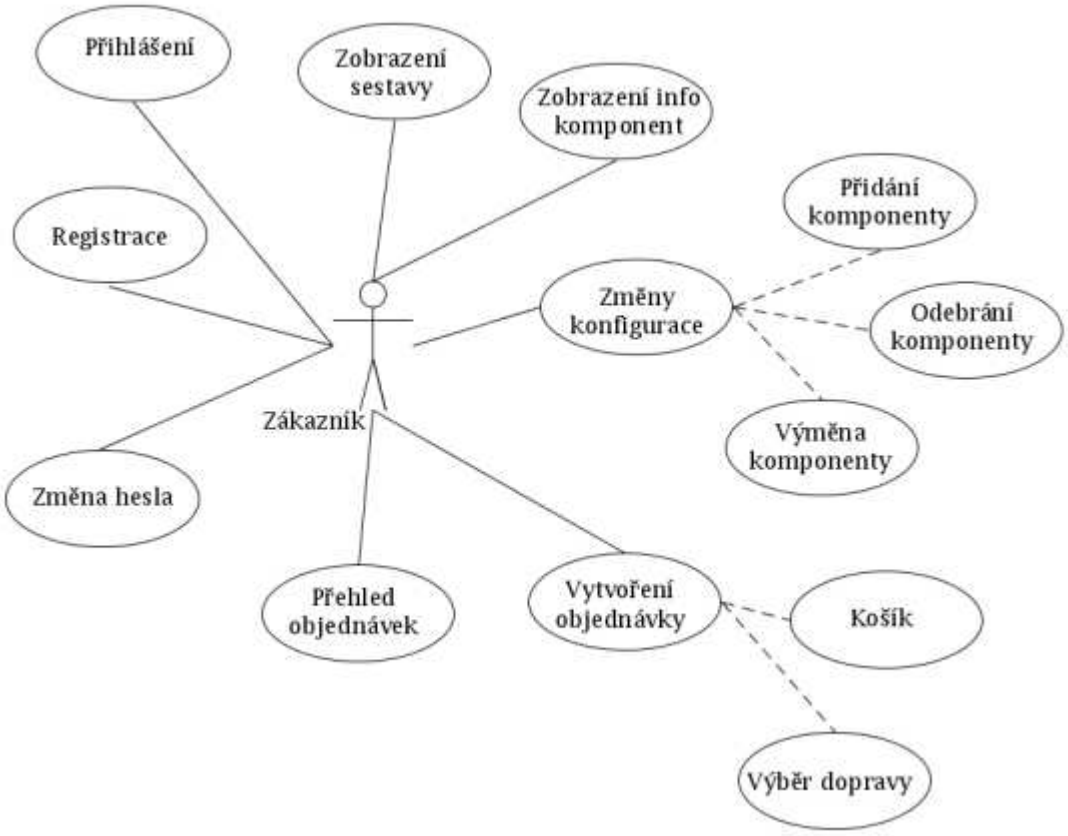
- Zobrazit nabídku jednotlivých kategorií, složení sestav, popis a obrázky k jednotlivým komponentám.
- Ke každé defaultní komponentě nabídnout seznam možností změny, ke každé sestavě zobrazit přehled komponent, které lze přidat. Při konfiguraci měnit cenu.
- U jednotlivých komponent lze určit, zda je mohu z počítače odebrat nebo jen zaměnit. A dále maximální množství daného typu komponenty.
- Vložit zboží do košíku a vytvořit objednávku.
- Umožnit manuální kontrolu a potvrzení objednávky provozovatelem B2C eshopu.
- Dodržet obecné zásady bezpečnosti při obchodování.
- Důležitými požadavky z marketingového pohledu jsou snadná ovladatelnost, přehlednost zobrazení a důraz na nové prvky designu webových stránek.

6.2 Analytický návrh řešení

Analýzou se zde rozumí popis základního chování systému s využitím metodiky UP a modelovacího jazyka UML. Výsledkem analýzy je analytický model systému, který zachycuje opisovaný problém z určité perspektivy. Obsahuje artefakty modelující problémovou doménu. Model se zaměřuje na to, co systém musí udělat, avšak nezabývá se detaily týkající se způsobu, jakým to udělá. Tyto detaily jsou rozebrány až v konkrétním návrhu.

6.2.1 Diagramy případů užití (UseCase)

Popisují činnosti, které mohou uživatelé s programem vykonávat. Modelování případů užití je způsobem získávání a dokumentování požadavků. Modelování spočívá v nalezení účastníků a případů užití.



Obr. 16. UseCase zákazník

Než zákazník může pomocí konfigurátoru vytvořit objednávku, musí se do systému přihlásit, nový zákazník vyplní registrační formulář a zadá heslo. Při dalším vstupu do systému se přihlásí pomocí identifikace a hesla, má možnost editovat údaje o sobě a změnit heslo. Z důvodů zvýšení bezpečnosti je tvar hesla omezen, aby se zamezilo použití lehce zjistitelných řetězců.

Při zobrazení sestavy může získat údaje o jednotlivých součástech, které do ní byly navrženy i o dílech, které zobrazí v seznamech nabízených variant. Komponenty pak může zaměnit, odebrat nebo naopak přidat. Po každé změně má k dispozici aktuální cenu sestavy. V systému jsou zabudována omezení pro maximální i minimální počty

jednotlivých typů komponent, např. počítačová sestava musí obsahovat alespoň jeden procesor, ale maximálně jsou to dva. Tento interval se načítá ze zdrojových dat, kde je určován podle vlastností základní desky PC.

Upravenou sestavu pak může vložit do košíku, ze kterého dál vytvoří objednávku. Z volí způsob dopravy, zkontroluje dodací a fakturační adresu, která se doplní z jeho profilu a odešle objednávku.

V seznamu objednávek může zkontrolovat jejich stav.

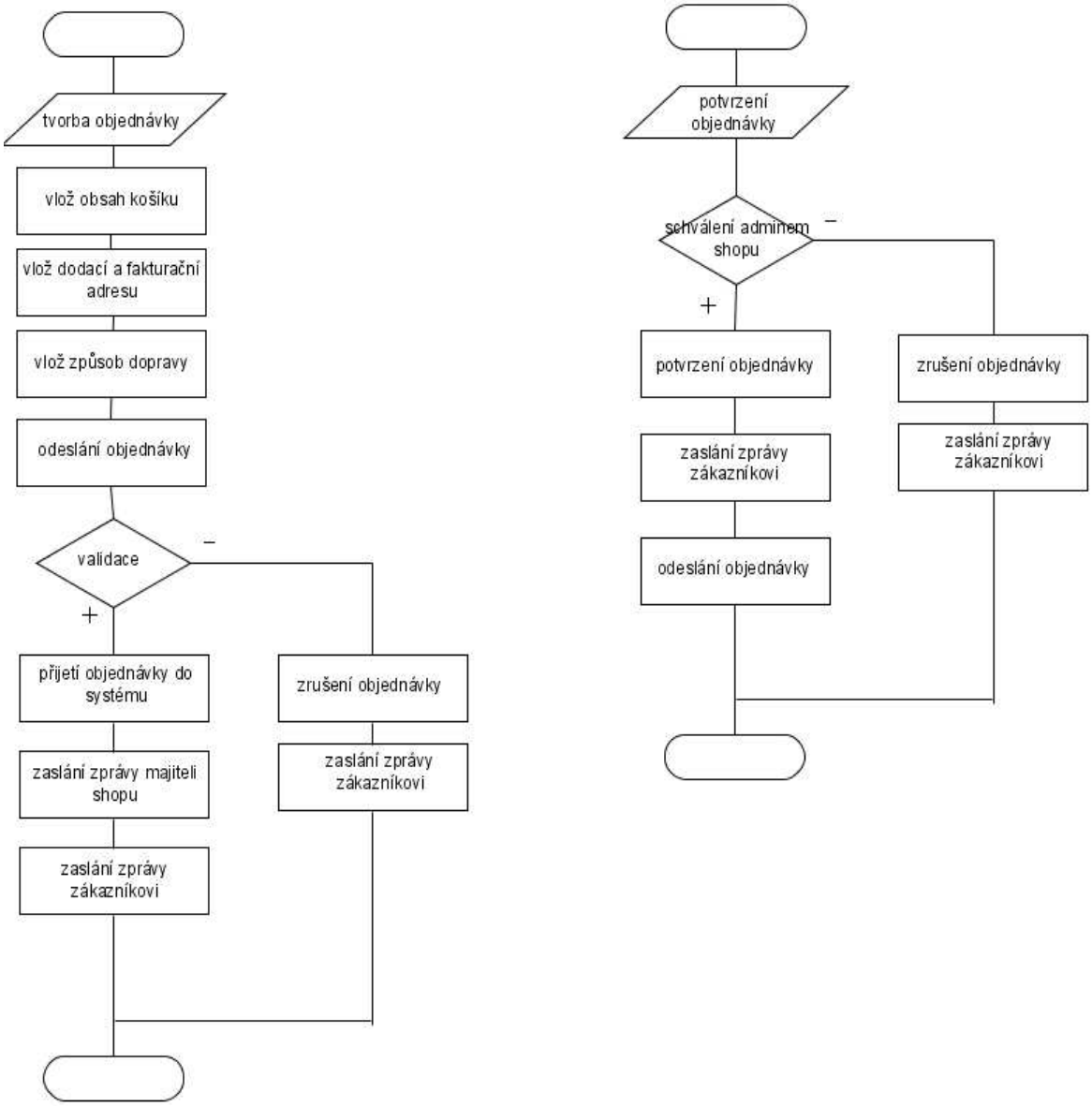
Obr. 17. UseCase dealer

Každá objednávka vložená do systému musí být schválena provozovatelem Eshopu v roli dealer. O každé nové objednávce je informován emailem, v té chvíli už je automaticky zkontrolována správnost cen a dalších údajů.

Dále je mu k dispozici přehled všech objednávek na jeho shopu, které může prohlížet, třídit.

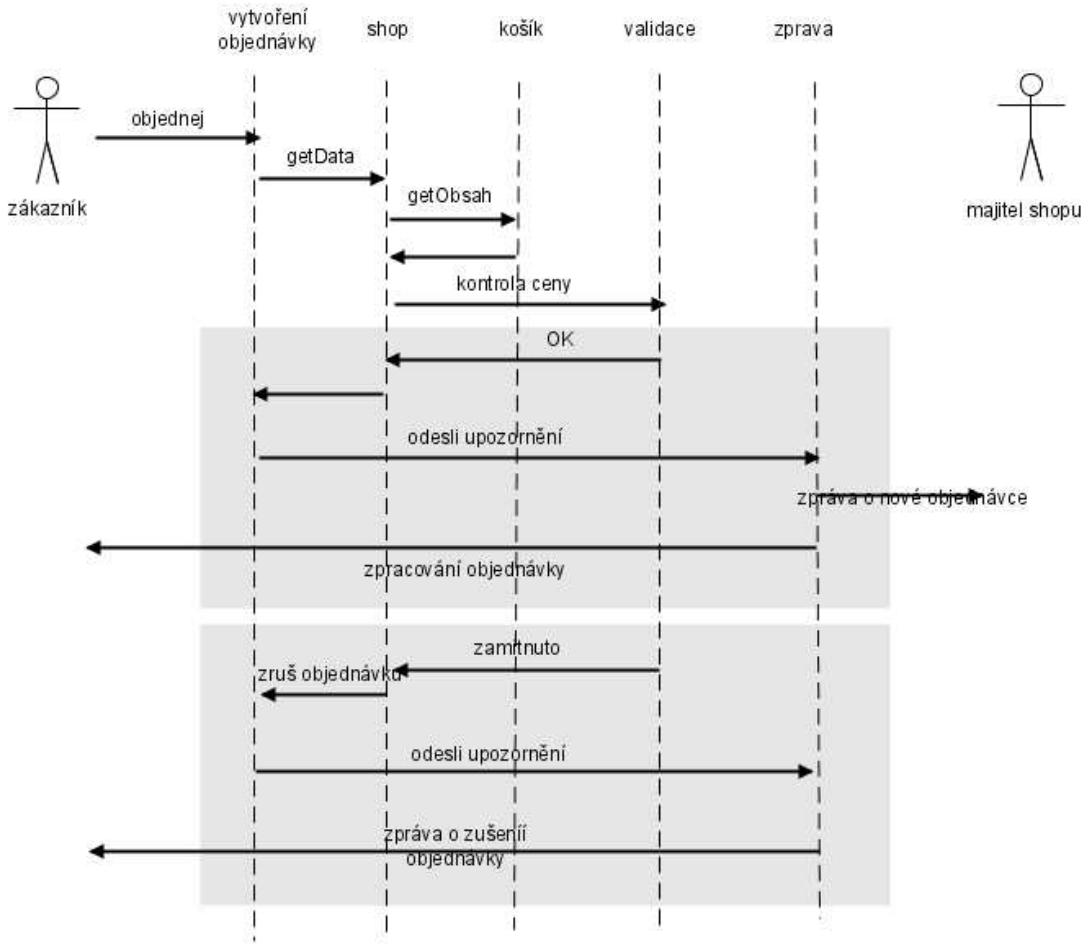
6.2.2 Servisně orientovaná analýza

Servisně orientovaná analýza určuje jak reprezentovat požadavky na automatizaci. V tomto projektu jsem ji použil při návrhu realizace objednávky zboží. Prvním krokemje dekompozice řídícího procesu, následující obrázek představuje diagram kroků procesu, již po odstranění dějů, které nelze automatizovat. Provádí je zákazník, jedná se o výběr zboží, způsobu dopravy atd. ne všechny kroky procesu se nakonec stanou operacemi služeb.



Obr. 18. Proces vytvoření a schválení objednávky

Při zachování principů znuvupoužitelnosti, autonomie, bezstavovosti a zjistitelnosti vytvoříme kandidáty služeb a ty můžeme dále seskupovat do kompozic. U této úlohově zaměřené služby dále vytvoříme logiku pracovního toku pro koordinaci služeb v kompozici. V našem případě tu jsou dvě možné varianty ukončení, validace proběhla úspěšně či ne. Ke znázornění použijeme sekvenční diagram. Pro jednoduchost jsem obě varianty zakreslil do jednoho grafu.

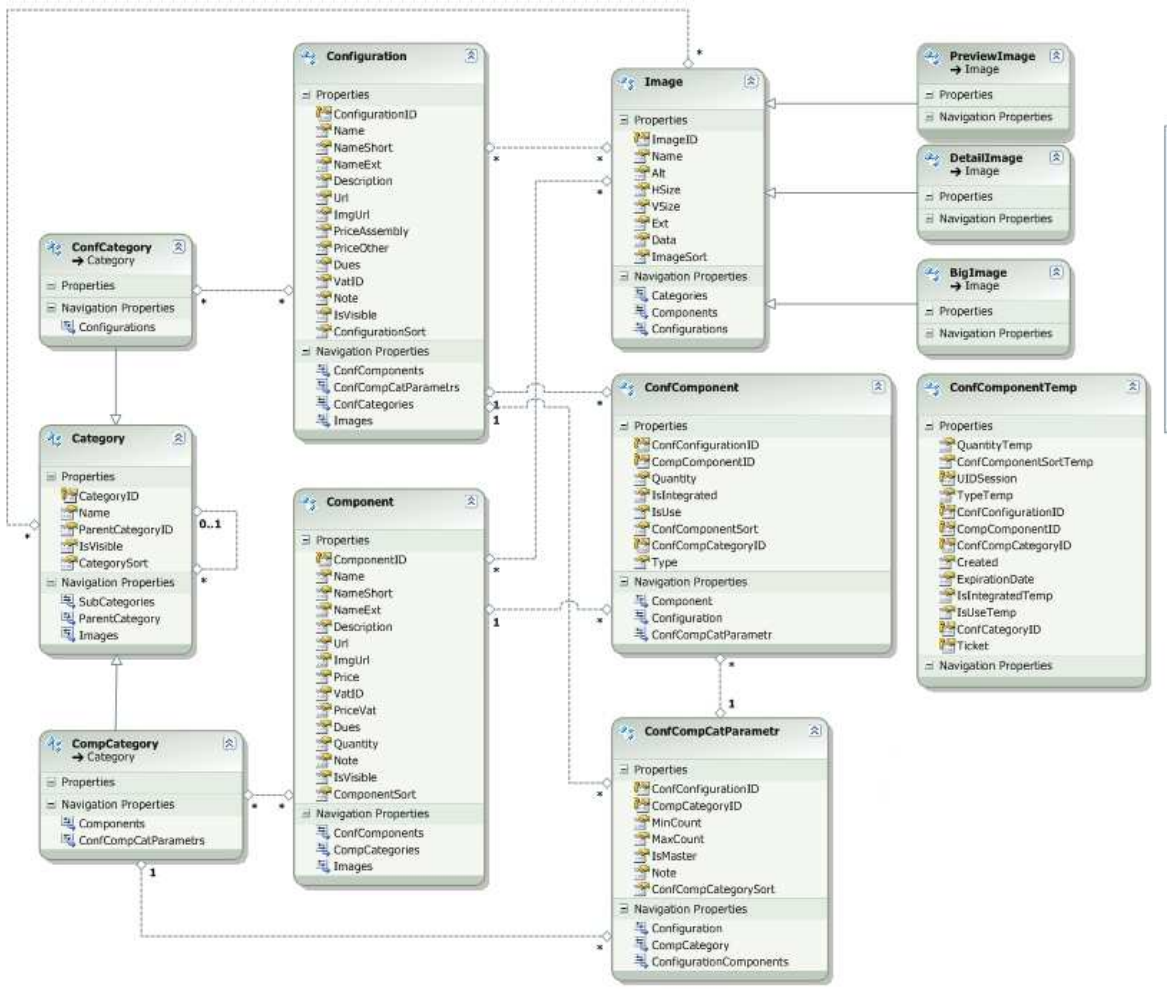


Obr. 19. Sekvenční diagram vytvoření objednávky

Stejné služby jsou znovupoužitelné i při zpracování potvrzení objednávky. Aby byla objednávka odeslána do systému B2B eshopu, kde bude vyřízena, musí ji ještě potvrdit administrátor aplikace.

6.2.3 Datový model entit

Pro vytvoření datového modelu jsem použil EDM integrovaný ve Visual Studiu 2010, který umožňuje generovat entity z existující databáze.

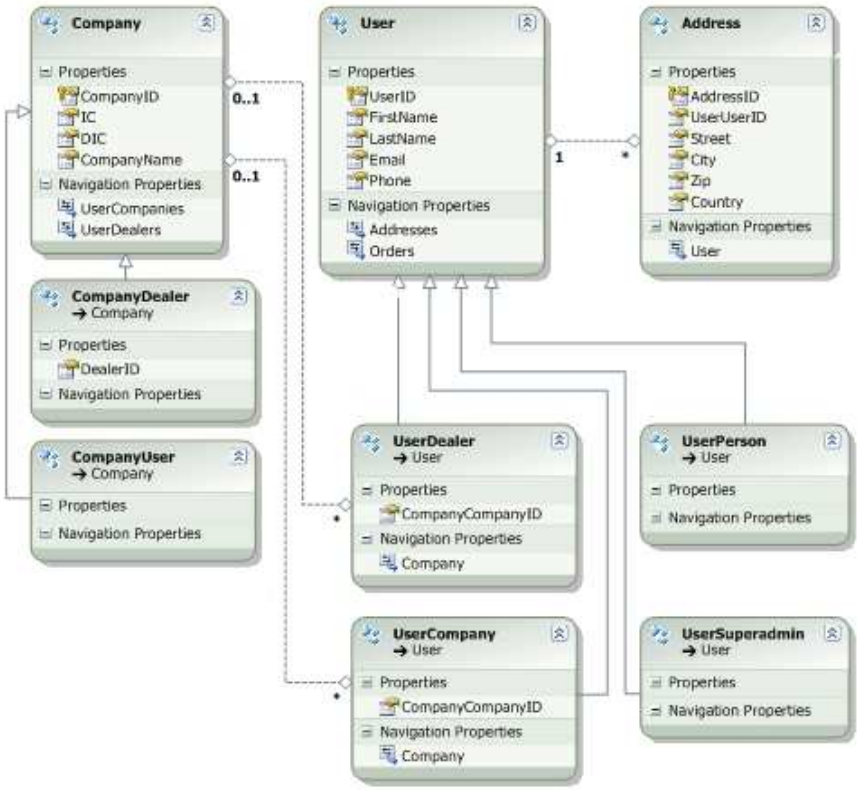


Obr. 20 Entitní datový model pro konfigurace

Základem aplikace je možnost upravovat nabízené sestavy, které jsou reprezentované entitou *Configuration*. Mezi její vlastnosti patří název, popis, cena, ale také informace o tom, zda sestavu zobrazit. Každá sestava patří do jedné nebo více kategorií (entita *Category*), které jsou uspořádány do stromové struktury podkategorií. V modelu je to implementováno pomocí vazby entity *Category* sama na sebe, jedná se vlastně o dvě vazby, na kterých můžu ukázat pluralizaci při tvorbě názvů, *ParentCategory* je typu 0..1 a *SubCategories* typu 0..n.

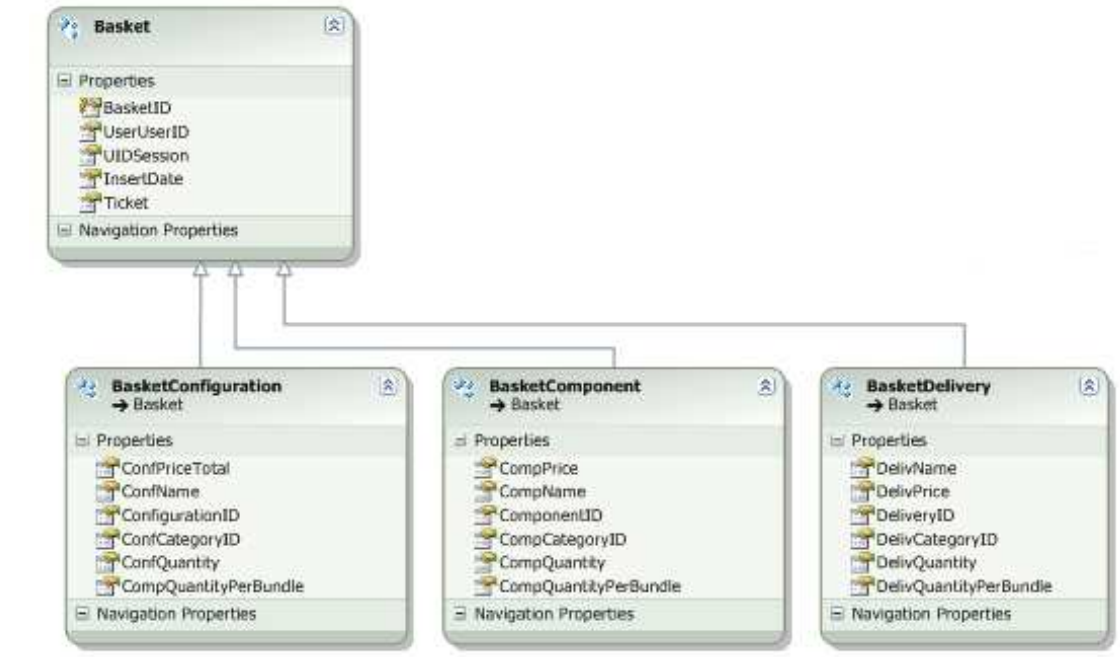
Entita *Category* je také použita ke třídění jednotlivých součástí sestav, které představuje entita *Component*. Vazbu mezi komponentou a sestavou popisuje entita ConfComponent, která udává množství, zda se jedná o integrovanou komponentu, výchozí předkonfigurovanou součást sestavy nebo je vybrána uživatelem. Entita

ConfCompCategoryParametr udává, ze kterých částí se daná sestava skládá, kolik komponent dané kategorie může sestava obsahovat. Například jedná-li se o počítač a kategorii pamětí bude MinCount 1 (počítač musí obsahovat paměťový modul) a MaxCount bude dán vlastnostmi základní desky. Pokud je MinCount roven nule, lze odebrat i výchozí prvek dané kategorie.



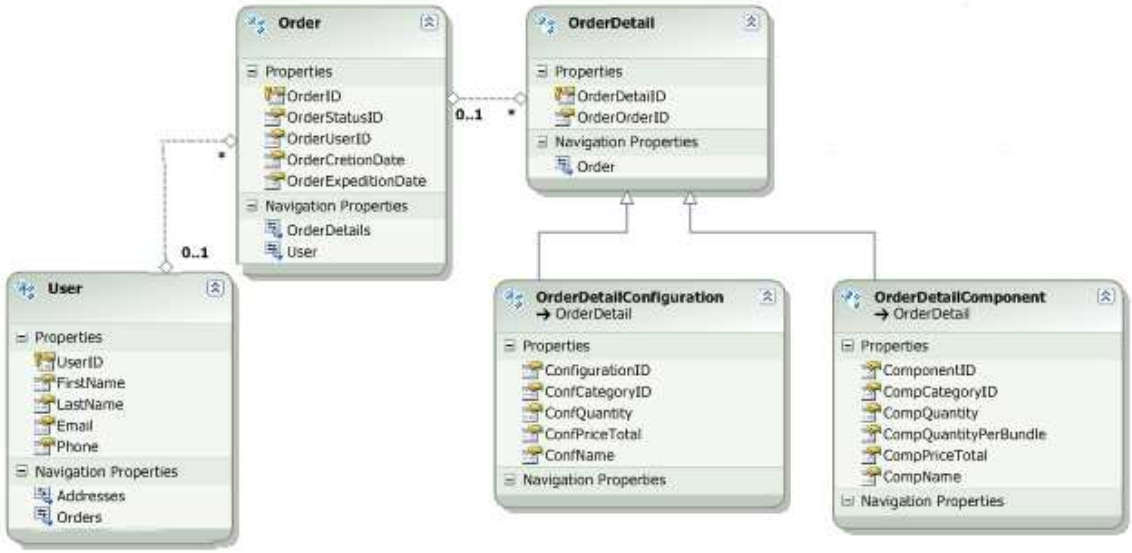
Obr. 21. Entitní datový model pro uživatele

Pro vytvoření různých typů uživatelů jsem využil dědičnosti, abstraktní entita *User*, obsahuje ID, jméno, email a je ve vztahu 1..n s entitou *Address*. Mimo klasického uživatele – zákazníka obchodu existují další typy uživatelů dealer a firemní uživatel, entity jsou napojeny na *Company*, posledním typem je admin.



Obr. 22. Entitní datový model pro košík

Košík je opět realizován pomocí abstraktní entity *Basket*, ze které odvozené entity dědí ID uživatele, session, čas vytvoření a tzv. ticket (GUID, které slouží k jednoznačné identifikaci *BasketConfiguration* (zde se nachází např. celková cena sestavy), *BasketComponent* (název, množství, cena) a *BasketDelivery*.



Obr. 23. Entitní datový model pro objednávku

6.3 SWOT analýza

Výhoda tohoto řešení spočívá v nenáročnosti aplikace. Nabízí se řešení např. na notebooku dealera přímo na prodejně. Nutné je ovšem kvalitnější internetové připojení, neboť jej Ajaxové aplikace obecně potřebují.

Konfigurátor je konzument služeb, pro které je nutné generovat konfigurace v externí aplikaci dodavatele produktů. Pokud bude i nadále přetrvávat klesající trend v prodeji počítačů, je možné, že opadne zájem o toto řešení. Ovšem i přes uvedené riziko, lze aplikaci dále provozovat se sortimentem spotřební elektroniky, např. komplety pro příjem satelitního vysílání, domácí audio video zařízení.

7 NÁVRH

V analytické části byly sepsány požadavky na program, byly vytvořeny analytické modely obecného řešení. Tato část se již zabývá konkrétním řešením. Přitom se respektují výsledky, kterých bylo dosaženo v analýze. Návrh tedy rozpracovává analytické modely, přidává do nich detaily implementace a specifická technická řešení.

7.1 Výběr platformy

Jedná se o webovou aplikaci postavenou na architektuře dynamických stránek Microsoft ASP.NET, využívajících pro programování knihovnu Microsoft .NET Framework. Tato moderní technologie zaručuje dostatečnou výkonnost aplikace, zabezpečení, použití standardních webových prvků. Existuje také podpora vývojářů, technologie je neustále vyvíjena a jsou vydávány opravy známých chyb. Pro vývoj jsem zvolil nejnovější verzi Visual Studio 2010 Beta1, Beta2, RC1, .NET Framework 4.0 Beta1, Beta2, RC1.

Webová aplikace bude nainstalovaná na webovém serveru Microsoft IIS7 (Internet Information Services) nebo 7.5. IIS7 je součástí operačního systému Windows Vista a Windows 7 nebo je využít MS2008 Server, případně MS2008 Server R2, který už obsahuje IIS7.5. Databáze jsou vytvořeny na MS SQL Server 2008 Express firmy Microsoft. Výhodou je především její provázanost s technologií .NET. Pro přístup k datům využívám technologii LINQ, která podporuje oddělení nezávislost na fyzickém zdroji dat. Datová vrstva je vymodelována EDM ADO.NET Entity Frameworku.

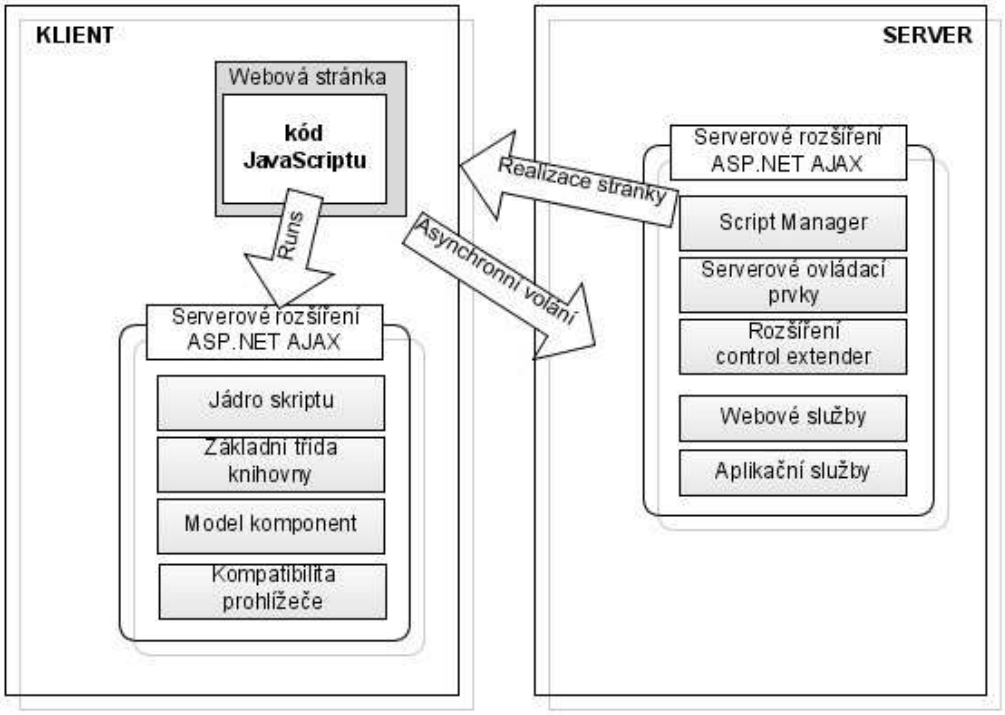
7.2 Programovací jazyk

Programovacím jazykem aplikace je Microsoft C# 4.0. Dotazech databáze jsou napsány v jazyce SQL, uložené procedury v Microsoft Transact-SQL, který rozšiřuje možnosti SQL. Vhodného rozložení a vzhledu webových prvků ASP.NET stránek bylo dosaženo použitím CSS kaskádových stylů.

7.3 ASP.NET AJAX

Jedná se o sadu nástrojů, které programátorovi nabízí pomoc při vytváření ajaxových stránek. Skládá se ze dvou částí:

- na straně klienta – sada knihoven JavaScriptu, které nejsou spojeny ASP.NET
- na straně serveru – obsahují ovládací prvky a komponenty, které používají klientské knihovny JavaScriptu.



Obr. 24. Architektura ASP.NET Ajax

Serverové ovládací prvky umožňují práci s kódem výhradně na straně serveru, protože vygenerují potřebný skript. Výhodou tohoto řešení je vyšší výkon, nevýhodou naopak nižší flexibilita. Ve svém projektu konkrétně používám UpdatePanel, který umožní obnovit části serverové stránky ve stylu Ajaxu. Obnovuje obsah své <ContentTemplate>, obnova je realizována pomocí triggerů (eventů ostatních prvků) nebo obsahu CT. Obvykle se používá ve spojitosti s AutoPostBack=true. Když v prvku UpdatePanel nastane nějaká událost, která by spustila odeslání zpět na server, UpdatePanel tuto událost zachytí a provede asynchronní volání. Vše probíhá v následujících krocích:

- Uživatel klikne na tlačítko v prvku UpdatePanel
- Javascriptový kód na straně klienta, který byl vygenerován komponentou ASP.NET AJAX, zachytí událost a provede zpětné volání serveru.

- Na serveru se provede normální životní cyklus stránky s obvyklými událostmi, nakonec je stránka realizována do HTML a vrácena prohlížeči.
- Kód JavaScriptu na straně klienta obdrží odpovídající HTML a aktualizuje každý prvek UpdatePanel – nahrazením jeho aktuálního kódu za nový obsah. Pokud nastane změna v obsahu mimo UpdatePanel ignoruje ji.

I když ASP.NET nezahrnuje žádná rozšíření ovládacích prvků, existuje tato možnost díky ASP.NET AJAX Control Toolkit. Jedná se o kolekci ovládacích prvků a rozšíření, která využívá funkcionality ASP.NET AJAX. Prvky mohou být umístěny na webové stránky stejným způsobem jako klasické serverové ovládací prvky. Tato sada nástrojů je vyvíjena pod Open source licencí ve spolupráci s MS. V následující tabulce je uveden stručný popis některých dostupných rozšíření.

Tab. 2. Ovládací prvky v ASP.NET AJAX Control Toolkitu

Název	Popis
Accordion	Tento ovládací prvek naskládá na sebe několik obsahových panelů a umožní zobrazit je jeden po druhém. Po kliku se panel rozbalí, ostatní automaticky sbalí až na záhlaví. Zahrnuje efekt zesvětlování a omezení velikosti prvku.
CollapsiblePanelExtender	Toto rozšíření umožňuje sbalit a rozbalit panely na stránce, zbytek obsahu bude automaticky obtékán.
DynamicPopulateExtender	Nahradí obsah ovládacího prvku za výsledek volaní nějaké webové služby
PopupControlExtender	Poskytuje vysunovací obsah, který lze obrazit vedle jakéhokoliv ovládacího prvku.

7.3.1 Instalace ASP.NET AJAX Control Toolkitu

ASP.NET AJAX Control Toolkitu je dispozici ke stažení zdarma na oficiálních stránkách platformy http://ajax.asp.net/ajaxtoolkit nebo na stránkách http://www.codeplex.com. Nachází se zde různé balíky ke stažení, podle používané verze a zda požadujete zdrojový kód. ZIP soubor obsahuje adresář SampleWebSite se vzorovým webem, uvnitř podadresáře Bin se nachází klíčové soubory, včetně centrální assembly Ajax-ControlToolkit.dll a

dalších assembly podporující jiné lokalizace. Pro další využití je nejvýhodnější přidat nové komponenty do Toolboxu Visual Studia. Postup:

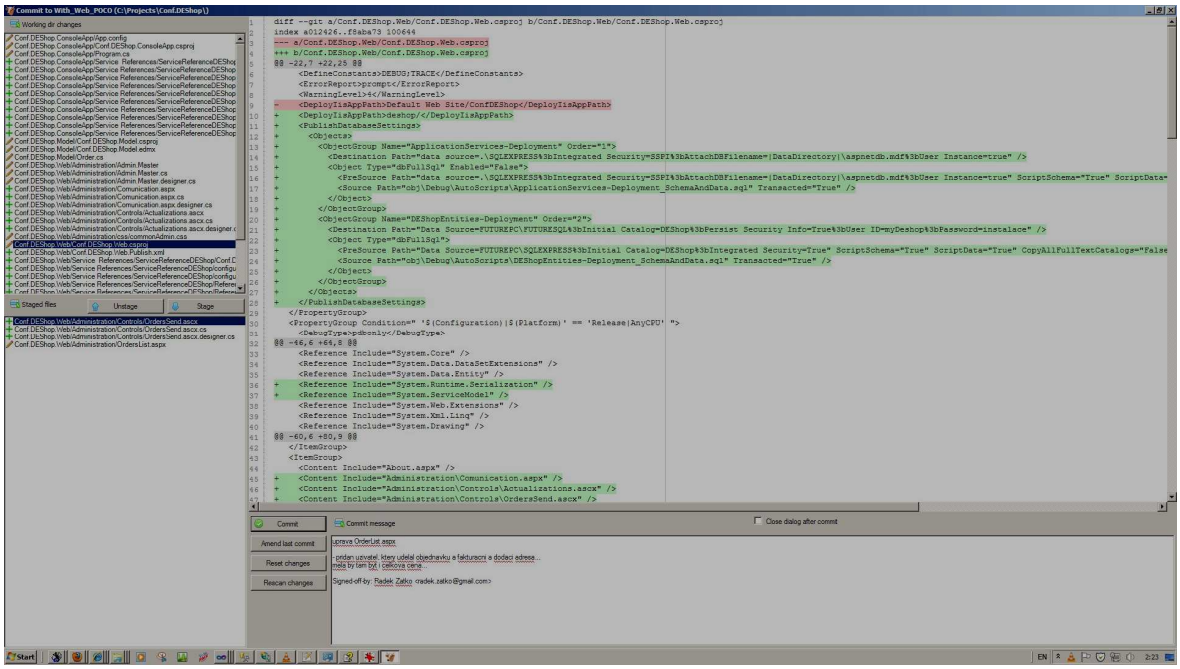
Složku umístit do trvalého umístění, po dodatečném přesunu, by VS nenalezlo Ajax-ControlToolkit.dll.

Vytvořit novou složku Toolboxu, Add Tab.

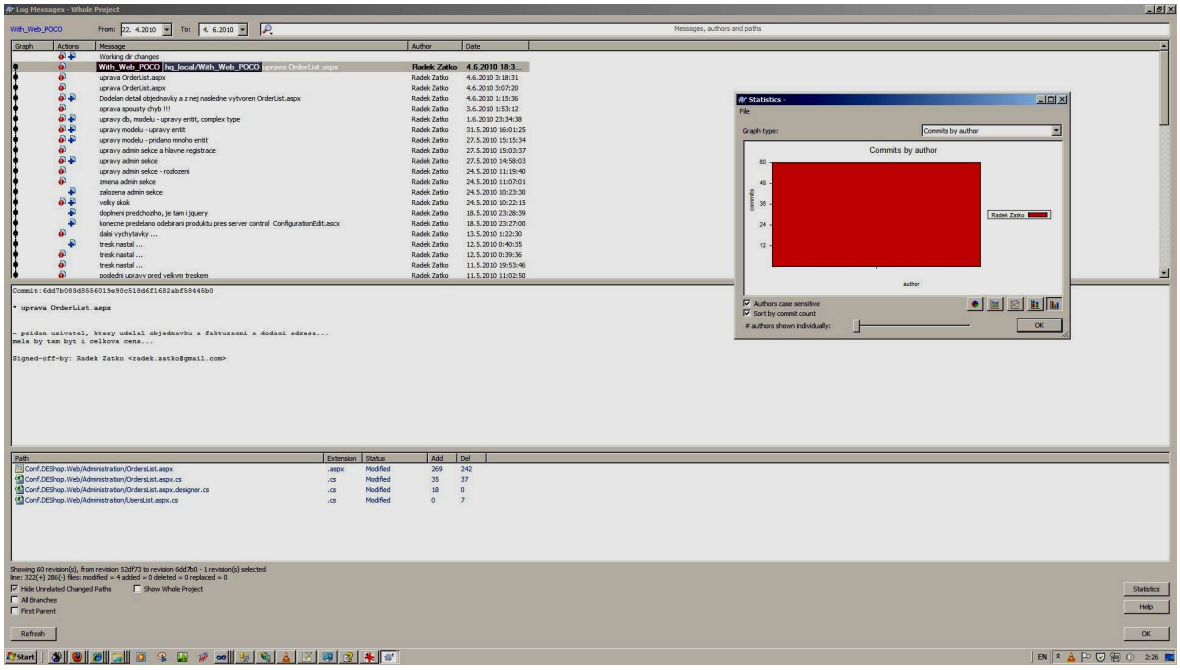
Přidat ovládací prvky na tuto novou kartu, Choose Item a vybrat umístění Ajax-ControlToolkit.dll. Nyní se v seznamu komponent, je možné vybrat a vložit jen část z nich nebo všechny.

7.4 GIT

Dalším pluginem pro Visual Studio 2010, který jsem při vývoji použil, je GIT. Jedná se o systém pro zprávu verzí. GIT je decentralizovaný systém, který dovoluje mít několik lokálních branches zdrojového kódu, které na sobě mohou být zcela nezávislé. Vytváření, spojování a mazání těchto „větví“, resp. „verzí“.



Obr. 25. Porovnání verzí v GIT



Obr. 26. Logování v GIT

7.5 Testování

Během vývoje aplikace byla každá nová funkce otestována a ověřeny její reakce na špatné, nebo nečekané vstupy. Nebyly provedeny zátěžové testy, ověřující výkonnost aplikace při přístupu více uživatelů současně. Pro relevantní výsledky zátěžových testů je nutné použít stejné, nebo alespoň podobné HW prostředky. Volba HW konfigurace, ani parametry webového a databázového serveru nejsou doposud známi.

Činnosti provedené v rámci testování:

- testy uživatelského rozhraní, reakce na chybné vstupy a nečekané ukončení prohlížeče
- prověření přihlašování

8 UŽIVATELSKÁ PŘÍRUČKA

Obr. 27. Úvodní strana aplikace

8.1 Úvod

Uživatelská příručka k aplikaci Konfigurátoru počítačových sestav DEShop popisuje instalaci, poinstalační nastavení, uživatelské použití a správu.

DEShop umožňuje :

- konfiguraci počítačových sestav
- zobrazit informace o jednotlivých produktech
- vytvoření objednávky
- správu objednávek
- použití objednávkového systému pro dealera

Podpora: dotazy a připomínky zasílejte na zatko.radek@gmail.com.

8.2 Instalace

Požadavky:

- Windows 7
- webový server Microsoft Internet Information Services IIS7.0 -7.5
- ASP.NET Framework 4.0
- Entity Framework
- podpora WCF
- Microsoft Sync Framework
- SQL Server 2008 SQLExpress

Před samotnou instalací je nutné u SQL Server 2008 nastavit způsob přihlašování na SQL Server Authentification, přihlašování pomocí uživatele a hesla. Dále vytvořit databázi s názvem deshop, vytvořit uživatele s právy zápisu.

Obr. 28. Import aplikace DEShop

V IIS manageru v hlavním webovém adresáři založíme novou aplikaci, jako alias zvolíme deshop a fyzickou cestu zvolíme např. inetpub/wwwroot, vytvoříme nový adresář deshop. Vytvoříme nový aplikační pool deshoppool a vybereme verzi frameworku 4.0. Tento pool

vybereme pro naši aplikaci. Pro adresář aplikace zvolíme deploy a přidat, vybereme instalační balík přiložený na CD s názvem conf.deshop.web.zip.

Pro testovací účely jsou v databázi pro každý typ uživatele vytvořeny účty:

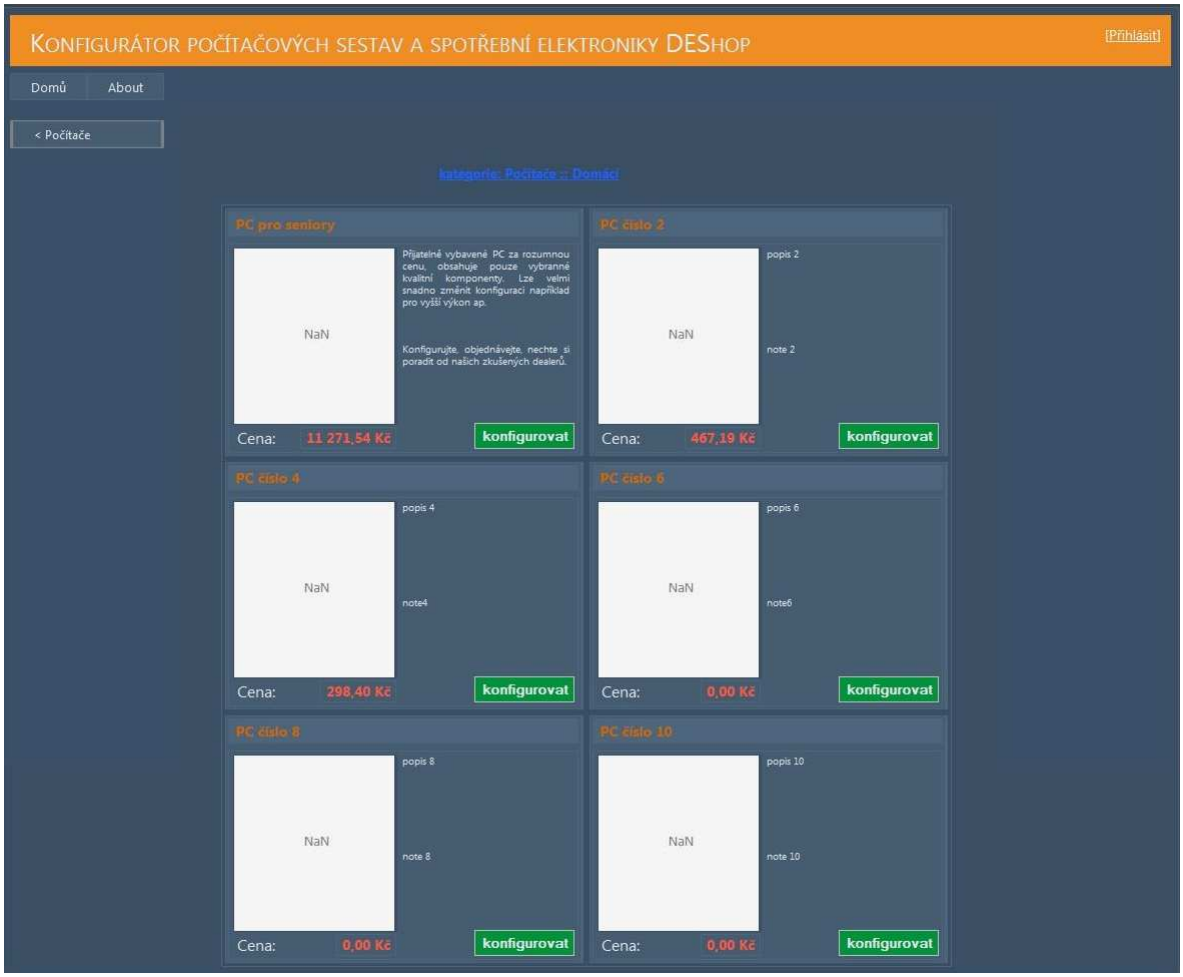
Tab. 3. Přehled testovacích účtů

User name	Password
zakaznik	zakaznik
prodejce	prodejce
Podnikatel	podnikatel

Po té se spustí instalace webové aplikace do tohoto adresáře a zároveň je třeba vyplnit v průvodci connection string databáze, uživatelem. Naimportuje se kompletní databáze i s daty. Pokud je nutné upravit práva k adresářům, postupujte podle msdn.

8.3 Používání

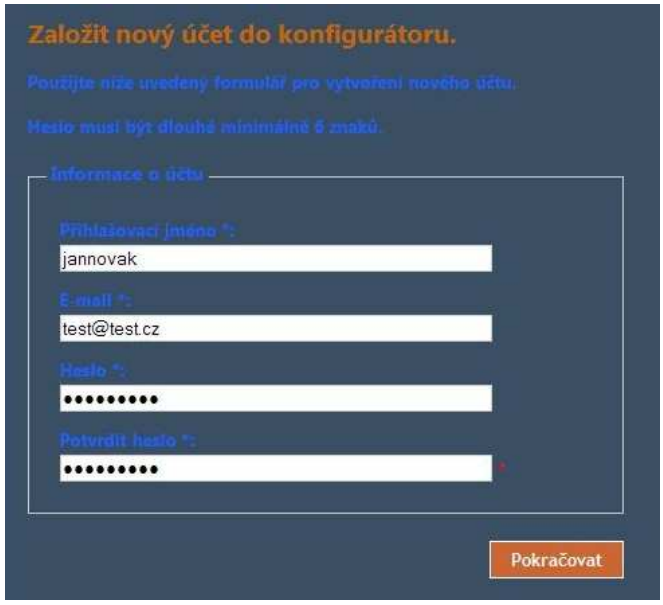
Po příchodu na stránky DEShopu je možné procházet jednotlivé podkategorie zboží a seznamy produktů. K dispozici jsou základní informace jako cena, vyobrazení a kratký popis. Ke změnám v konfiguracích je nutné být přihlášen.



Obr. 29. Seznam produktů v DEShopu

8.3.1 Přihlášení

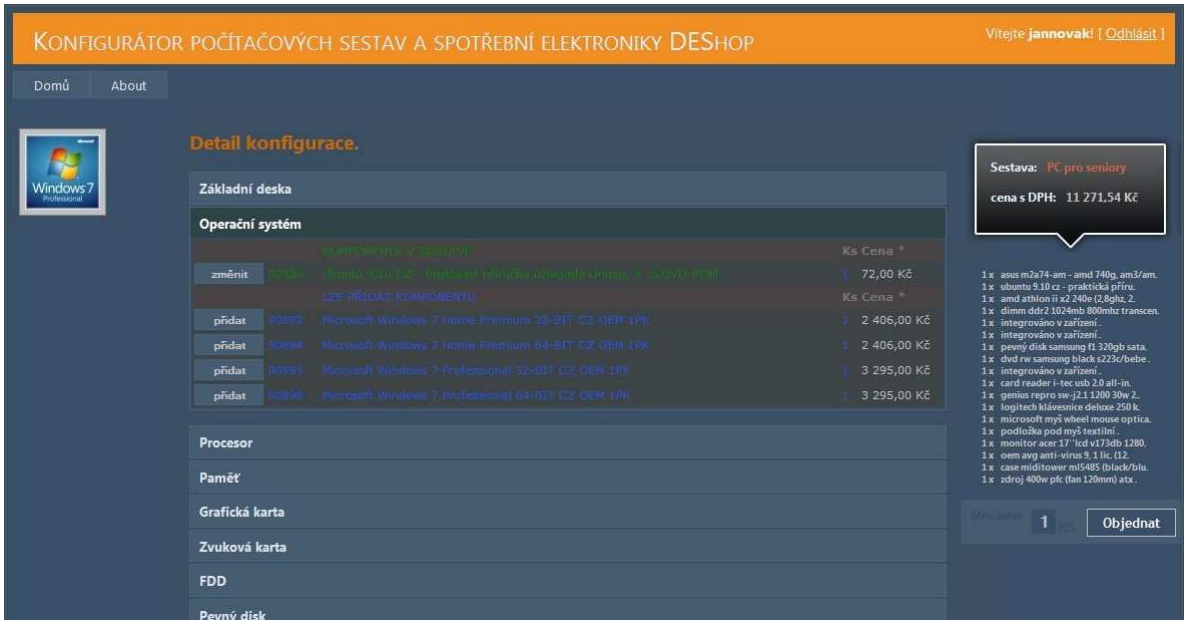
Přihlášení lze provést pomocí odkazu nebo jste k němu vyzváni automaticky při volbě *konfigurovat* u vybrané sestavy. Zobrazí se standardní formulář pro zadání uživatelského jména a hesla nebo nabídka nové registrace, která probíhá ve dvou krocích.



Obr. 30. Registrace uživatele

U nové registrace je možné specifikovat, zda se jedná o firemního zákazníka nebo soukromou osobu. Po vyplnění požadovaných údajů je vytvořen nový uživatel a jste přihlášen do systému.

8.3.2 Konfigurace a vytvoření objednávky

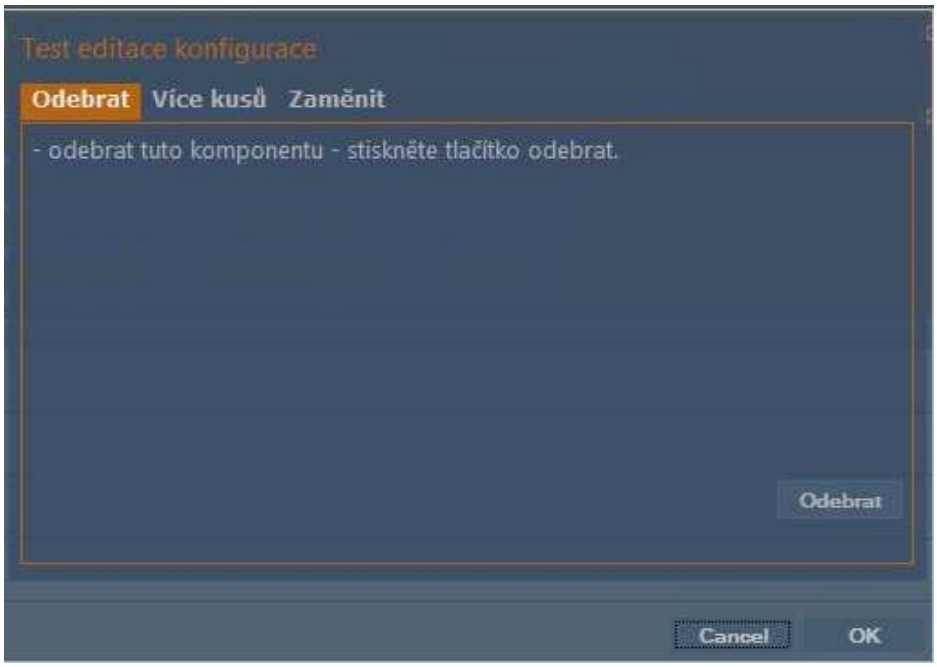


Obr. 31. Výpis konfigurované sestavy

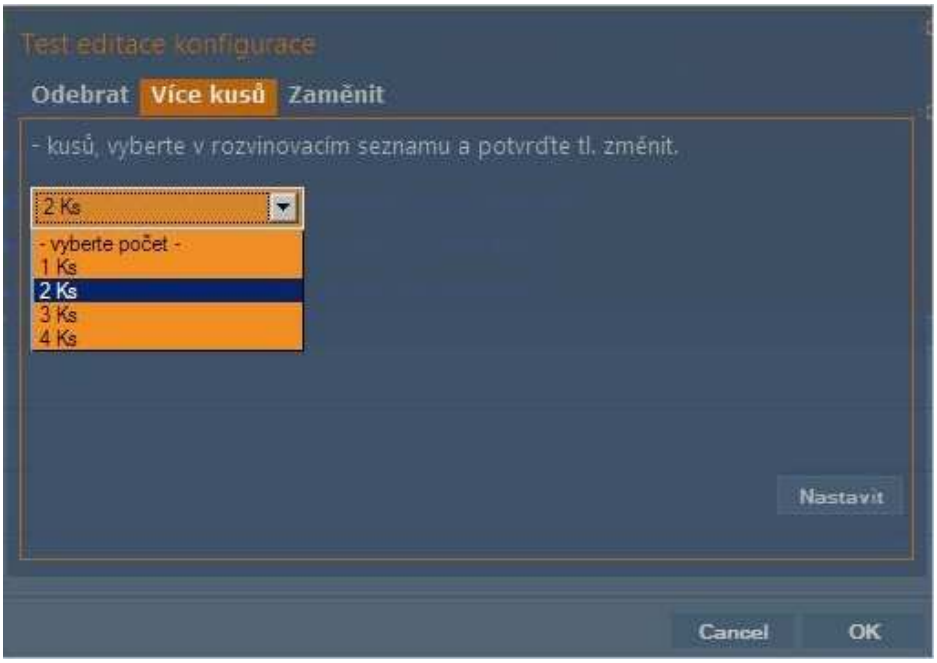
Po volbě konfigurovat se zobrazí detail konfigurace, v pravé části seznam aktuálně zvolených komponent a základ stránky tvoří seznam kategorii a po rozkliku se zobrazí seznam volitelných položek. V levé části obrazovky jsou zobrazeny údaje o jednotlivých součástech sestav, mění se při přejetí myši k dané komponentě. Výchozí je popis komponenty, která je už součástí sestavy, v seznamu je na prvním místě a je barevně zvýrazněna. U této komponenty je volba *změnit* a u ostatních *přidat*. Obrázky () ukazují jednotlivé možnosti změny:

- *odebrat* – je umožněno pouze v případě, že sestava nemusí obsahovat žádný prvek z této kategorie (např. nelze odebrat procesor, pokud je v počítačové sestavě jediný)
- *více kusů* – zvolíme počet kusů, které přidáme
- *zaměnit* – aktuální komponentu zamění za vybranou

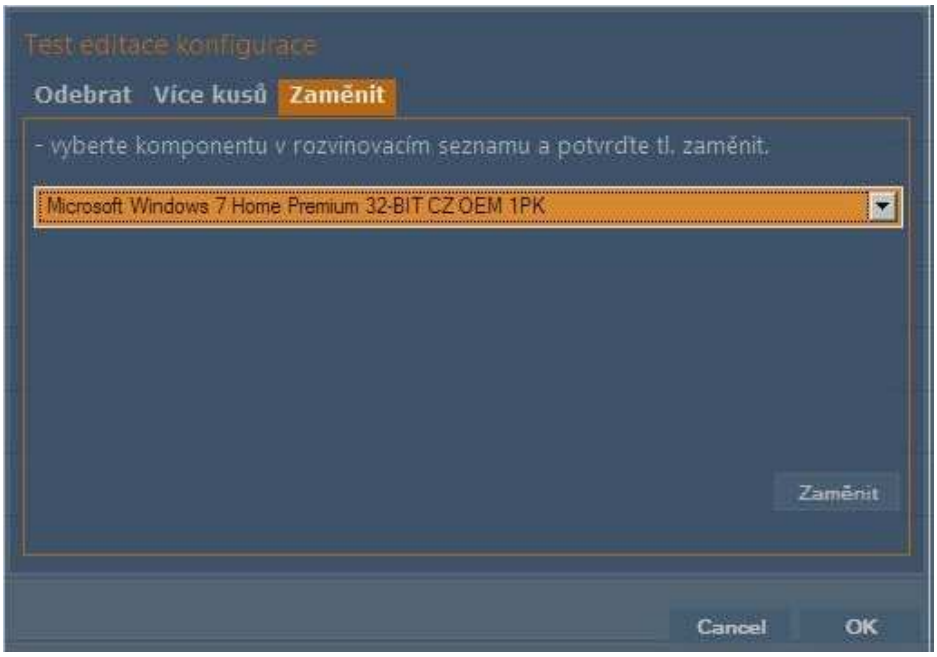
Po každé úpravě můžete vpravo sledovat změnu ceny.



Obr. 32. Odebrání komponenty

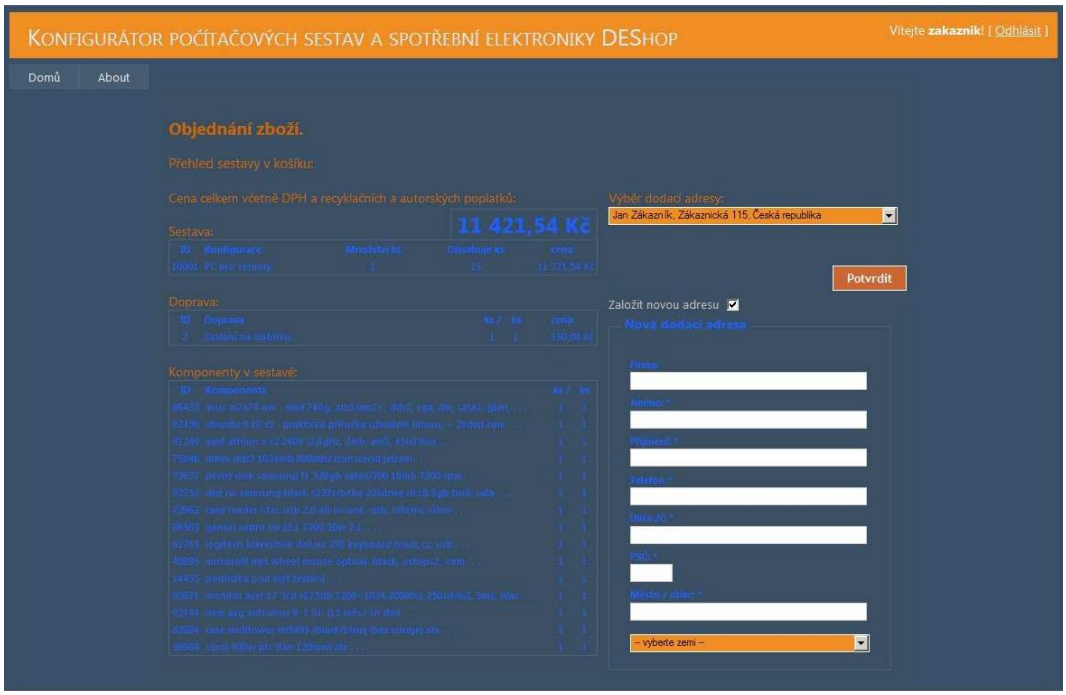


Obr. 33. Změna počtu kusů



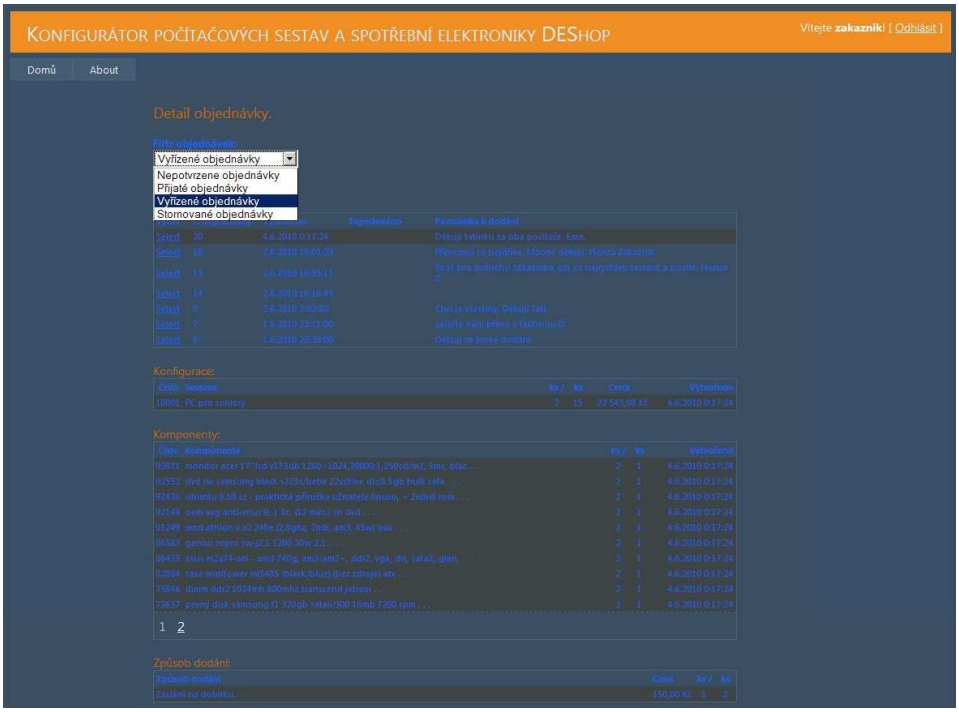
Obr. 34. Záměna komponent

Po skončení úprav sestavu, úpravě požadovaného množství stiskněte *objednat*. Je zboží vloženo do košíku a vy můžete pokračovat v tvorbě objednávky. Zvolíte způsob dopravy, v případě volby na dobírku, vyberete ze seznamu adresu dodání nebo můžete přidat další. Ta bude uložena pro další použití.



Obr. 35. Výběr způsobu dopravy

Po potvrzení tlačítkem *potvrdit* je objednávka přijata do systému a vy můžete její stav sledovat v souhrnném výpisu Vašich objednávek.



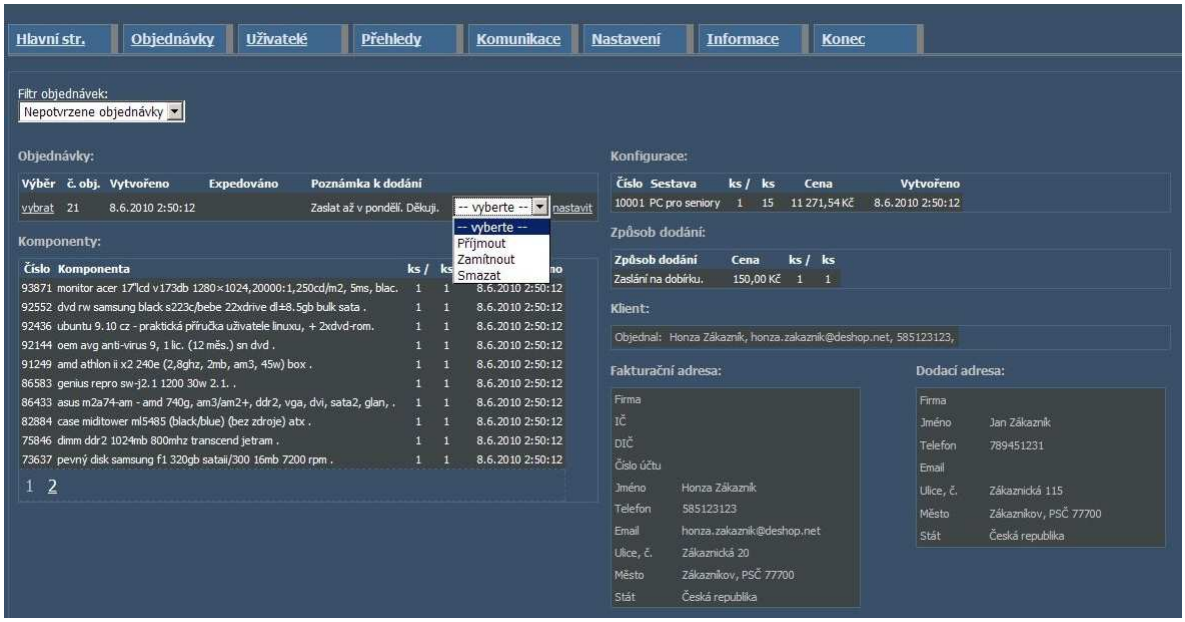
Obr. 36. Přehled objednávek

8.3.3 Prodejce DEShopu

Kromě možností objednávkového systému popsaného v předchozí kapitole, obsahuje rozhraní pro správu obchodu.

Obr. 37. Rozhraní uživatele typu dealer

Každá objednávka vytvořená uživatelem je dealerem schválena a odeslána vyřízení.



Obr. 38. Správa objednávek DEShopu

Další možnosti, které bude aplikace nabízet, je správa uživatelů a synchronizace nejen objednávek, ale i vstupních dat pro konfiguraci. Ta nebude spouštěna automaticky, ale ve volbě *aktualizace databáze webu*.

ZÁVĚR

V diplomové práci jsem navrhl a realizoval konfigurátor výpočetní techniky a spotřební elektroniky s možností využití jako B2C e-shop. Samotnému návrhu a tvorbě aplikace předcházela teoretická příprava.

Seznámil jsem se a analyzoval současnou situaci využití a implementace konfigurátorů na webových stránkách e-shopů českých prodejců. Výsledky jsem shrnul v teoretické části práce.

Nejvýznamnější částí práce je návrh a realizace této aplikace za použití nových technologií.

Při tvorbě jsem využil Entity Framework, Linq, Ajax, WCF. Z pohledu zákazníka se jedná o běžný eshop, který je však doplněný o synchronizaci dat s centrálním serverem dodavatele výpočetní techniky, poskytovatele dat.

Uživatelská příručka popisuje instalaci webové aplikace na server internetové informační služby včetně instalace databáze se vstupními daty. Popis prostředí webové aplikace od registrace po objednání a následné zpracování dealerem.

CONCLUSIONS

In the master thesis I have designed and implemented configurator computer technology and consumer electronics, with the possibility of use as B2C e-shop. There was a theoretical preparation before making the design and the implementation.I acquaint myself with the current situation and implement the use configurator on the website of the Czech e-shop vendors. I analyzed this and summarized the results in the theoretical part of this thesis.

I have used in creating Entity Framework, LINQ, Ajax, WCF. From a customer perspective it is a common shop, which is supplemented by data synchronization with central server vendor computing, data providers.

User's Guide describes how to install, uninstall, setup IIS web server, database server MS SQL Server 2008, and a description of using configurator.

SEZNAM POUŽITÉ LITERATURY

[1] BRUST, Andrew J.; FORTE Stephen. *Mistrovství v programování SQL Serveru 2005*. Brno: Computer Press, 2007. 848 s. ISBN 978-80-251-1607-4.

[2] BUDD, Andy; MOLL, Cameron; COLLISON, Simon. *CSS- filtry, hacky a pokročilé postupy*. Brno: Zoner Press, 2007. 272s. ISBN:978-80-86815-54-1

[3] DUTHIE, G. Andrew. *Microsoft ASP.NET Krok za krokem*. Praha: MobilMedia, 2003. 511s. ISBN 80-86593-33-9.

[4] CASTRO, Elizabeth. *XML pro World Wide Web*. Brno: SoftPress, 2001. 255s. ISBN: 80-86497-07-0

[5] CONOLY, Thomas; BEGG, Carolyn; HOLOWCZAK, Richard. *Mistrovství - Databáze – Profesionální průvodce tvorbou efektivních databází*. Brno: Computer Press, 2009. 584s. ISBN: 978-80-251-2328-4.

[6] ERL, Thomas. *SOA Servisně orientovaná architektura – Kompletní průvodce*. Brno: Computer Press, 2009. 672 s. ISBN 978-80-251-1607-4.

[7] EVJEN, Bill; HANSELMANN, Scott; RADER, Devon. *ASP.NET 3.5 v jazycích C-SHARP a Visual Basic – Programujeme profesionálně*. Brno: Computer Press, 2009. 1600s. IBSN 978-80-251-2069-9.

[8] HERNANDEZ, Michael J.; VIESCAS, John L..*Myslíme v jazyku SQL tvorba dotazů*. Praha: Grada Publishing, 2004. 378s. ISBN 80-247-0899-X

[9] MACDONALD, Matthew; SZPUSZTA, Mario. *ASP.NET 2.0 a C-SHARP – Tvorba dynamických stránek profesionálně*. Brno: Zoner Press, 2006. 1376s. ISBN 80-86815-38-2.

[10] MACDONALD, Matthew; SZPUSZTA, Mario. *ASP.NET 3.5 a C-SHARP 2008 Tvorba dynamických stránek profesionálně*. Brno: Zoner Press, 2008. 1376s. ISBN 978-80-7413-008-3.

[11] NAGEL, Christian; EVJEN Bill; GLUNN Jay; WATSON, Karl; JONES, Allan. *C-SHARP 2005 Programujeme profesionálně*. Brno Computer Press, 2006. 1400s. ISBN 80-251-1181-4.

[12] PIALORSI, Paolo; RUSSO, Marco. *Microsoft LINQ – Kompletní průvodce programátora*. Brno: Computer Press, 2009. 616s. ISBN: 978-80-251-2735-3.

[13] SCHMULLER, Joseph. *Myslíme v jazyku UML*. Praha: Grada Publishing, 2001. 359s. ISBN 80-247-0029-8.

[14] WALTERS, Robert E.; COLES, Michael; RAE, Robert; FERRACCHIATI, Fabio; FARMER, Donald. *Mistrovství v Microsoft SQL Server 2008*. Brno: Computer Press, 2009. 864s. ISBN: 978-80-251-2329-4.

[15] MICROSOFT CORPORATION [online]. [cit. 2009-05-10]. MSDN online. Dostupné na <www: http://msdn.microsoft.com>

[16] *Web Services Description Language (WSDL) Version 1.1: Primer* [online]. [cit. 2009-05-12]. Dostupné z WWW: <http://www.w3.org/TR/wsdl>

[17] *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer* [online]. [cit. 2009-05-12]. Dostupné z WWW: <http://www.w3.org/TR/wsdl20-primer/>

[18] *XML Schema Part 2: Datatypes Second Edition* [online]. [cit. 2009-05-12]. Dostupné z WWW: <http://www.w3.org/TR/xmlschema-2/>

[19] *Využití webových služeb a protokolu SOAP při* komunikaci [online]. [cit. 2009-05-12]. Dostupné z WWW: <http://www.kosek.cz/diplomka/ >

[20] *Základy .NET XML Webových Služeb* [online]. [cit. 2009-05-12]. Dostupné z WWW: <http://herakles.zcu.cz/education/net/lectures/WSOld.pdf>

[21] *Návrhové vzory, příklad - design systému* [online]. [cit. 2009-05-12]. Dostupné z WWW:< http://objekty.vse.cz/Objekty/

[22] *AJAX Tutorials* [online]. [cit. 2010-03-5]. Dostupné z WWW: <http://www.asp.net/ajax/>

[23] *Announcing the release of Entity Framework 4* [online]. [cit. 2010-05-5]. Dostupné z WWW: <http://blogs.msdn.com>

[24] ČINČURA, J. *Microsoft Sync Framework – Úvod* [online]. 2007. [cit. 2010-05-5]. Dostupné z WWW: < http://www.vyvojar.cz/Articles/557-microsoft-sync-framework-uvod.aspx>

[25] HERCEG, T. *Používáme T4 šablony* [online] 2010. [cit. 2010-06-01]. Dostupné z WWW: < http://www.vbnet.cz/clanek--158-pouzivame_t4_sablony.aspx>

[26] AUGUSTÝN, M. *LINQ a lambda expressions* [online] 2009. [cit. 2010-06-01]. Dostupné z WWW: < http://zdrojak.root.cz/clanky/linq-a-lambda-expressions/>

[27] AUGUSTÝN, M. *LINQ a lambda expressions* [online] 2009. [cit. 2010-06-01]. Dostupné z WWW: < http://zdrojak.root.cz/clanky/linq-a-lambda-expressions/>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application programming interface
ASP	Active Server Pages.
B2B	business-to-business
B2C	business-to-consumer
C2B	consumer-to-business
C2C	consumer-to-consumer
CLR	Common Language Runtime
CSDL	Conceptual Schema Definition Language
CT	ContentTemplate
DBMS	Database management system
EDM	Entity Data Model designer
EDM	Entity Data Modeling
EF4	Entity Framework 4
HTML	HyperText Markup Language
IPOCO	Interface-Based Plain Old CLR Objects
J2EE	Java 2 Enterprise Edition
LINQ	Languafe Integrated Query
MSDN	Microsoft Developer Network
MSL	Mapping Schema Language
OOP	Object-oriented Programming
POCO	Plain Old CLR Objects
QOS	Quality of Service
RPC	Remote procedure call

SGML	Standard Generalized Markup Language
SOA	Service oriented architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSDL	Storage Scheme Definition Language
T4	Text Templating Transformation Toolkit
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
URL	Uniform Resource Locator
W3C	The World Wide Web Consortium
WS	Web Service, webové služby
WSDL	Web Service description Language
XML	Extensible Markup Language
XSD	XML Schema Definition Language
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformation Language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK