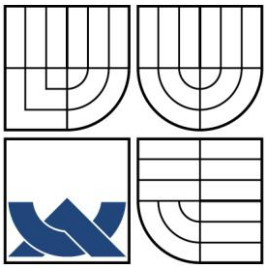
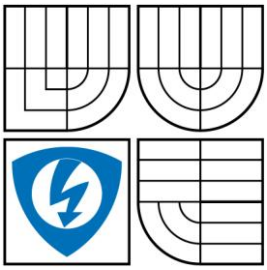


VYSOKÉ UČENÍ TECHNICKÉ
V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

VÍCEVRSTVÉ APLIKACE V PROSTŘEDÍ .NET

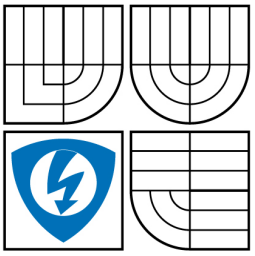
MULTILAYER APPLICATIONS IN .NET ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — MAREK PALKECH
AUTHOR

VEDOUCÍ PRÁCE — ING. JAN KACÁLEK
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Marek Palkech ***ID:*** 83388
Ročník: 2 ***Akademický rok:*** 2008/2009

NÁZEV TÉMATU:

Vícevrstvé aplikace v prostředí .NET

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se v prostředím .NET, návrhovým vzorem Model-View-Controller a s výhodami vícevrstvého návrhu aplikací. Navrhněte v prostředí .NET a jazyce C# knihovnu umožňující jednoduchou realizaci vícevrstvých aplikací pomocí návrhového vzoru Model-View-Controller.

DOPORUČENÁ LITERATURA:

[1] Evjen B., C# 2005 Programujeme profesionálně, Computer Press 2007, ISBN 80-251-1181-4
[2] Troelsen A., C# a .NET 2.0 profesionálně, ZONER Press 2007, ISBN 80-86815-42-0

Termín zadání: 9.2.2009 ***Termín odevzdání:*** 26.5.2009

Vedoucí práce: Ing. Jan Kacálek

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

UPOZORNĚNÍ:

ABSTRAKT

Práce popisuje návrhový vzor Model-View-Controller. Soustřeďuje se především na popis principu a funkčnosti jednotlivých vrstev architektury. Pojednává také o příčinách vzniku třívrstvé architektury, výhodách, které tenhle návrhový vzor poskytuje ale také o jeho nevýhodách. Pozornost je věnovaná i příkladu nejčastější implementace tohoto návrhového vzoru v praxi a to při implementaci přístupu k databáze pomocí webového rozhraní. Práce dále pojednává o platformě .NET Framework, tvořenou velmi objemnou, jazykově neutrální knižnicí, teda jakousi kolekci kódů poskytující řešení na běžné programátorské problémy a rozhraním na spouštění aplikací vytvořených v prostředí .NET. Práce se soustřeďuje na vznik platformy, velká pozornost je věnovaná zejména architektuře .NET Framework. Zmíněné jsou také jednotlivé verze .NET, technologie na přístup k datům ADO.NET a prvek ASP.NET ObjectDataSource. V kapitole popisující podporované jazyky je pozornost zaměřená na jazyk C# a jeho jednotlivé verze. Závěrečná část práce popisuje praktickou aplikaci popsaných technologií při návrhu aplikace „Vícevrstvé aplikace v prostředí .NET“. Práce popisuje architekturu aplikace se zaměřením se na implementaci jednotlivých vrstev návrhového vzoru Model-View-Controller ve formě projektů ve vývojovém prostředí Microsoft Visual Studio 2005. Velká pozornost je věnovaná každé operaci, kterou aplikace uživatelovi umožňuje s datama uloženými v jednotlivých tabulkách databáze vykonávat, jako například vkládání dát, jejich editování, mazání či výběr. Podrobně je také popsaný proces generování potomků obecného business objektu.

Klíčová slova: Model-View-Controller, vícevrstvé aplikace, platforma .NET Framework, jazyk C#, ADO.NET, business objekty

ABSTRACT

This thesis represents the Model-View-Controller pattern. It is focused especially on the description of the particular architecture layers principle and its functionality. It deals with the reasons of the three-layer architecture invention and it also deals with advantages and disadvantages provided by this pattern. The most frequent implementation of the MVC – the access to the data stored in the database through the web user interface is also described. The next part of the thesis is concentrated on .NET Framework platform created from very voluminous, language-neutral library that is basically huge collection of the source codes providing the solution for common programmer's problems and from the interface used for running up the applications created in .NET environment. The goal of the chapter concerning .NET Framework is to describe its architecture. The thesis also describes the platform invention, various versions of the .NET, the data access technology ADO.NET and the ASP.NET member ObjectDataSource. The chapter describing languages supported by the .NET Framework is focused on the C# language and its versions. The application "Multilayer applications in .NET environment" is the practical implementation of the mentioned technologies and it is described in the last chapter. The application's architecture with concentration on the particular Model-View-Controller layers implementation in the form of Microsoft Visual Studio 2005 projects is also described in the thesis. Special attention is paid to each operation over the data stored in the database tables that the application enables user to execute, as for example data inserting, updating, selecting or deleting. The common business object's child generation process is also described deep into the details.

Keywords: Model-View-Controller, multilayer applications, .NET Framework platform, C# language, ADO.NET, business objects

BIBLIOGRAFICKÁ CITÁCIA PRÁCE

PALKECH, M. *Vícevrstvé aplikace v prostředí .NET*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 68 s. Vedoucí diplomové práce Ing. Jan Kacálek.

PREHLÁSENIE

Prehlasujem, že svoju diplomovú prácu na téma Vícevrstvé aplikace v prostředí .NET som vypracoval samostatne pod vedením vedúceho diplomovej práce s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nepovoleným spôsobom do cudzích autorských práv osobnostných a som si plne vedomý následkov porušení ustanovení § 11 a nasledujúcich autorského zákona č. 121/2000 Zb., vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovení § 152 trestného zákona č. 140/1961 Zb.

V Brne dňa ....

podpis autora

POĎAKOVANIE

Ďakujem vedúcemu diplomovej práce Ing. Janovi Kacálkovi za veľmi užitočnú metodickú pomoc a cenné rady pri spracovaní diplomovej práce.

V Brne dňa ....

podpis autora

OBSAH

ZOZNAM OBRÁZKOV

1 ÚVOD

Každý človek je v súčasnosti doslova obklopený elektronikou rôzneho druhu. Rôzne prístroje nám pomáhajú prekonávať každodenné prekážky a postupom času sme na nich stali závislí. Jedným z takých prístrojov je aj počítač, bez ktorého si svoj život nevie predstaviť veľká väčšina najmä pracujúceho obyvateľstva.

Ľudstvo disponuje obrovským množstvom dát, ktoré bývali ukladané napríklad v obrovských papierových kartotékach. V terajšej dobe sú tieto dáta ukladané v elektronickej podobe. Ako najčastejšie úložisko dát slúžia relačné databázy, ktoré sú schopné tieto obrovské množstvá dát štruktúrovane uložiť. Databázy pracujúce s potrebnými dátami sa teda stali neodmysliteľnou súčasťou každej väčšej aplikácie. Ak má byť aplikácia efektívna, rýchla, spoľahlivá a má poskytovať užívateľsky príjemné rozhranie, musí efektívne komunikovať s úložiskom dát, ktoré potrebuje pre svoju prácu, väčšinou teda s relačnou databázou.

S nárastom aplikácií využívajúcich služby databáz pochopiteľne rástli aj nároky na vývojový model samotnej aplikácie. Nároky na model boli rôzne, medzi najhlavnejšie patrila možnosť spolupráce aplikácií s klientmi pripájajúcimi sa cez rôzne rozhrania, možnosť pridania nového rozhrania bez nutnosti modifikácie aplikácie, aby po zmene dát v jednom rozhraní automaticky aplikácia zmenila dáta vo všetkých ostatných rozhraniach a podobne. A práve tieto požiadavky spĺňa návrhový vzor trojvrstvovej aplikácie Model-View-Controller, ktorému je venovaná prvá časť tejto práce. V kapitole venujúcej sa tomuto návrhovému vzoru je popísaný jeho vznik, činnosť jednotlivých vrstiev architektúry, jej výhody, nevýhody a implementácie návrhového vzoru v praxi.

Druhá časť práce je venovaná platforme .NET Framework. Je to produkt firmy Microsoft, ktorého hlavnou úlohou bolo vytvoriť platformu, ktorá by odstránila zložitosť komunikácie medzi aplikáciami, priniesla kompatibilitu medzi rôznymi programovacími jazykmi a zároveň bola založená na princípoch objektovo-orientovaného programovania. Technológia je určená pre operačné systémy Microsoft Windows. Tvorí ju veľmi objemná, jazykovo neutrálna knižnica, teda akási kolekcia

kódov poskytujúca riešenia na bežné programátorské problémy a rozhranie na spúšťanie aplikácií vytvorených v prostredí .NET.

Posledná časť práce je venovaná praktickej implementácii návrhového vzoru Model-View-Controller na platforme .NET Framework. Výsledkom je aplikácia „Viacvrstvové aplikácie v prostredí .NET Framework“. Kapitola popisuje architektúru aplikácie so zameraním sa na implementáciu jednotlivých vrstiev vo forme projektov vo vývojovom prostredí Microsoft Visual Studio 2005. Následne je podrobne rozobratá funkčnosť aplikácie popísaním každej operácie, ktorú aplikácia umožňuje užívateľovi vykonať s dátami uloženými v databáze.

2 TROJVRSTVOVÁ ARCHITEKTÚRA

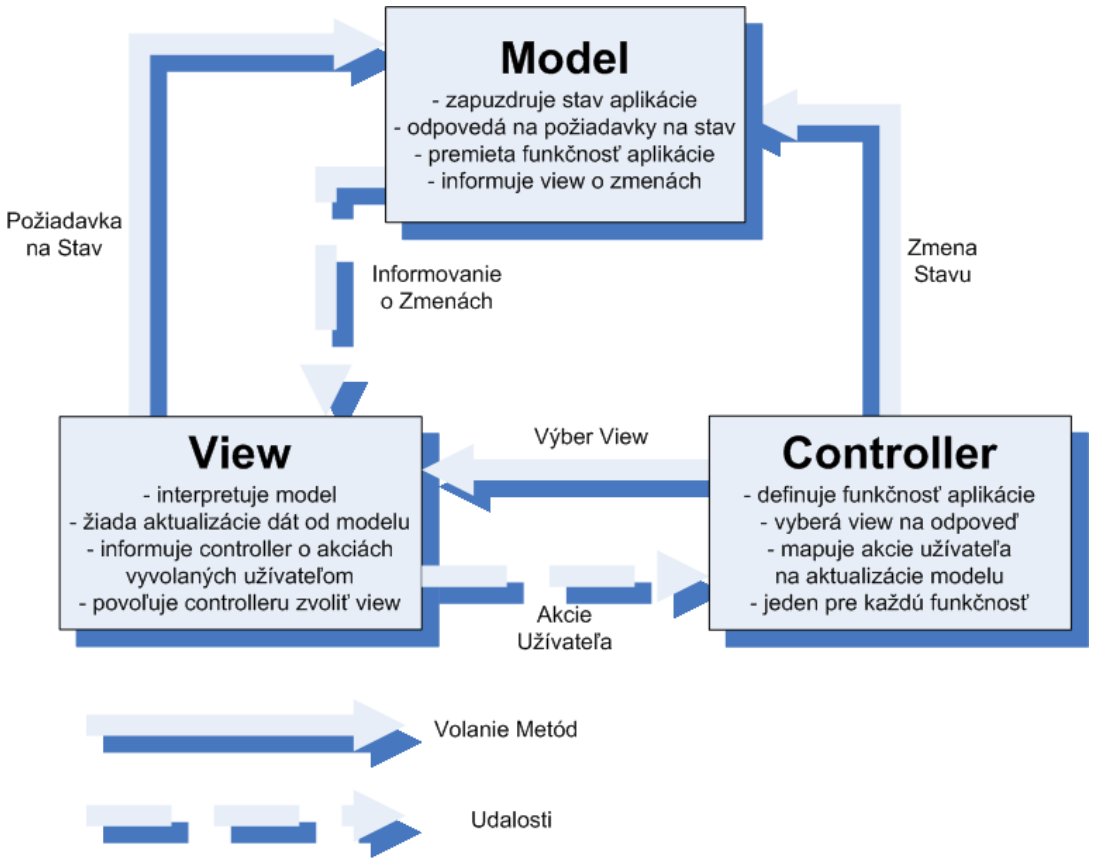
2.1 DÔVODY VZNIKU TROJVRSTVOVEJ ARCHITEKTÚRY

Hlavnou úlohou väčšiny moderných aplikácií je získať dáta z úložiska dát a následne tieto dáta zobraziť užívateľovi. Užívateľ si získané dáta môže prehliadnuť, modifikovať a zmenené dáta opäť zaslať na server. Aplikácia napokon dáta v databázy zaktualizuje. Z načrtnutého modelu je zrejmé, že hlavný tok informácií je medzi rozhraním užívateľa a úložiskom dát. Cieľom vývojárov moderných aplikácií bolo zvýšenie ich výkonu, čo bolo možné dosiahnuť redukovaním kódovania medzi spomínanými entitami. Avšak tento prístup má niekoľko veľmi vážnych problémov. Najhlavnejší problém je ten, že užívateľské rozhranie sa môže pomerne často meniť, zatiaľ čo systém úložiska dát zostáva dlhšiu dobu nemenný. To znamená, že pri zmene rozhrania by vývojár opäť musel meniť zdrojový kód aplikácie. Tento koncept sa teda neukázal ako vhodný a bolo potrebné vytvoriť model, ktorý by spĺňal nasledujúce požiadavky:

- spolupráca aplikácií s množstvom klientov s rôznymi rozhraniami (online obchod musí obsluhovať klientov pripájajúcich sa cez web (HTML, prípadne WML pre wireless zákazníkov), musí poskytovať zohranie pre dodávateľov, rozhranie pre pracovníkov a taktiež rozhranie pre správcov systému),
- zmena alebo pridanie nového rozhrania bez nutnosti modifikácie aplikácie,
- modifikácia dát v jednom rozhraní automaticky modifikuje dáta aj pre iné rozhrania (jeden užívateľ vyberá všetky dáta z databázy cez jedno rozhranie, druhý zobrazí graf len z vyfiltrovanej časti údajov pomocou druhého rozhrania, ak prvý užívateľ dáta modifikuje, druhému sa tieto zmeny musia automaticky aktualizovať),
- tvorba profesionálnych, efektívnych a zároveň vizuálne príťažlivých HTML stránok vyžaduje dokonalú znalosť problematiky tvorby HTML stránok a taktiež „vizuálne cítenie“. Málokedy dokáže oboje jedna osoba, preto je vhodné vývoj HTML stránok rozdeliť na dve separátne časti,
- čo najjednoduchší presun aplikácie z jedného zariadenia na druhé (pri presune aplikácia z počítača napríklad na PDA je potrebné urobiť rozsiahle zmeny v užívateľskom rozhraní, ale nie v samotnej aplikácii) [4].

2.2 MODEL-VIEW-CONTROLLER

V roku 1979 bol nórskym počítačovým vedcom Trygve Reenskaugom prvýkrát popísaný návrhový model, ktorý vychádza z vyššie popísaných požiadaviek. Bol nazvaný ako *Model-View-Controller (MVC)*. Tento model rozdeľuje aplikáciu na tri nezávislé komponenty – dátový model, užívateľské rozhranie a riadiacu logiku, takým spôsobom, že modifikácia niektorého z nich má len minimálny vplyv na ostatné komponenty.



Obr. 2.1: Model-View-Controller diagram

2.2.1 Vrstva Model

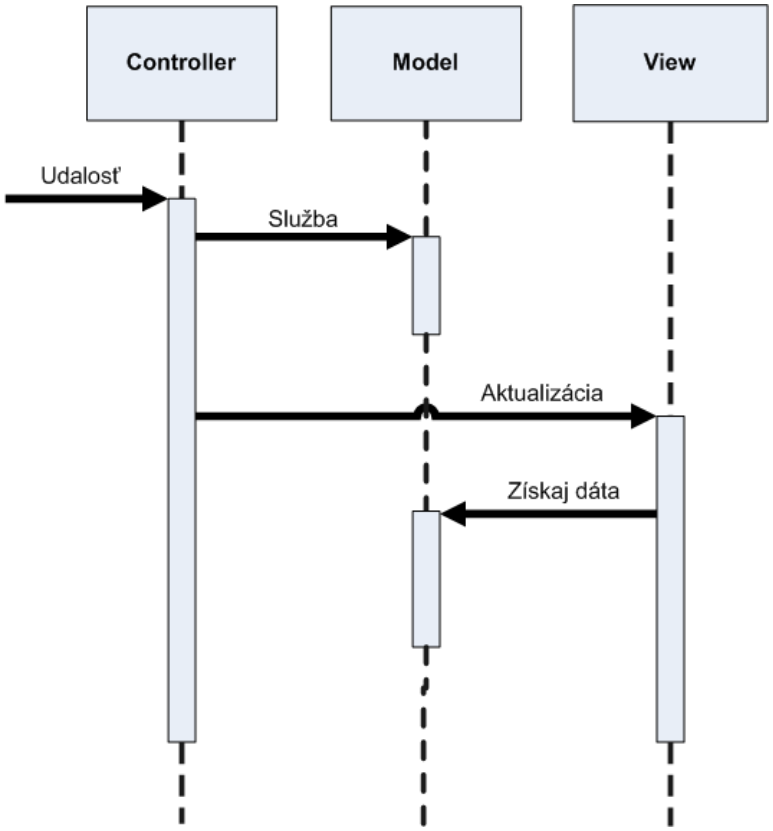
Model zaisťuje doménovo špecifickú reprezentáciu dát, s ktorými aplikácia pracuje. Doménová logika pridáva význam čistým dátam (výpočet, či má dnes užívateľ narodeniny, počítanie priemerov, súčtov a podobne). Model spravuje dáta aplikačnej domény, odpovedá na požiadavky klienta týkajúce sa dát (väčšinou od vrstvy View) a odpovedá na inštrukcie na zmenu stavu (zvyčajne od vrstvy Controller).

Ako už bolo spomínané, model nesmie byť závislí na zvyšných dvoch vrstvách architektúry. Nesmie teda vytvárať priame premenné odkazujúce sa na jednu z vrstiev. Väčšina aplikácií používa perzistentné mechanizmy na ukladanie dát (typickým príkladom perzistentného mechanizmu je databáza). MVC však nepopisuje spôsob prístupu k dátam, keďže tento prístup je priamo zapuzdrený v modeli [4].

Pasívny model

Pasívny model sa používa v prípade, keď jeden controller manipuluje výhradne s jedným modelom. Controller modifikuje model a následne informuje view, že sa model zmenil a že by sa view mal obnoviť (viď Obr. 2.2). Objekty použité v pasívnom modeli si nie sú „vedomé“, že sú použité v trojvrstvovej aplikácii, model je teda úplne nezávislý na zvyšných vrstvách. Preto nie je potrebné, aby pasívny model poskytoval mechanizmy na oznamovanie zmeny dát. Keď controller vykoná na modeli operáciu, ktorá bude vyžadovať aby sa dáta zobrazené vo vrstve view aktualizovali, vrstve view to oznámi controller.

Tento model je bežne používaný vo webových aplikáciách, keďže vrstva view je po každom cykle request/response protokolu HTTP znovu vygenerovaná, bez ohľadu na zmenu dát [4].

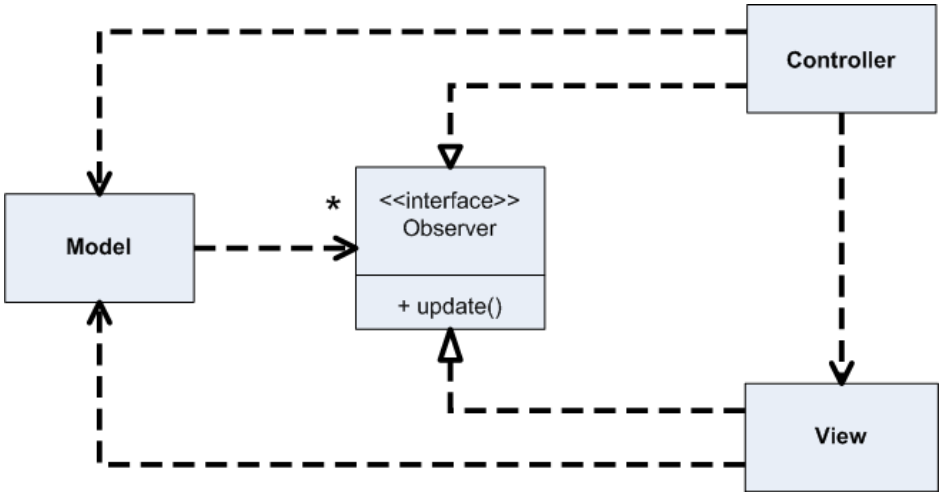


Obr. 2.2: Diagram funkčnosti pasívneho modelu

Aktívny model

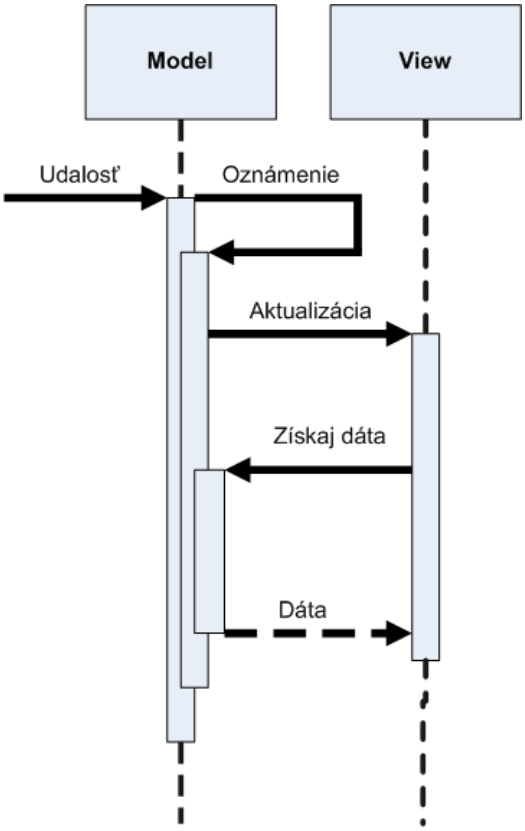
Aktívny model sa používa v prípade, keď model mení stav aj bez zapojenia controllera. Tento stav môže nastať napríklad, ak iné zdroje dát menia tieto dáta. V takomto prípade dokáže zmeny zistiť jedine model, to znamená, že práve model musí oznámiť vrstve view, aby aktualizovala display.

Avšak, ako už bolo spomenuté v predchádzajúcich častiach tejto kapitoly, jedným z cieľom architektúry MVC je, aby boli view nezávislé na modeli. Ak by model mal oznamovať view, že došlo k zmenám dát, opäť by sa do aplikácie zaviedli závislosti, ktorým sa chcelo použitím konceptu MVC predísť. Z tohto dôvodu sa v aktívnom modeli využíva tzv. *Observer* (pozorovateľ).



Obr. 2.3: Použitie Observera v architektúre MVC

Observer poskytuje mechanizmus, pomocou ktorého dokáže model informovať ostatné objekty o jeho zmene bez vytvárania akýchkoľvek závislostí medzi danými objektmi. Model zistí zoznam všetkých objektov, ktoré musia byť o zmenách informované, keď k nim dôjde. Model však nepotrebuje nejaké špecifické informácie o rôznych view. V prípadoch, keď controller musí byť o zmenách informovaný modelom (napr. v prípade povolenia alebo zamietnutia zobrazenia niektorých položiek v menu), controller musí implementovať rozhranie Observera a oznámiť modelu, aby ho o zmenách informoval. V prípadoch, keď aplikácia poskytuje viacero view, je výhodné definovať viacnásobné subjekty, kde každý z nich popisuje iný druh zmeny dát modelu a následne môže prijímať len tie informácie o zmenách, ktoré sú pre neho podstatné [4].



Obr. 2.4: Diagram funkčnosti aktívneho modelu

2.2.2 Vrstva View

Úlohou vrstvy view je zobraziť dáta obdržané od vrstvy model a zobraziť ich užívateľovi. Jedná sa teda o výstup aplikácie. View má zvyčajne voľný prístup do modelu, nesmie však meniť jeho stav. Špecifikuje formu, akou majú byť dáta zobrazené a taktiež musí vedieť reagovať, keď sa dáta v modeli zmenia. Ak aplikácia používa aktívny model, view sa dokáže na modeli registrovať a následne prijímať informácie o jeho zmenách [4].

2.2.3 Vrstva Controller

Controller spracováva a odpovedá na udalosti a vyvoláva zmeny na vrstve model, prípadne na vrstve view. Väčšinou ide o inštrukcie zadané užívateľom za pomoci klávesnice alebo myši. Metódy, ktoré menia stav modelu, sú volané práve controllermi– v pasívnom modeli to vrstve view oznamuje controller, v aktívnom modeli sa to deje pomocou mechanizmu na šírenie informácií o zmenách. Controller sa nenachádza

medzi vrstvami model a view, ale obe vrstvy majú rovnaké možnosti prístupu k modelu (viď Obr. 2.1). Controller teda nekopíruje dáta z modelu pre view, dokáže ich však meniť a o zmene následne vrstvu view informovať.

Taktiež je dôležité zdôrazniť, že view aj controller sú závislé na modeli. Model je však na týchto vrstvách nezávislý a to umožňuje to vytvárať a testovať nový modul bez ohľadu na jeho užívateľské rozhranie – a to je najväčšia výhoda separácie aplikácií na trojvrstvové [4].

2.2.4 Princíp funkčnosti

Koncept MVC môže byť realizovaný rôzne. Jeden z najčastejších princípov, ktorý sa v praxi využíva je nasledovný:

1. užívateľ vykoná nejakú akciu v užívateľskom rozhraní (napríklad klikne myšou na tlačidlo, stlačí nejakú klávesu na klávesnici atď.),
2. controller zachytí vstupnú inštrukciu z objektu užívateľského rozhrania,
3. controller oboznámi model o vykonanej akcii a prípadne zmení stav jeho dát (zaktualizuje nákupný košík užívateľa),
4. zmenené dáta sú spracované doménovou logikou (napr. prepočíta celkové náklady aj s daňami a poplatkami za dovoz materiálu pre užívateľa). Ako sú tieto dáta uložené model MVC nerieši (najčastejšie sa jedná o perzistentné uloženie dát v databázy),
5. nové dáta sú predané užívateľskému rozhraniu a rozhranie čaká na novú akciu užívateľa, čím sa celý cyklus opäť zopakuje [4].

2.2.5 Výhody a nevýhody architektúry MVC

V predchádzajúcich kapitolách bol popísaný princíp MVC a taktiež funkčnosť jednotlivých vrstiev. Väčšina výhod tejto architektúry už bola spomenutá alebo aspoň načrtnutá. Táto podkapitola poskytuje prehľadný súhrn výhod a nevýhod, ktoré separácia na jednotlivé vrstvy poskytuje.

Výhody MVC:

- *zameniteľné užívateľské rozhranie* – pre jeden model môže byť navrhnutých viac vrstiev view a controller, ktoré poskytujú alternatívne užívateľské rozhranie, to znamená, že rovnaké dáta môžeme zobraziť ako tabuľku, stĺpcový graf alebo ako koláčový graf,
- *komponenty užívateľského rozhrania* – MVC vyžaduje, aby užívateľské rozhranie aplikácie bolo štruktúrované do hierarchie objektov a taktiež definuje štandardné závislosti medzi týmito objektmi. To znamená, že môžu byť vytvorené generické verzie daných objektov. Nazývajú sa komponenty užívateľského rozhrania,
- *viacnásobné simultálne zobrazenie rovnakých dát rôznymi vrstvami view* – rôzne view môžu byť aktívne v rovnakom čase a každá z nich zobrazuje dáta nezávisle na ostatných,
- *synchronizácia* – mechanizmy na šírenie informácií o zmene dát zabezpečujú, že sa dané dáta automaticky aktualizujú okamžite po zmene vykonanej v modeli,
- *ľahké zmeny užívateľského rozhrania* – zmena v užívateľskom rozhraní neovplyvní model aplikácie,
- *jednoduchšie testovanie.*

Nevýhody MVC:

- *zložitosť*,
- *previazanosť vrstiev view a controller na model* – zmena v rozhraní modelu vyžaduje zásah do view a vo väčšine prípadov aj do controllera. To môže byť časom veľmi náročné,
- *previazanosť vrstiev view a controller medzi sebou* – je takmer nemožné tieto dve vrstvy od seba dokonale oddeliť [4].

2.2.6 Implementácie MVC

Trojvrstvová architektúra bola prvýkrát použitá v jazyku Smalltalk vyvíjanom v Xerox Research Labs. Veľa ďalších projektov sa postupom času inšpirovalo touto

architektúrou, za zmienku stoja najmä NeXTSTEP, OPENSTEP, Cocoa alebo Microsoft Foundation Classes (MFC).

V súčasnosti je najčastejšie táto architektúra použitá pri vývoji webových aplikácií. Najmä pri zložitejších aplikáciách, ktoré sa vyvíjajú pomerne rýchlo a teda dochádza k mnohým zmenám, zaručuje ich flexibilitu a spoľahlivosť. Architektúra MVC je použitá v Microsoft .NET Frameworku (ASP.NET, Maverick.NET, ProMesh.NET), v Jave (Cocoon, Google Web Toolkit, Oracle Application Framework), v Perle (Catalyst, Maypole), v PHP (ash.MVC, CakePHP, phpXCore, Zend Framework) ale taktiež v Pythone (Django), v Ruby (Ruby on Rails)alebo v SmallTalku [4], [5].

3 MICROSOFT .NET FRAMEWORK

3.1 VZNIK .NET PLATFORMY

V dobe operačných systémov MS-DOS na počítači vždy bežala v jednu dobu len jedna aplikácia. Nebolo teda potrebné riešiť problém komunikácie medzi aplikáciami. Situácia sa zmenila po príchode operačného systému Windows. Aplikácie spolu dokázali komunikovať vďaka protokolom. Tieto protokoly boli založené hlavne na správach, ktoré väčšinou obsahovali číslo, ktoré definovalo, akú akciu má aplikácia vykonať.

Kvôli rozmachu objektovo-orientovaného programovania (OOP) boli tieto jednoduché protokoly onedlho nedostačujúce. Objektovo-orientované programovanie je vo svojom princípe v rozpore so spomínanými protokolmi – vytvára objekty, ktoré majú svoje vlastnosti určené len na čítanie a nad týmito vlastnosťami operujú metódy. Časom začali vznikať komponenty, ktoré bolo možné používať aj medzi viacerými vývojovými prostrediami – tzv. *COM objekty*. Tieto objekty však mali veľké nedostatky. Ich registrácia do systému bola príliš zložitá a kompatibilita dátových typov nebola dostatočná. Bolo teda potrebné vytvoriť platformu, ktorá by čo najviac známych nedostatkov odstránila a jej implementáciou by vznikli aj mnohé ďalšie výhody – jednoduchosť komunikácie medzi aplikáciami, kompatibilita medzi rôznymi programovacími jazykmi, absolútna objektovosť, prípadne väčšia bezpečnosť. Odpoveď špecialistov z Microsoftu na túto výzvu bola technológia *.NET* [9].

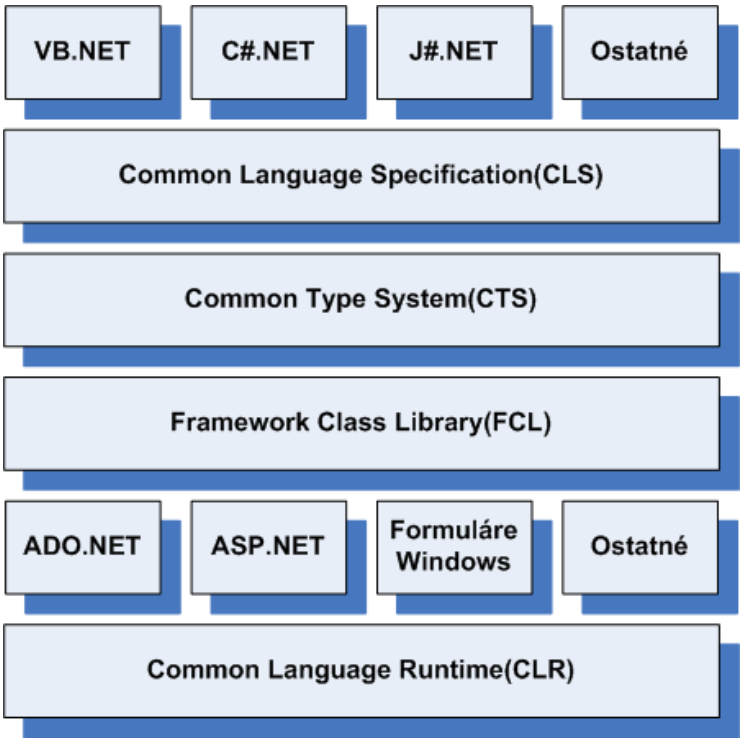
3.2 MICROSOFT .NET FRAMEWORK

3.2.1 Čo je .NET Framework

Microsoft .NET Framework je softwarová technológia dostupná k rôznym operačným systémom Microsoft Windows. Tvorí ju veľmi objemná, jazykovo neutrálna knižnica – *Framework Class Library* (FCL), teda akási kolekcia kódov, ktorá poskytuje riešenia na bežné problémy, s ktorými sa môže programátor stretnúť. Ak bude chcieť

programátor vytvoriť aplikáciu, ktorá sa napríklad bude pripájať k TCP/IP sieti, nemusí dokonale ovládať problematiku TCP/IP protokolu, ale stačí mu použiť triedu v knižnici .NET Frameworku, ktorá toto pripojenie dokáže zrealizovať. Tieto predprogramované riešenia poskytujú naozaj obrovský rozsah funkčností – užívateľské rozhrania, prístup k dátam, pripojenie k databáze, podpora vývoja webových aplikácií, kryptografiu, už spomínanú sieťovú komunikáciu, či numerické algoritmy.

Okrem spomínaných knižníc .NET Framework poskytuje aj rozhranie na spúšťanie aplikácií vytvorených v .NET. To má za úlohu *Common Language Runtime (CLR),* ktorý je taktiež súčasťou .NET Frameworku. CLR poskytuje bezpečnosť, spravuje pamäť, zabezpečuje, že programátor nemusí brať do úvahy schopnosti procesora, ktorý bude spúšťať daný program. Taktiež kontroluje bezchybnosť napísaného kódu a spravuje manipuláciu s výnimkami vzniknutými počas behu programu. Spomínané knižnice spolu s CLR tvoria teda základ Microsoft .NET Frameworku.



Obr. 3.1: Architektúra .NET Framework

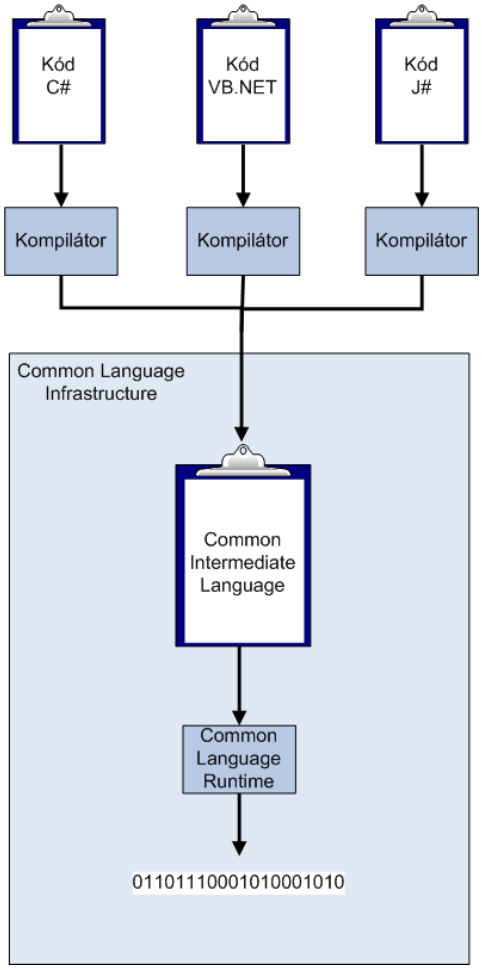
Microsoft .NET Framework bol navrhnutý, aby spĺňal nasledujúce ciele:

- poskytnúť objektovo-orientované prostredie na programovanie aplikácií nezávisle na tom, či je kód aplikácie ukladaný a spúšťaný lokálne, spúšťaný lokálne, ale distribuovaný po Internete alebo spúšťaný vzdialene,
- poskytnúť prostredie, ktoré minimalizuje konflikty vo vykonávaní kódu spôsobené odlišnými verziami softwaru,
- poskytnúť prostredie, ktoré eliminuje problémy s výkonom skriptovacích alebo interpretovacích prostredí,
- zaistiť, aby celá komunikácia bola postavená na priemyselných štandardoch a tým docieliť, aby kód založený na .NET Frameworku dokázal spolupracovať s ľubovoľným iným kódom,
- implementovať prácu s XML,
- zvýšiť bezpečnosť,
- jednoduchosť inštalácie [8].

3.2.2 Architektúra .NET

a) Common Language Runtime

Cieľom *Common Language Infrastructure* (CLI) je poskytnúť jazykovo neutrálnu platformu pre vývoj a realizáciu aplikácií. Zaisťuje manipuláciu s výnimkami vzniknutými počas behu programu, bezpečnosť a taktiež tzv. Garbage Collection (kapitola 3.2.2, odrážka g)). Microsoft svoju implementáciu CLI nazval *Common Language Runtime*.



Obr. 3.2: Štruktúra CLI

Common Language Runtime je, ako už bolo naznačené na začiatku tejto kapitoly, rozhranie, ktoré umožňuje spustiť vytvorenú aplikáciu. Pracuje ako pomocná vrstva medzi operačným systémom a aplikáciou. Hlavnou úlohou CLR je konvertovať tzv. *Managed Code* do natívneho kódu a následne spustiť program. Pod pojmom Managed Code sa rozumie kód, ktorý je spustiteľný pomocou CLR. Je robustnejší, ako kód spustiteľný priamo procesorom (tzv. Unmanaged Code), ale hlavná výhoda je v bezpečnosti. Prístup k Managed Code je len priamo z CLR a to robí aplikáciu izolovanejšou. Unmanaged Code môže volať priamo operačný systém a to spôsobuje veľké bezpečnostné riziko. Kompilátor počas kompilovania programu konvertuje zdrojový kód na MSIL (Microsoft Intermediate Language). Ide o sadu inštrukcií, ktorá je nezávislá na procesore a ktorá môže byť efektívne prekonvertovaná do natívneho kódu. Počas behu programu CLR konvertuje MSIL do natívneho kódu pre operačný systém. Keď kompilátor produkuje MSIL, produkuje taktiež aj metadáta.

Počas behu programu CLR spravuje pamäť, Garbage Collection, manipuláciu s výnimkami, Common Type System (CTS, kapitola 3.2.2, odrážka e)) a ďalšie systémové služby. Ak chceme, aby Runtime poskytoval služby nad skompilovaným kódom, kompilátor musí vytvárať metadáta, ktoré popisujú typy, členov a referencie v kóde a sú ukladané s kódom. Runtime automaticky pracuje s rozvrhnutím objektu, riadi referencie na objekt a uvoľňuje ich, keď už nebudú ďalej používané. Tento proces eliminuje zbytočné zahltenia pamäte nepotrebnými dátami, jedná sa o už spomínaný Garbage Collector [3].

b) Framework Class Library

.NET Framework Class Library (FCL) je jeden z dvoch hlavných komponentov platformy .NET Framework. FCL je knižnica, ktorá obsahuje veľké množstvo použiteľných tried, rozhraní a dátových typov, ktoré urýchľujú a optimalizujú proces vývoja aplikácie a poskytujú prístup k funkčnosti systému. Medzi najznámejšia a najpoužívanejšie patria najmä ADO.NET, ASP.NET, WinForms, Windows Presentation Foundation či Windows Communication Foundation.

FCL je organizovaná do hierarchickej stromovej štruktúry a je rozdelená do tzv. *Namespaces,* teda menných priestorov. Menné priestory členenia FCL do logických skupín v závislosti na funkčnosti, použiteľnosti a kategórie, do ktorej môžu patriť. Jedná sa teda o akési zoskupenie za účelom identifikácie. Medzi najčastejšie používané patria najmä System.* a Microsoft.*. Tieto zoskupenia môžu byť použité pre hociktorý z .NET jazykov. Triedy sú volané pomocou menných priestorov, ktoré sú umiestnené v zoskupeniach (Assemblies, viď kapitola 3.2.2, odrážka c)) [3].

c) .NET Assembly

Microsoft .NET assembly je logická jednotka kódu obsahujúca kód, ktorý vykonáva CLR. Je to kolekcia typov a zdrojových informácií, ktoré sú vytvorené, aby pracovali spolu. Počas kompilovania programu sa spolu s MSIL vytvárajú aj metadáta a tie sú uložené v súbore *Manifest*. Tento súbor obsahuje informácie o členoch,

dátových typoch, referenciách a všetkých ďalších dát, ktoré CLR potrebuje pre spustenie programu.

Každý vytvorený Assembly obsahuje Manifest a jeden alebo viac súborov. Tieto súbory môžu byť dvoch typov: procesový (s príponou *.exe) alebo knižnicový (s príponou *.dll). Existujú dva typy Assembly – privátne a zdieľané. Privátne sú používané len jednou aplikáciou a väčšinou sú uložené priamo v inštalačnom adresári aplikácie. Zdieľané môžu byť použité viacerými aplikáciami a aby sa zabezpečila ich prístupnosť, sú pridávané do tzv. Global Assembly Cache (GAC) [3].

d) Metadáta

Ako už bolo spomenuté v predchádzajúcej kapitole, metadáta sa vytvárajú počas kompilovania programu spolu s MSIL a sú uložené v súbore zvanom Manifest. V princípe sú metadáta vlastne binárna informácia, ktorá popisuje charakteristiky zdroja. Obsahuje popis Assembly, dátových typov a členov s ich deklaráciami a implementáciami, referencie na iné typy a členov, bezpečnostné povolenia a podobne. Jednoducho metadáta obsahujú všetky potrebné informácie, ktoré sú potrebné na spoluprácu s inými modulmi [3].

e) Common Type System

Common Type System popisuje sadu dátových typov použitých v aplikácii a zabezpečuje vzájomnú spoluprácu objektov vytvorených v rôznych .NET programovacích jazykoch. Tieto typy môžu byť hodnotové alebo referenčné. Vďaka CTS dokáže CLR načítať spustiť zdrojový kód napísaný v hociktorom z .NET jazykov, avšak dátové typy musia byť popísané v CTS [3].

f) Common Language Specification

Common Language Specification (CLS) je sada základných jazykových vlastností, ktoré sú potrebné na vývoj aplikácií a služieb kompatibilných s .NET Frameworkom.

CLS zabezpečuje kompletnú spoluprácu a kompatibilitu medzi aplikáciami bez ohľadu na programovací jazyk, v ktorom bola daná aplikácia vytvorená. Definuje podmnožinu CTS [3].

g) Garbage Collection

Microsoft .NET Framework poskytuje mechanizmus pomocou ktorého sa uvoľňujú z pamäte objekty, na ktoré už nie je ďalej v programe referencia a teda tieto objekty už program ďalej nepotrebuje pre svoju funkčnosť. Tento proces sa nazýva *Garbage Collecting* (GC, voľne by sa dal preložiť ako zberač odpadu). Každý objekt po vytvorení zaberá určité miesto v pamäti. Keď počas vykonávania kódu program už nemá žiadne referencie na daný objekt, pamäť objektu sa stane nedosiahnuteľná, avšak nie uvoľnená. A práve v takýchto prípadoch je užitočný GC – kontroluje, či exituje objekt, ktorý už nie je využívaný aplikáciou a ak existuje, tak pamäť použitá týmto objektom môže byť znovu použitá. Objekty bez referencií vymaže sú odstránené z pamäti a tým vzniká miesto pre nové objekty.

V platforme .NET sa o tento proces stará Garbage Collector, ktorý môže byť v programe zavolaný použitím System.GC.Collect [3].

3.2.3 Verzie platformy .NET Framework

.NET Framework 1.0

Začiatkom roku 2002 vyšla prvá verzia .NET Frameworku. Bola dostupná pre operačné systémy Windows 98, NT 4.0, 2000 a XP. Táto verzia však bola z pohľadu funkčnosti pomerne slabá. Veľmi veľa dnes už štandartných funkcií v nej chýbalo a museli sa používať importy knižníc operačného systému [8].

.NET Framework 1.1

Verzia 1.1 bola vydaná v apríli 2003, teda viac ako rok po verzii 1.0. Na rozdiel od predchádzajúcej verzie mala pomerne značný úspech a začala sa prakticky používať. Bola to prvá verzia, ktorá bola zahrnutá ako časť operačného systému (Windows Server 2003) a taktiež bola súčasťou druhej edície Microsoft Visual Studio .NET (nazvanej Microsoft Visual Studio .NET 2003). Hlavná podpora tejto verzie už skončila (október 2008),ale keďže je súčasťou Windows Server 2003, ktoré sa ešte používajú, podporovaná by mala byť do júna 2013.

Oproti predchádzajúcej verzii v nej došlo k viacerým veľmi dôležitým zmenám. Jednou z hlavných zmien bolo, že funkcie z verzie 1.0, ktoré sa v nej používali ako add-ony, sa stali súčasťou frameworku. Išlo najmä o podporu pre ASP.NET. Pribudla podpora IPv6, či podpora pre ODBC a Oracle databázy. Vznikla verzia .NET Compact Framework, ktorá je určená pre menšie zámery. V neposlednom rade pribudli aj zmeny v zabezpečení vytvorených aplikácií – v ASP.NET to bolo povolenie Code Access Security. Ide o riešenie, ktoré zabraňuje nedôveryhodnému kódu spúšťať privilegované akcie [8].

.NET Framework 2.0

Verzia .NET Framevork 2.0 vyšla spolu s Visual Studiom 2005. Ne tejto verzii je postavený aj SQL Server 2005. Je to posledná verzia podporujúca operačné systémy Windows 98, 2000 a Millenium. Software Developement Kit pre verziu 2.0 môže byť stiahnutý zdarma zo stránok Microsoftu.

Hlavnou novinkou vo verzii 2.0 sú šablóny (Generics). Programátorom, ktorý prešli z iných programovacích jazykov (najmä z C++) veľmi chýbali v predchádzajúcich verziách a preto sa Microsoft rozhodol implementovať ich. Verzia 2.0 už podporuje 64-bitovú architektúru hardwaru. Oproti verzii 1.1 pribudlo veľmi veľa vylepšení hlavne v oblasti ASP.NET komponentov (napr. personalizácia – podpora mnohých skinov a motívov) [8].

.NET Framework 3.0

Táto verzia vyšla koncom roku 2006. Je to interná časť operačných systémov Windows Vista a Windows Server 2008, ale je ju možné použiť aj pre Windows XP SP2 a Windows Server 2003.

.NET Framework 3.0 neprináša výrazné zmeny v architektúre frameworku oproti verzii 2.0, dokonca používa aj rovnaké CLR. Prináša však nový sadu managed kódov a najmä štyri nové moduly:

- *Windows Presentation Foundation* (WPF) – subsystém užívateľských rozhraní založený na XML a vektorovej grafike, používa Direct3D technológie,
- *Windows Communication Foundation* (WCF)– systém, ktorý sa snaží o zjednotenie komunikácie medzi aplikáciami (napr. zjednotením komunikačných protokolov),
- *Windows Workflow Foundation* (WF) – systém na vytváranie tzv. workflow (pracovných tokov) a ich manipuláciu,
- *Windows CardSpace* – systém na bezpečné ukladanie osobných užívateľských informácií, poskytuje jednotné rozhranie pre rozhrania, kde sa tieto informácie využívajú (napr. pri prihlásení sa na webovú stránku) [8].

.NET Framework 3.5

Táto verzia rovnako, ako verzia 3.0, používa CLR z verzie 2.0. Jej hlavným cieľom je poskytnúť podporu pre aplikácie na Windows Mobile a Windows Embedded CE.

Najväčším prínosom je rozšírenie Language Integrated Query (LINQ). LINQ dáva natívnym dátam možnosť vytvárať požiadavky na .NET jazyky použitím syntaxe pripomínajúcej SQL príkazy. Tieto požiadavky môžu byť na objekty (LINQ to Objects), XML (LINQ to XML) alebo na SQL (LINQ to SQL). Verzia 3.5 taktiež prináša nové C# 3.0 a VB.NET 9.0 kompilátory, podporu stránkovania pre ADO.NET, podporu pre http pipelining (technika, pri ktorej môže byť odoslaných viac HTTP požiadaviek bez

nutnosti čakania na odpoveď na predchádzajúcu požiadavku) a veľa ďalších vylepšení [8].

3.2.4 Nevýhody architektúry

- niektoré aplikácie vytvorené v prostredí .NET Framework potrebujú na svoj beh oveľa viac systémových prostriedkov ako podobne funkčné aplikácie, ktoré však pristupujú k systémovým prostriedkom priamo,
- použitie garbage collectora má aj svoju nevýhodu – počas odstraňovania už nepoužívaných objektov sa beh aplikácie na krátky čas zastaví (rádovo pár milisekúnd) a to môže spôsobiť nevhodnosť použitia niektorých typov aplikácií (typicky aplikácie pracujúce real-time),
- staršie verzie operačných systémov Microsoft Windows .NET Framework nepodporujú, to znamená, že pre použitím aplikácie je nutné Framework nainštalovať a to môže niektorých užívateľov odradiť [9].

3.3 JAZYKY PODPOROVANÉ .NET FRAMEWORKOM

Kompatibilita medzi viacerými jazykmi je jedna z najhlavnejších výhod .NET Frameworku. Poskytuje možnosť vzájomného ovplyvňovania komponentov .NET Frameworku bez ohľadu na jazyk, v akom sú napísané. Napríklad aplikácia napísaná v jazyku VB.NET môže odkazovať na knižnicu napísanú v jazyku C# a podobne. Táto jazyková vzájomná zlúčiteľnosť zasahuje až k dedičnosti OOP a je možná vďaka CLR.

Zrejme najpoužívanejším jazykom používaným v prostredí .NET Framework je jazyk *C#* (viď Kap.3.3.1). Tento jazyk bol navrhnutý priamo pre prostredie .NET Framework. Ďalším podporovaným jazykom je jazyk Visual Basic.NET (VB.NET), ktorého úlohou bolo nahradiť Visual Basic 6. Pre skriptovanie je dostupný jazyk JScritp.NET, ako alternatíva k C# môže byť použitý jazyk J#, ktorý je vlastne konverzia Javy. Podporované sú aj jazyky APL, Cobol, C++, Forth, Mercury, Pascal, Perl, Python Scheme, Small Talk a veľa ďalších (celkovo je ich 44) [9].

Nasledujúca kapitola sa bude bližšie venovať jazyku C#, keďže v tomto jazyku bude písaná aj moja diplomová práca.

3.3.1 Jazyk C#

Jazyk C# bol vyvinutý firmou Microsoft zároveň s platformou .NET. Ide o objektovo-orientovaný jazyk. Jeho vývoj bol poznačený snahou spojiť komplexnosť a silu jazyka C++ s jednoduchosťou Visual Basicu. Je veľmi podobný jazyku Java od spoločnosti SUN, ale keďže vznikol neskôr, vývojári sa snažili poučiť z chýb Javy. Ide teda o pomerne jednoduchý, viacúčelový jazyk, ktorý podporuje princípy softwarového inžinierstva (garbage collector a iné). Aplikácie vytvorené v tomto jazyku sú veľmi rýchle a k pamäťovým prostriedkom pristupujú veľmi efektívne, avšak v porovnaní s aplikáciami napísanými v jazyku C sú menej efektívne.

Verzie jazyka C#

- *C# 1.0* – ide o prvú verziu jazyka vytvorenú priamo s prostredím .NET Framework 1.0 vydanú v roku 2002. Ako už bolo spomenuté, vychádzala z jazykov C++ a Javy. Už prvá verzia C# podporovala objektovo-orientované programovanie,
- *C# 2.0* – verzia vydaná v roku 2005 priniesla množstvo veľmi užitočných vylepšení jazyka – za zmienku stoja najmä iterátory, statické triedy, podpora generík na úrovni CLI,
- *C# 3.0* – verzia vydaná koncom roka 2007 spoločne s novou verziou .NET Frameworku 3.5 a vývojovým prostredím Visual Studio 2008 taktiež prinášajúca mnohé vylepšenia (LINQ, ananymné triedy + kľúčové slovo var pre ich použitie, lambda výrazy). Jej veľkou výhodou však je, že aplikácie napísané v tejto verzii budú spustiteľné aj na staršej verzii .NET Frameworku v prípade, že budú s aplikáciami distribuované aj potrebné kniťžnice [9].

3.4 ADO.NET

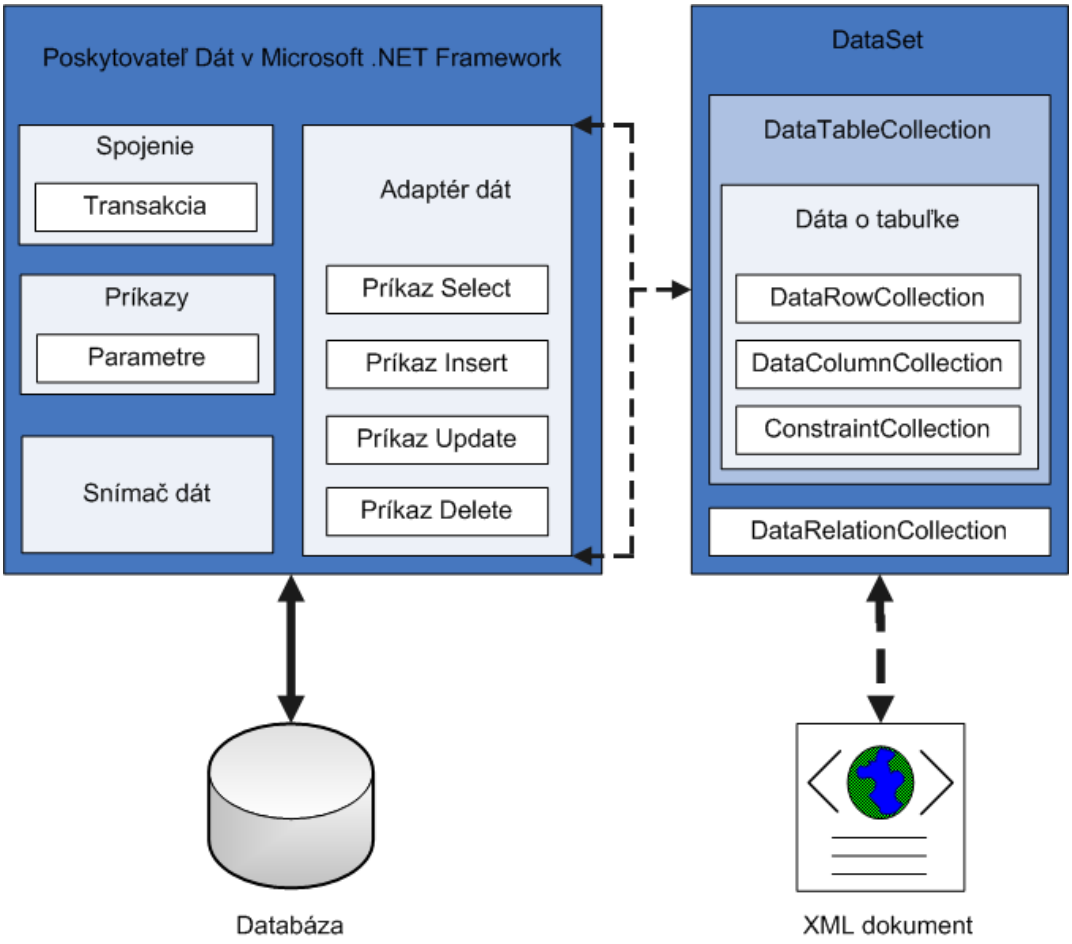
Väčšina súčasných aplikácií potrebuje k svojej činnosti prístup k dátam, tieto dáta sú najčastejšie uložené v databáze. Ak má aplikácia správne fungovať, je potrebné, aby

sa k potrebným dátam dokázala kedykoľvek dostať. Rôzne aplikácie majú rôzne požiadavky na prístup do databázy. Prostredie .NET Framework využíva svoju vlastnú technológiu na prístup k dátam a ich následnú modifikáciu – ADO.NET [7].

3.4.1 Architektúra ADO.NET

ADO.NET bol navrhnutý tak, aby spĺňal potreby vývojárov moderných aplikácií – reprezentácia dát s možnosťou kombinácie dát z viacerých rôznorodých zdrojov, úzka spolupráca s XML, optimalizované príslušenstvo na podporu vzájomného ovplyvňovania sa s databázou.

Architektúra ADO.NET je založená na dvoch hlavných komponentoch – *Data Provides* a *DataSet*. Tieto komponenty sa môžu používať samostatne alebo spojito.



Obr. 3.3: Architektúra technológie ADO.NET

ADO.NET Data Providers

Data providers poskytujú prístup k dátam. Ich úlohou je predovšetkým zabezpečiť pripojenie k databáze, vykonať daného príkazu a následné získať výsledok predchádzajúcej požiadavky. Tieto výsledky môžu byť buď spracované priamo, alebo môžu byť umiestnené do tzv. ADO.NET DataSetu (viď. kapitola 3.4.1.2). Data Provider bol navrhnutý tak, aby vytváral len minimálnu vrstvu medzi zdrojovým kódom a dátami v databáze a najmä aby zvyšoval výkon aplikácií bez nutnosti zníženia funkčnosti.

.NET Framework Data Provider sa skladá zo štyroch hlavných objektov:

- *Connection* – vytvára spojenie so zdrojom dát,
- *Command* – vykonáva príkazy nad zdrojom dát,
- *DataReader* – číta dáta získané zo zdroja dát,
- *DataAdepter* – vypĺňa DataSet a rieši zmeny v zdroji dát (SQL príkazy Select, Insert, Delete, Update).

Platforma .NET Framework poskytuje štyri rôzne typy Data Providerov v závislosti na zdroji dát:

- *Data Provider pre SQL Server* – používa svoj vlastný protokol na komunikáciu s SQL serverom (verzia Microsoft SQL Server 7.0 alebo novšia), je veľmi rýchly, keďže pristupuje k dátam priamo, bez pridávania ODBC prípadne OLE DB vrstvy, k jeho triedam je možné pristupovať cez System.Data.SqlClient,
- *Data Provider pre OLE DB* – na prístup k dátam používa natívne OLE DB cez COM (Component Object Model), podporuje lokálne aj distribuované transakcie, triedy na prácu s Data Providerom pre OLE DB sú prístupné cez System.Data.OleDb,
- *Data Provider pre ODBC* – používa natívny ODBC Driver Manager cez COM na prístup k dátam a taktiež podporuje jednak lokálne aj distribuované transakcie, k jeho triedam je možné pristupovať cez System.Data.Odbc,
- *Data Provider pre Oracle* – umožňuje prístup k dátam uloženým v Oracle databáze cez Oracle software pre klientské pripojenie (verzia 8.1.7 a novšie), podporuje oba typy transakcií, triedy sú prístupné cez System.Data.Oracle [7].

ADO.NET DataSet

ADO.NET DataSet je pamäťová reprezentácia dát. To znamená, že DataSet má v pamäti uložené dáta získané z databázy a s týmito dátami pracuje, pripojenie k databáze je v tom čase odpojené. Dáta sa teda modifikujú v pamäti nezávisle na databáze. Po ukončení zmien dát v DataSete sa tieto zmeny pomocou Data Providerov aktualizujú aj v úložisku dát.

Dáta uložené v DataSete nesú informácie o štruktúre databázy, to znamená, že sa v nich nachádza zoznam tabuliek (DataTableCollection). Každý zoznam tabuliek obsahuje informácie o štruktúre samotnej tabuľky – zoznam stĺpcov (DataColumnCollection), zoznam riadkov (DataRowCollection) a zoznam relácií tabuľky (DataRelationCollection) [7].

3.5 OBJECTDATASOURCE

ADO.NET DataSet bol po svojom vzniku veľmi obľúbený. Je integrovaný priamo vo Frameworku a veľkou výhodou je aj jeho veľká všestrannosť. Avšak projekty sa časom stávali väčšie, zložitejšie, nároky na komplexnosť narastali a tak sa práca s DataSetmi stávala čoraz obtiažnejšia. Postupne sa zistilo, že pri komplexných aplikáciách je výhodnejšie mať model založený na objektoch, ako práca s nízko-úrovňovými dátami ako sú napr. riadky alebo stĺpce. Veľkým krokom vpred bol príchod infraštruktúry podporujúcej *data binding* (proces zostavujúci spojenie medzi užívateľským rozhraním aplikácia a logikou modelu). Tvorbu efektívnych vlastných objektov využívajúcich data binding podporuje práve *ObjectDataSource*. ObjectDataSource je teda objekt kontroly zdroju dát v ASP.NET. Podporuje model vykonávajúci data binding na rozličné úložiská dát ako napríklad SQL databáza alebo XML dokument. Vytvára medzi vrstvu medzi prezentačnou vrstvou a úložiskom dát s možnosťami získavania a modifikácie dát [6].

3.5.1 Business Object

Väčšina ovládačov zdrojových dát (napr. SqlDataSource) sú používané v aplikáciách s dvojvrstvovou architektúrou, kde prezentačná vrstva (napr. webová stránka) komunikuje priamo s dátovou vrstvou (databáza, XML dokument). Cieľom je však tieto dve vrstvy od seba oddeliť a logiku modelu zapuzdriť do modelu, ktorý vytvorí medzivrstvu medzi týmito dvoma vrstvami. Tento objekt sa nazýva *business object*. ObjectDataSource podporuje trojvrstvovú architektúru práve pomocou business objektov.

ObjectDataSource umožňuje vykonávať rôzne príkazy nad dátami pomocou business objektov:

- *selecting* – výber dát z databázy,
- *sorting* – zoradenie dát podľa určitých kritérií,
- *filtering* – vyfiltrovanie údajov získaných metódou Select,
- *paging* – ak je zadaný maximálny počet záznamov, ktoré chceme metódou Select zobraziť a je zadané taktiež poradové číslo prvého záznamu, táto metóda umožňuje stránkovať dáta získané metódou Select,
- *updating* – aktualizácia dát v databáze,
- *inserting* – vkladanie dát do databázy,
- *caching* – umožňuje „cachovať“ dáta [6].

4 VIACVRSTVOVÁ APLIKÁCIA V PROSTREDÍ .NET

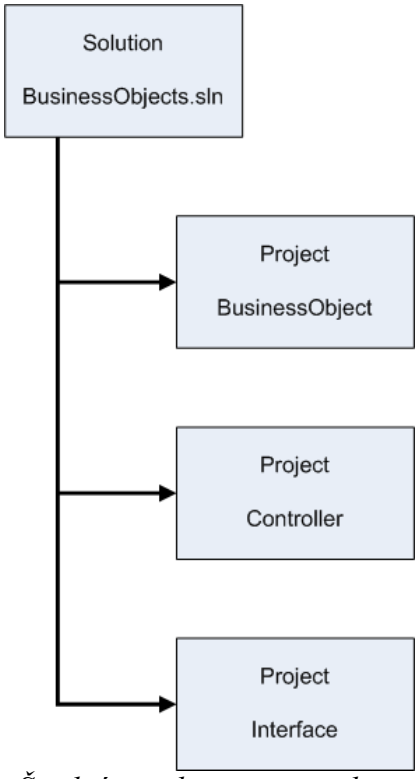
S dátami uloženými v databáze sa pracuje pomerne obtiažne. Platforma .NET Framework je spoločne s jazykom C# veľmi silný nástroj a s ich pomocou je možné vytvoriť trojvrstvovú aplikáciu založenú na princípoch MVC. Vytvorenie tejto aplikácie bolo jedným z cieľov tejto diplomovej práce.

Základom aplikácie je knižnica napísaná v jazyku C#, teda akýsi obecný business objekt, ktorý generuje business objekty konkrétneho typu nad tabuľkami databázy. Business object obecného typu obsahuje základné virtuálne metódy na prácu s databázou (update, insert, select, delete atď.) a metódy zabezpečujúce funkčnosť aplikácie (generovanie business objektov, generovanie webových formulárov atď.), ktoré sú zdedené v jeho potomkoch a v nich sa tieto metódy môžu následne modifikovať (princíp *partial classes*). Metódy priamo implementujú vytváranie príslušných SQL príkazov, takže užívateľ môže túto časť práce s databázou úplne vynechať. Potomkovia obecného business objektu slúžia už ako samotné implementácie knižnice pre vybranú databázu. Atribúty týchto potomkov sú typované, to znamená, že dátový typ atribútu je priamo závislý na konkrétnom type dátového typu odpovedajúceho stĺpca v databáze. Pomocou systémového katalógu databázy sú SQL dátové typy premapované na C# dátové typy. Táto časť aplikácie slúži ako kontrola pre programátora, pretože kompilátor ešte pred začatím vykonávania samotných SQL príkazov zistí, že aplikácia chce vkladať nekorektné dáta a následne ho na túto chybu upozorní.

Vyvinutá aplikácia je založená na princípe trojvrstvovej architektúry Model- View-Controller. Na vývoj aplikácie bola použitá platforma Microsoft .NET Framework a vývojové prostredie Microsoft Visual Studio 2005. Aplikácia je navrhnutá pre databázový server Microsoft SQL Server 2005. SQL príkazy sú spracované databázovým enginom a výsledky príkazov sú následne zobrazené užívateľovi. Funkčnosť aplikácie je nezávislá na type dát v databáze. Súčasťou aplikácie je aj užívateľské rozhranie, umožňujúce užívateľom pomerne jednoducho dané business objekty využívať.

4.1 ARCHITEKTÚRA APLIKÁCIE

Ako už bolo spomenuté, aplikácia je vytvorená vo vývojovom prostredí Microsoft Visual Studio 2005 a napísaná je v jazyku C#. Aplikácia je zložená z troch vrstiev – z vrstvy Model, vrstvy View a vrstvy Controller, pričom každá z nich je vytvorená ako samostatný projekt. Tieto tri projekty sú obsiahnuté v jednom tzv. solution (BusinessObjects.sln).



Obr. 4.1: Štruktúra solution a projektov aplikácie

4.1.1 Projekt BusinessObject

Funkčnosť vrstvy Model zabezpečuje projekt BusinessObject. Obsahuje súbor businessObject.cs obsahujúci zdrojové kódy základov aplikácie a generované súbory (business objekty) s názvami v tvare GBO-[názov_tabuľky].cs ktoré slúžia na prácu s konkrétnymi tabuľkami databázy. Táto vrstva je v aplikácii tvorená dynamickou knižnicou DLL (BusinessObejct.dll). Jej úlohou je vygenerovať spomínané business objekty nad zvolenými tabuľkami v databáze, vygenerovať webové formuláre pre užívateľské rozhranie pre tieto business objekty a následne zabezpečovať funkčnosť celej aplikácie poskytovaním základných metód nad dátami uloženými v SQL databáze, ako napríklad vkladanie, mazanie, aktualizácia či výber dát, práca s filtrami a podobne.

4.1.2 Projekt Controller

Tento projekt vykonáva funkcie vrstvy Controller. Jeho jadrom je súbor eventHandlers.cs, ktorého úlohou je obsluhovať udalosti počas behu programu. V aplikácii obsluhuje dve udalosti – zaisťuje, aby pri generovaní súborov bola vždy zvolená aspoň jedna tabuľka a zabraňuje presmerovaniu na stránky, na ktorých užívateľ pracuje s dátami v databáze, pokiaľ ešte neboli vygenerované potrebné súbory na prácu s nimi.

4.1.3 Projekt Interface

Projekt Interface je vlastne grafické užívateľské rozhranie. Ide o webové rozhranie vytvorené v jazyku ASP designované pre prehliadač Mozilla Firefox. Táto vrstva zobrazuje užívateľovi dáta získané z vrstvy model, jedná sa teda o výstup aplikácie. Ďalej umožňuje užívateľovi aplikáciu pomerne jednoducho a intuitívne používať.

4.2 FUNKČNOSŤ APLIKÁCIE

4.2.1 App.config

Pred prvým spustením aplikácie je potrebné vytvoriť dva konfiguračné súbory. Prvým je súbor app.config v projekte BusinessObjects a je nutné ho vytvoriť v prostredí Visual Studio nasledujúcim spôsobom:

1.) kliknúť pravým tlačidlom myši na projekt BusinessObjects,

2.) kliknúť na Add a zvoliť možnosť New Item...,

3.) zvoliť možnosť Application Configuration File a zmeniť názov na app.config a kliknúť na tlačidlo Add,

4.) súbor app.config by mal vyzerať podobne ako na obrázku 4.2, kde namiesto server_name bude názov SQL servera, ku ktorému sa chce užívateľ pripojiť, namiesto db_name bude názov databázy, ku ktorej sa chce užívateľ pripojiť, namiesto user_name bude prihlasovacie meno užívateľa a namiesto user_pwd bude jeho heslo.

App.config*

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="constring" value="server=server_name; database = db_name;
         uid = user_name; pwd =tuser_pwd; MultipleActiveResultSets=True"/>
  </appSettings>
</configuration>
```

Obr. 4.2: Súbor app.config

Tento súbor slúži ako konfiguračný súbor aplikácie, ktorý môže obsahovať informácie potrebné pre aplikáciu. Tá tieto informácie číta počas behu programu. Informácie sú vkladané do hlavného tagu appSettings, atribút key definuje kľúč, pomocou ktorého je možné sa na danú informáciu odkázať, atribút value definuje hodnotu tohto kľúča. Pomocou direktívy ConfigurationManager.AppSettings[“”] je potom možné sa kdekoľvek v kóde odkazovať na zadanú hodnotu.

4.2.2 Web.config

Projekt Interface je navrhnutý ako webová aplikácia, preto je ako konfiguračný súbor použitý súbor web.config, ktorý je však vytvorený obdobným spôsobom. Tento súbor má rovnaké využitie ako súbor app.config (viď kapitola 4.2.1). Pre jeho vytvorenie je potrebné postupovať nasledovne:

1.) kliknúť pravým tlačidlom myši na projekt Interface,

2.) kliknúť na Add a zvoliť možnosť New Item... ,

3.) zvoliť možnosť Web Configuration File a zmeniť názov na web.config a kliknúť na tlačidlo Add,

4.) súbor web.config by mal vyzerať podobne ako na obrázku 4.3, kde namiesto server_name bude názov SQL servera, ku ktorému sa chce užívateľ pripojiť, namiesto db_name bude názov databázy, ku ktorej sa chce užívateľ pripojiť, namiesto user_name bude prihlasovacie meno užívateľa a namiesto user_pwd bude jeho heslo.

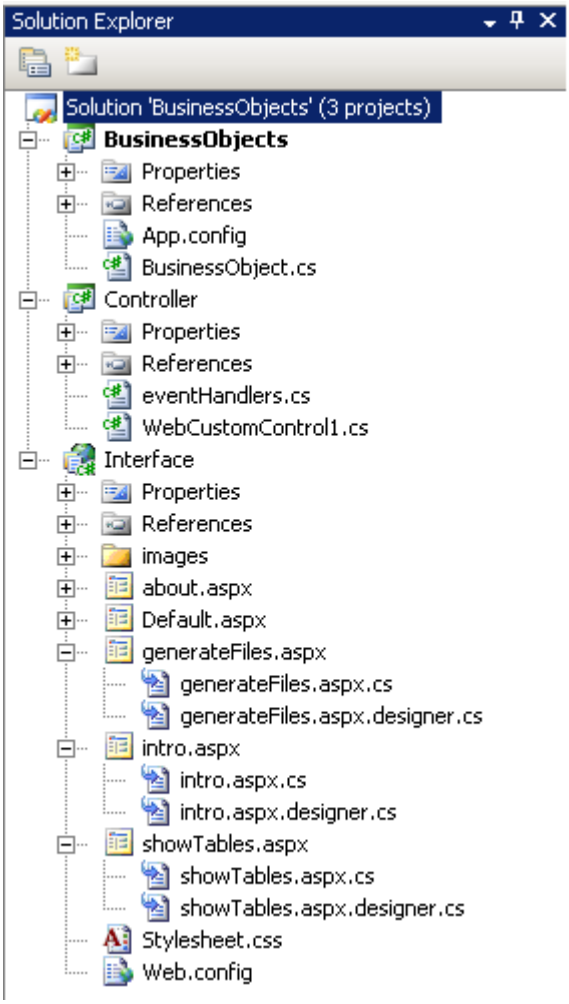
Web.config*

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="constring" value="server=server_name; database = db_name;
         uid = user_name; pwd =tuser_pwd; MultipleActiveResultSets=True"/>
  </appSettings>
    <connectionStrings/>
    <system.web>
        <compilation debug="true" />
        <authentication mode="Windows" />
    </system.web>
</configuration>
```

Obr .4.3: Súbor web.config

4.2.3 Počiatočný stav

Po vytvorení súborov podľa návodov z kapitol 4.2.1 a 4.2.2 vyzerá okno s jednotlivými súbormi v solution nasledovne:

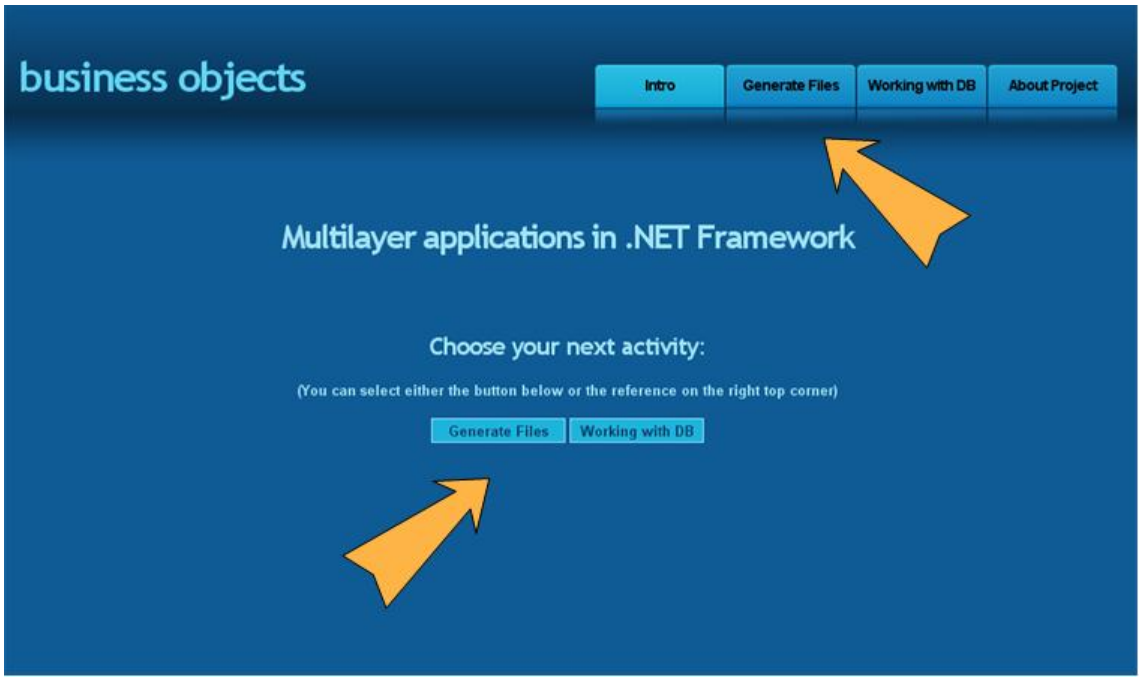


Obr. 4.4: Solution Explorer pred spustením aplikácie

Pred prvým spustením aplikácie je potrebné solution BusinessObjects skompilovať (kliknutím pravým tlačidlom myši na Solution a vybraním možnosti Build Solution). V prípade, že kód neobsahuje žiadne syntaktické chyby, zdrojové kódy budú preložené a aplikáciu bude možné spustiť.

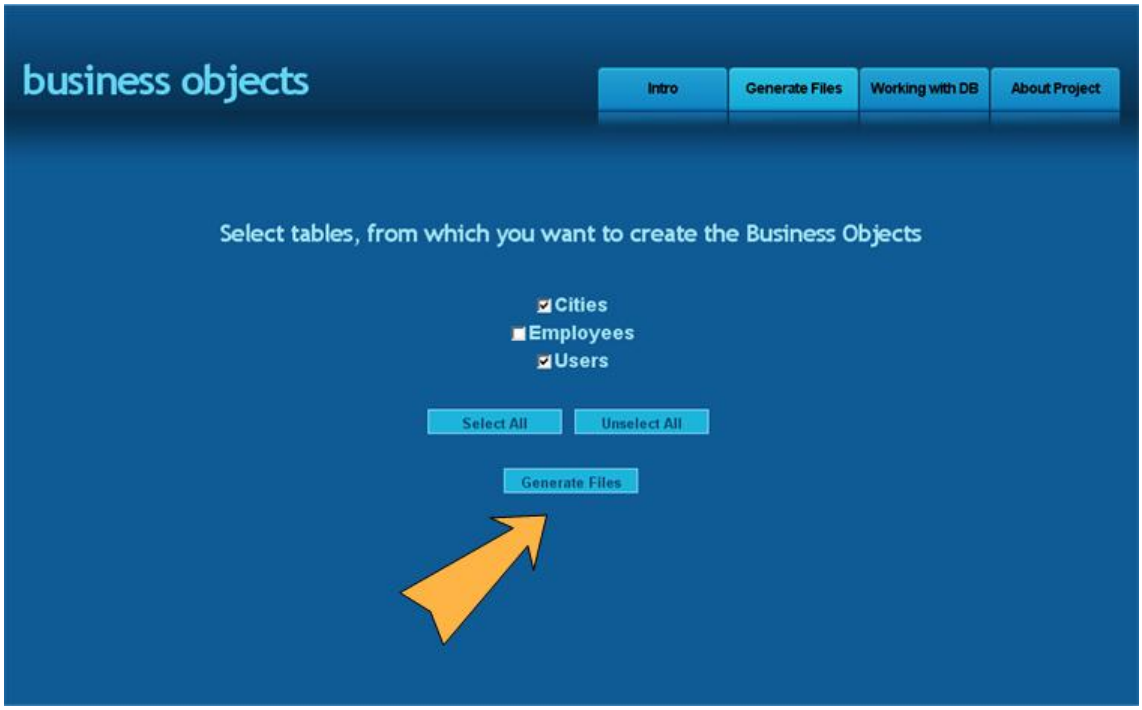
4.2.4 Generovanie súborov

Aplikácia v počiatočnom stave nie je schopná pracovať s dátami v databáze, preto je nutné vygenerovať súbory, ktoré túto prácu umožňujú. K formuláru, ktorý umožňuje generovanie súborov, je možné sa dostať z úvodnej stránky kliknutím na tlačidlo Generate Files, prípadne pomocou menu umiestneného v pravej hornej časti stránky (viď. Obr. 4.5).



Obr. 4.5: Možnosti presmerovania na stránku obsluhujúcu generovanie súborov

Po presmerovaní sa užívateľovi zobrazia všetky tabuľky, ktoré sa nachádzajú v databáze. Užívateľ zvolí tabuľku, prípadne tabuľky, s ktorými chce pracovať (viď Obr. 4.6) a aplikácia zavolá pomocou správcu udalostí metódy na generovanie súborov(generate_tableName_designer(string table), generate_tableName_cs(string table), generate_tableName_aspx(string table), generate_TableSelect() a generateBO(string table)). Tieto metódy sa zavolajú pre každú vybranú tabuľku zvlášť, v prípade, že užívateľ nijakú tabuľku nezvolil, správca udalostí vygeneruje varovné okno.

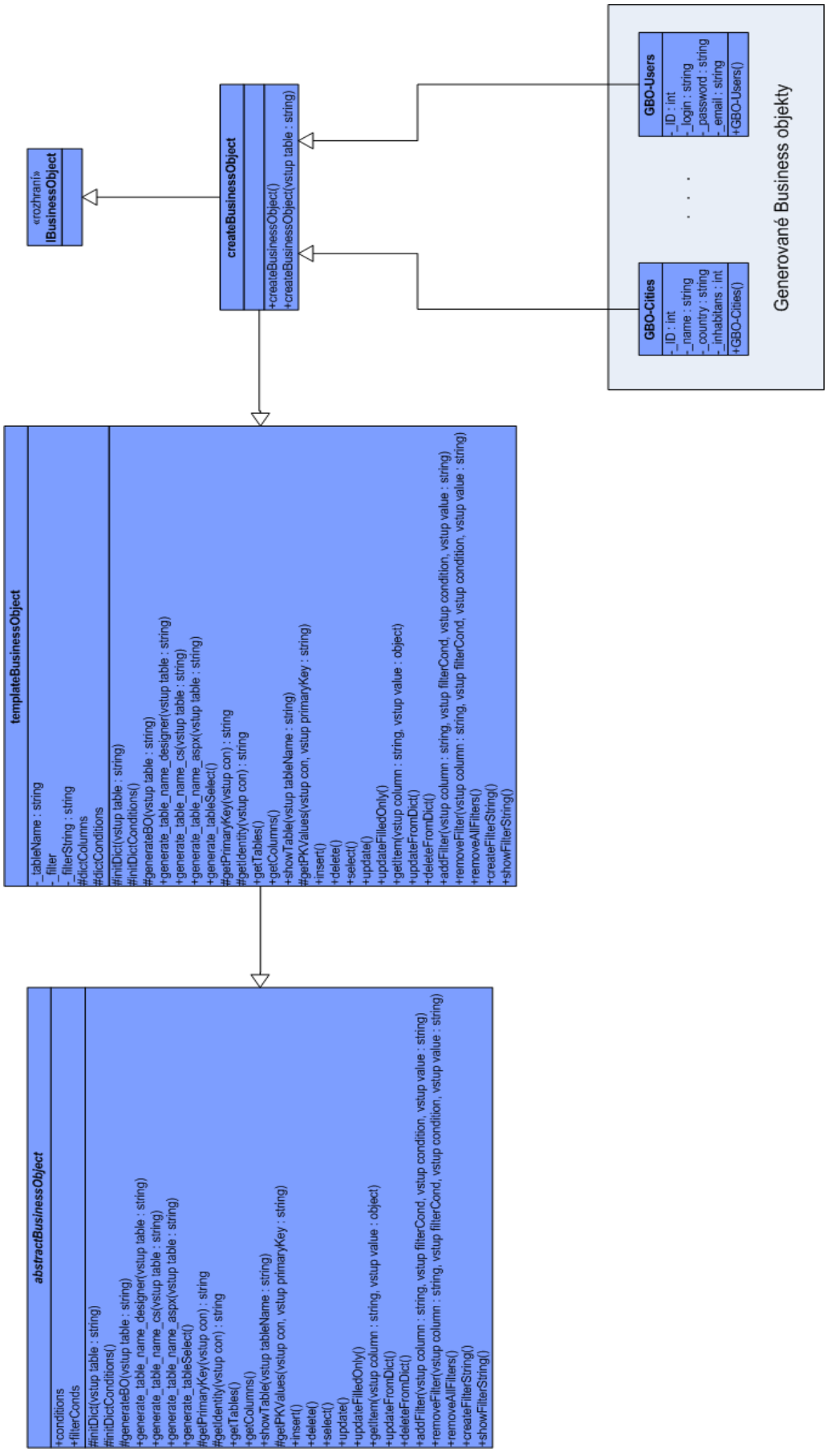


Obr. 4.6: Výber tabuliek

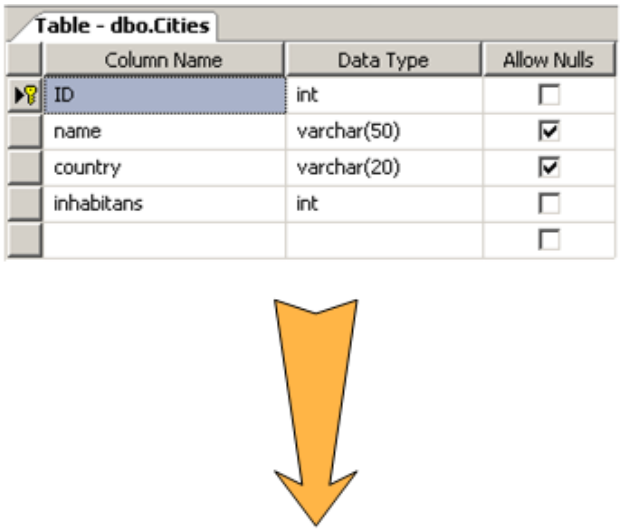
Kliknutím na tlačidlo Generate Files sa spustia nasledovné procedúry:

a) Generovanie Business objektov

Každá tabuľka v databáze má rôzny počet stĺpcov s rôznymi názvami a s rôznymi dátovými typmi. Z toho dôvodu je nemožné vytvoriť všeobecný business objekt, ktorý by bol rovnaký pre každú tabuľku. Aplikácia preto generuje business objekty, ktoré sú špecifické priamo pre tabuľku, pre ktorú chcel užívateľ objekt vytvoriť. Všetky generované objekty sú dedené z obecného business objektu (templateBusinessObject) (kompletný UML diagram aplikácie viď Obr. 4.7). Ich atribúty sú odvodené zo stĺpcov daných tabuliek – názov atribútu je zhodný s názvom stĺpca v tabuľke a jeho dátový typ je premapovaný z SQL dátového typu na C# dátový typ (príklad viď Obr. 4.8). Objekty generuje metóda generateBO(string table) a ukladajú sa do adresára C:\GeneratedFiles\BusinessObjects a ich názov je vo formáte GBO-[názov_tabuľky].cs.



Obr. 4.7: UML diagram aplikácie




```
public class GBO_Cities : templateBusinessObject<GBO_Cities>, IBusinessObject
{
    private System.Int32 _ID;
    private System.String _name;
    private System.String _country;
    private System.Int32 _inhabitans;
```

Obr. 4.8: Príklad mapovania dátových typov SQL na C#

b) Generovanie webových formulárov na prácu s dátami v databáze

Webové formuláre sú vygenerované špeciálne pre každú zvolenú tabuľku a teda sú pre každú tabuľku špecifické. Ich úlohou je sprostredkovať užívateľovi webové rozhranie na prácu s dátami v databáze. Generujú sa do adresára C:\GeneratedFiles\WebForms. Jedná sa o tri formuláre, ktorých názvy sú vo formáte:

- [názov_tabuľky].aspx – ide o ASP.NET stránku pozostávajúcu z kódu a značiek. Stránky sú vytvorené dynamicky a sú spustené na serveri, ktorý renderuje klientskému prehliadaču odpoveď. Keď klient žiada zdroje .aspx, ASP.NET v priebehu programu rozoberie a vytvorí cieľový súbor do .NET Framework triedy [1]. Súbor generuje metóda generate_tableName_aspx(string table).
- [názov_tabuľky].aspx.cs – dynamický kód bežiaci na serveri je v ASP možné uložiť v samostatnom zdrojovom súbore a znížiť tak zaťaženie servera pri obsluhovaní klientov (tzv. *code behind*). Tento súbor má rovnaký názov ako ASP stránka, avšak po prípone .aspx nasleduje ešte prípona .cs (ak je súbor písaný v jazyku C#, pre súbory písané napr. v jazyku VB sa používa koncovka .vb). Ide teda o súbor, ktorý obsluhuje akcie na udalosti, ktoré počas behu stránky nastanú, prípadne môžu nastať. Typicky sa

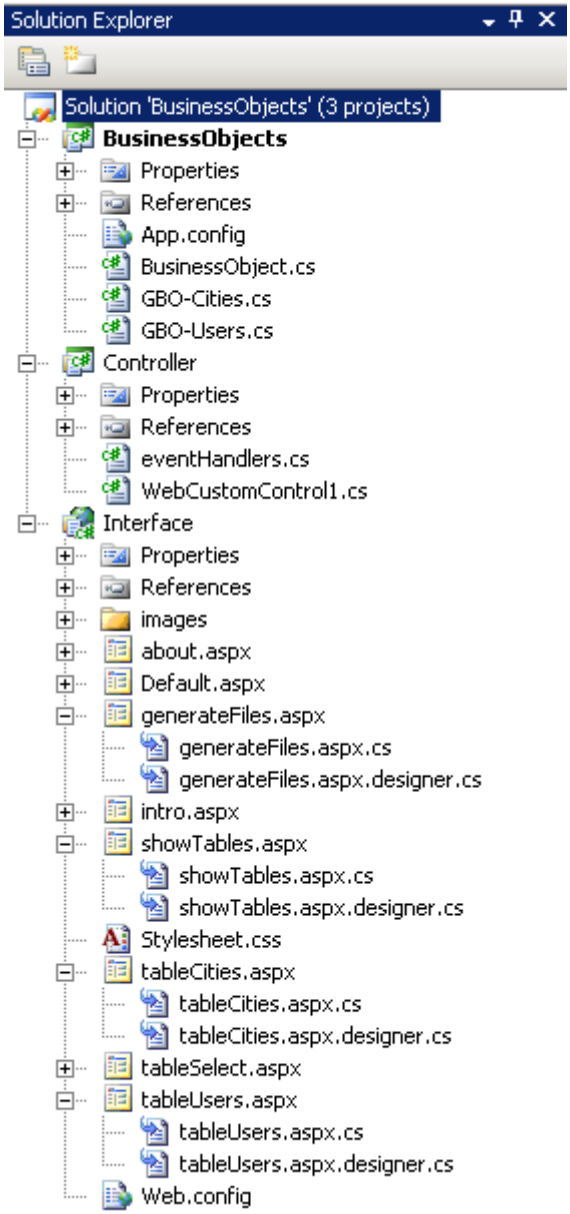
jedná o udalosti ako napríklad načítanie stránky, kliknutie na nejaký ovládací prvok a podobne. Metóda generate_tableName_cs(string table) generuje práve tento typ súborov.

- [názov_tabuľky].aspx.cs.designer – obsahuje deklarácie pre ovládacie prvky použité na .aspx stránke, generované sú metódou generate_tableName_designer(string table).

c) Generovanie tableSelect

Aplikácia zavolaním metódy generate_TableSelect vygeneruje tri súbory – tableSelect.aspx, tableSelect.aspx.cs a tableSelect.aspx.designer.cs, ktoré sú vytvorené v adresári C:\GeneratedFiles\WebForms. Tento webový formulár slúži ako pomocná stránka, ktorej úlohou je vytvoriť objekt rovnakého typu, ako je názov tabuľky, ktorá bola predtým uložená do SESSION. Vytvorený objekt uloží do SESSION a následne vykoná presmerovanie na vygenerovanú stránku pomocou ktorej užívateľ môže pracovať s dátami vo zvolenej tabuľke.

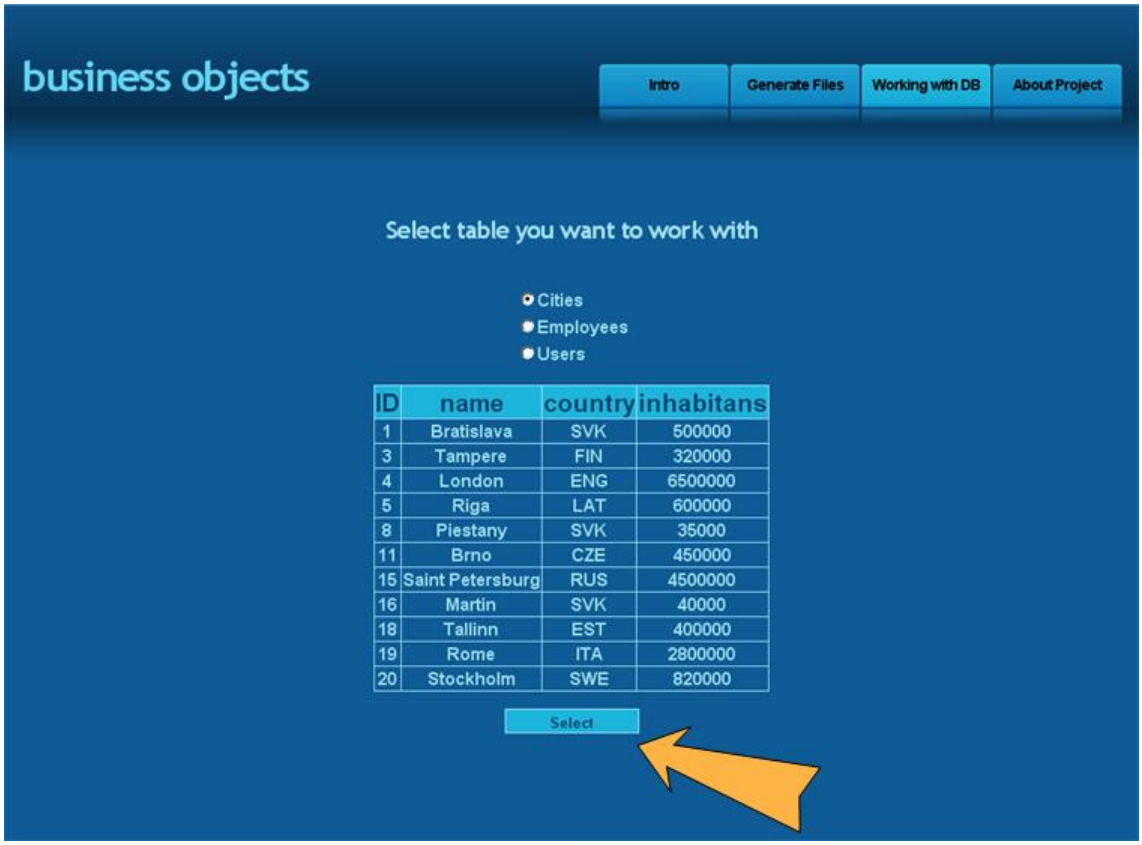
Vygenerované súbory je potrebné pridať do príslušných projektov (pravým kliknutím myši na názov projektu, zvoliť Add -> Existing Item, vybrať cestu k súborom a kliknúť na Add), teda business objekty je nutné pridať do projektu BusinessObjects a webové formuláre do projektu Interface. Tento proces je potrebné zopakovať aj po každom ďalšom generovaní súborov a následne je nutné opäť celé solution skompilovať. Na obrázku 4.9 je príklad zobrazenia Solution Exploreru po pridaní súborov vygenerovaných pre tabuľky Users a Cities z databázy.



Obr. 4.9: Solution Explorer po pridaní súborov

4.2.5 Výber tabuľky

Práca s dátami uloženými v databáze je v aplikácii riešená v dvoch krokoch. Po kliknutí na tlačidlo Working with DB respektíve zvolením tejto možnosti z menu v pravom hornom rohu sa zobrazí stránka so zoznamom všetkých vytvorených tabuliek v databáze a pod ním sa zobrazí prvých 15 záznamov v danej tabuľke, aby mal užívateľ aspoň približný prehľad, aké dáta sa v tabuľke nachádzajú. Kliknutím na tlačidlo Select sa zadaná možnosť potvrdí (viď Obr. 4.10).



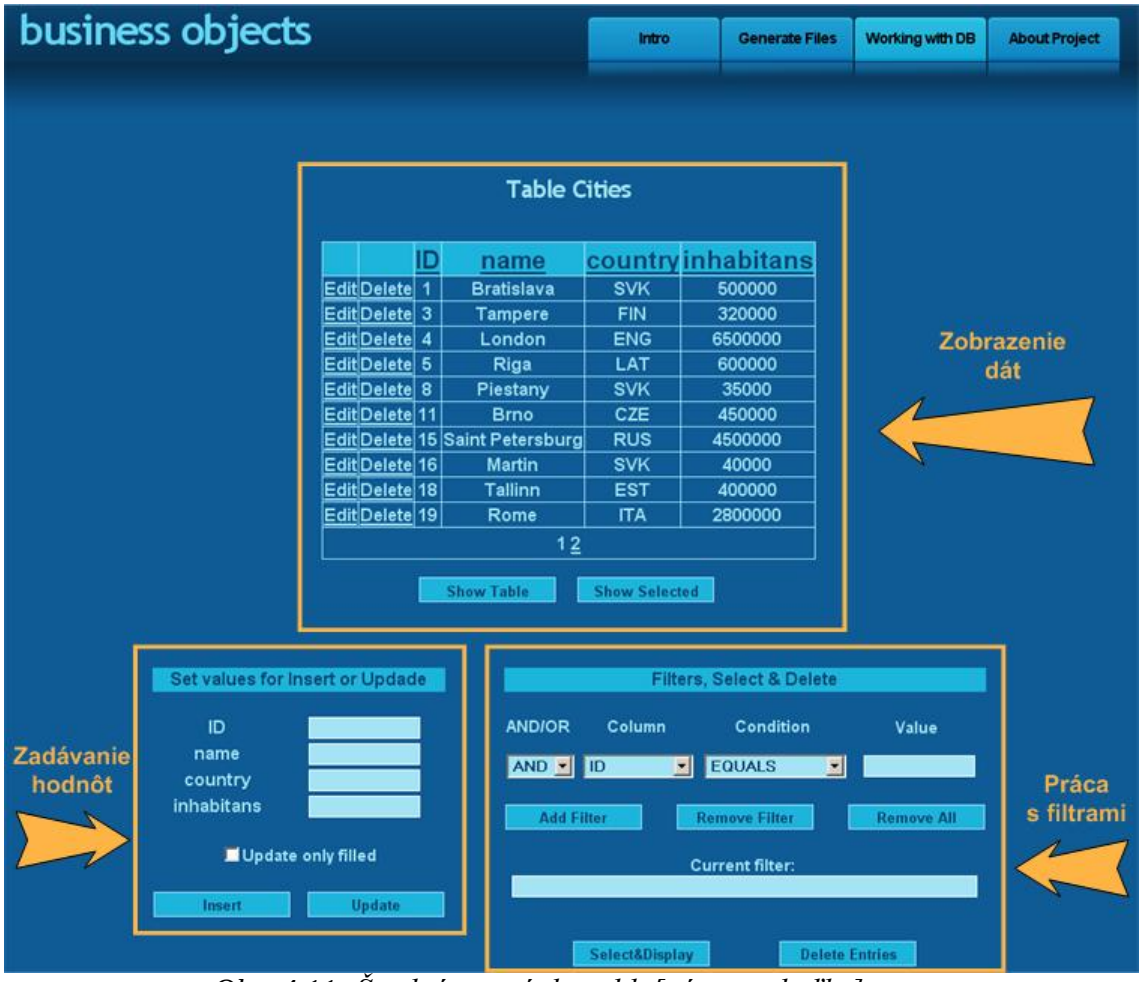
Obr. 4.10: Výber tabuľky s ktorou chce užívateľ pracovať

Užívateľ je následne presmerovaný na stránku tableSelect.aspx (viď Kap. 4.2.4 sekcia c). Ak boli súbory potrebné na prácu s dátami už vygenerované, v SESSION sa vytvorí objekt triedy odpovedajúci vybranej tabuľke a užívateľ je presmerovaný na stránku table[názov_tabuľky].aspx. Ak nie, aplikácia vytvorí chybové hlásenie, kde je užívateľ upozornený, že tieto súbory musia byť vygenerované. Štruktúra stránky table[názov_tabuľky].aspx je zobrazená na obrázku 4.11, ako príklad je použitá tabuľka Cities. Stránka sa skladá z troch hlavných častí:

- *zobrazenie dát* – zdrojom dát sú dva ObjectDataSource objekty – prvý obsahuje všetky dáta v tabuľke, druhý obsahuje dáta vybrané na základe zadaných filtrovacích podmienok. Týmito dátami je naplnený objekt GridView, ktorý umožňuje aktualizovať a mazať jednotlivé záznamy. Zdroje dát sa dajú meniť kliknutím na príslušné tlačidlo. GridView taktiež umožňuje listovanie záznamov a ich zoradenie podľa údajov v jednotlivých stĺpcoch tabuľky. Detailnejší popis funkčnosti jednotlivých ovládacích prvkov je popísaný nižšie.
- *zadávanie hodnôt* – obsahuje TextBox pre každý stĺpec v tabuľke, do ktorého užívateľ môže zadať hodnoty a tie následne pomocou tlačidla Insert vložiť do

tabuľky alebo pomocou tlačidla Update aktualizovať všetky záznamy, ktoré vyhovujú zadanému filtru. Pri aktualizovaní údajov je taktiež možné zaškrtnúť možnosť Update only filled a vtedy sa aktualizujú len tie hodnoty, ktoré sú v TextBoxoch vyplnené.

- *práca s filtrami* – rieši pridávanie a odstraňovanie filtrov, kliknutím na tlačidlo Select&Display sa v GridView zobrazia záznamy vyhovujúce zadaným podmienkam, tlačidlo Delete Entries slúži na zmazanie dát na základe zadaných filtrovacích podmienok.



Obr. 4.11: Štruktúra stránky table[názov_tabuľky].aspx

4.2.6 Práca s tabuľkou

a) Zdroje dát v GridView

Ako už bolo spomenuté, dáta z tabuľky sa zobrazujú v GridView, ktorý ako zdroj dát implicitne používa ObjectDataSource1. Ten slúži na zobrazenie všetkých dát v databáze (ako je to vidieť aj na obrázku 4.11), výber dát prebieha volaním metódy select(), avšak

v tomto prípade metóda neberie do úvahy prípadné zadané filtrovacie podmienky. Nastavenie ObjectDataSource1 ako zdroja dát je možné aj kliknutím na tlačidlo Show Table.

Druhou možnosťou zdroja dát pre GridView je ObjectDataSource2. Štruktúru má rovnakú ako predchádzajúci, rozdiel je v tom, že vyberie len tie dáta z tabuľky, ktoré vyhovujú zadaným filtrovacím podmienkam (viď obr. 4.14). Zobrazenie týchto dát je možné kliknutím na tlačidlo Show Selected respektíve Select&Display.

b) Editovanie záznamov v GridView

GridView priamo podporuje editovanie záznamov nastavením atribútu ShowEditButton na hodnotu True. GridView zavolá metódu, ktorú má ObjectDataSource definovanú v atribúte UpdateMethod. Po kliknutí na tlačidlo Edit pri zázname, ktorý má byť editovaný, sa otvorí okno na editovanie zvoleného záznamu, ako je to vidieť na obrázku 4.12.

Obr. 4.12: Editovanie zvoleného záznamu

Editovanie prebehne po kliknutí na tlačidlo Update zavolaním metódy updateFromDict(). Hodnoty v riadku sa nastavia ako atribúty triedy typu GBO-Cities. Aplikácia analyzuje štruktúru tabuľky a SQL príkaz Update sa prevedie len v stĺpcoch,

ktoré nie sú udané ako primárne kľúče tabuľky a zároveň nemajú nastavenú vlastnosť Identity (automatické číslovanie záznamov pri ich vkladaní).

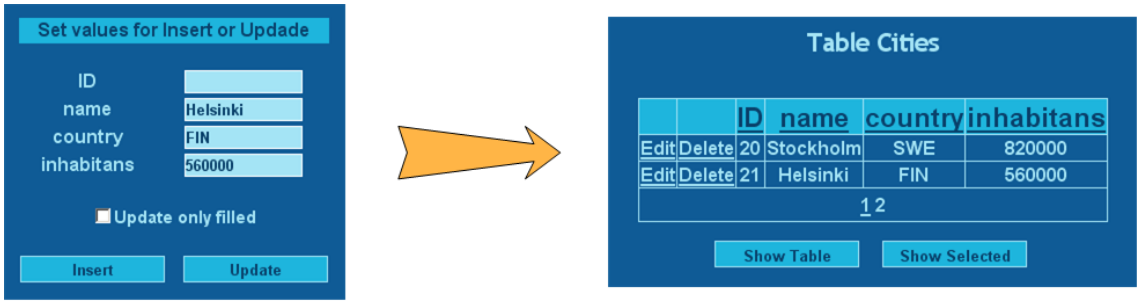
c) Mazanie záznamov z GridView

Rovnakým spôsobom, ako je podporované editovanie údajov v GridView, je podporované aj ich mazanie. Atribúd GridView ShowDeleteButton je nastavený na hodnotu True a kliknutím na tlačidlo Delete sa volá metóda definovaná v ObjectDataSource v atribúte DeleteMethod. Mazanie vybraného záznamu z tabuľky vykonáva metóda deleteFromDict(). Hodnoty uložené v tabuľke sa nastavia ako atribúty triedy GBO-Cities, a ako filtrovacia podmienka je nastavená hodnota atribútu, ktorý má rovnaký názov ako primárny kľúč tabuľky.

d) Vkladanie dát do databázy

Formulár pre zadávanie hodnôt je na stránke umiestnený v ľavom dolnom rohu – na obrázku 4.11 je oblasť označená ako „Zadávanie hodnôt“. Pre každý stĺpec v tabuľke sa na stránke vygeneruje TextBox, do ktorého užívateľ zadáva jednotlivé hodnoty údajov. Kliknutím na tlačidlo Insert sa zadané údaje nastavia ako atribúty triedy typu GBO-Cities a zavolá sa metóda insert().

Pred vykonaním SQL príkazu Insert aplikácia analyzuje štruktúru tabuľky a zistí hodnotu primárneho kľúča tabuľky a taktiež zistí, či má niektorý zo stĺpcov nastavenú hodnotu vlastnosti Identity na True. Ak niektorý zo stĺpcov má túto vlastnosť nastavenú, zadané dáta v tabuľke sa ignorujú a o ich korektné vloženie sa postará databázový engine, ako je to načrtnuté aj na obrázku 4.13. V prípade, že vlastnosť Identity nie je nastavená a stĺpec je zároveň aj primárnym kľúčom tabuľky, aplikácia skontroluje, či sa hodnota, ktorá má byť vložená do tabuľky, už v danom stĺpci vyskytuje. Ak áno, údaj pochopiteľne nemôže byť vložený a aplikácia vypíše varovné okno upozornením pre užívateľa. Ak sa tam daný údaj nenachádza alebo stĺpec nie je ani primárnym kľúčom tabuľky, ani nemá nastavenú vlastnosť Identity, dáta sa vložia do databázy, textové polia v TextBoxoch sa nastavia na prázdne reťazce a obnovia sa údaje v GridView.



Obr. 4.13: Vkladanie dát do databázy

e) Aktualizácia údajov

V prípade, ak chce užívateľ aktualizovať viac údajov naraz, editovanie záznamov v GridView mu to neumožňuje. Aktualizáciu viacerých údajov umožňuje tlačidlo Update nachádzajúce sa vpravo od tlačidla Insert. Všetky dáta, ktoré spĺňajú filtrovacie podmienky (viac informácií v odseku f)), sú aktualizované hodnotami zadanými v rovnakých TextBoxoch, ako v prípade funkcie Insert. Užívateľ ešte môže zaškrtnúť voľbu Update only filled, následkom čoho bude aplikácia ignorovať polia, v ktorých nie je zadaná žiadna hodnota. To môže byť užitočné najmä v prípadoch, ak majú byť aktualizované údaje iba v jednom stĺpci tabuľky pre viacero záznamov a teda záznamy vo zvyšných stĺpcoch zostanú nezmenené.

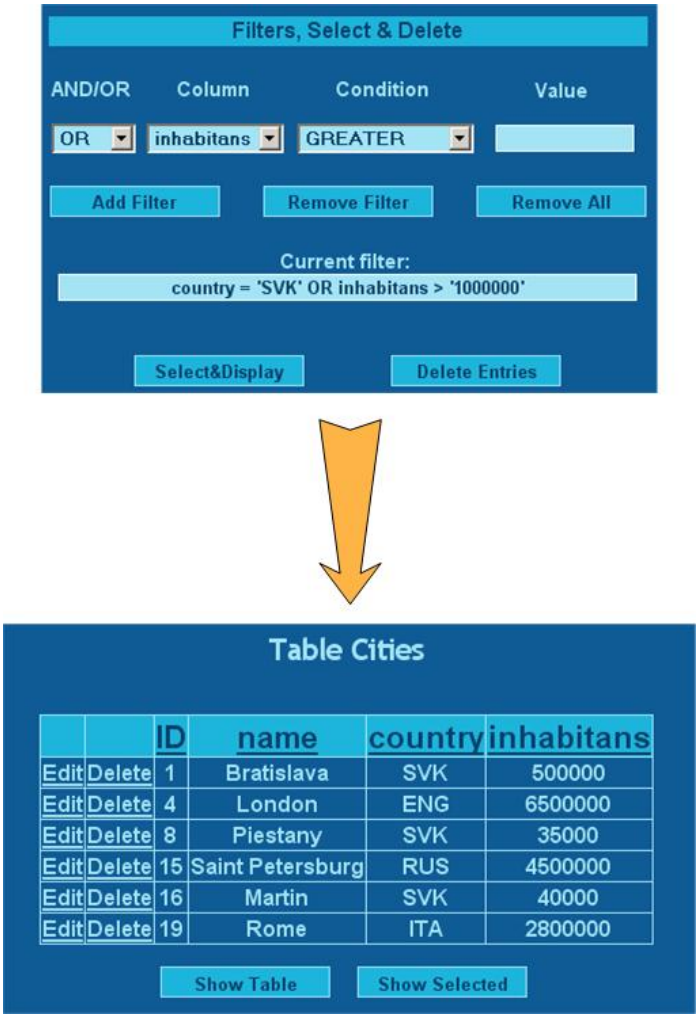
Po kliknutí na tlačidlo Update sa údaje zadané v TextBoxoch nastavia ako hodnoty atribútov triedy GBO-[názov_tabuľky]. Ak užívateľ zaškrtol voľbu Update filled only, zavolá sa metóda updateOnlyFilled(), v opačnom prípade sa zavolá metóda update(). Aplikácia analyzuje štruktúru tabuľky, zistí hodnotu primárneho kľúča a stĺpcov s hodnotou Identity. Záznamy pre stĺpec, ktorý je primárny kľúč, sa ignorujú. Metóda update() aktualizuje všetky zvyšné stĺpce, metóda updateFilledOnly() aktualizuje len tie údaje, ktorých hodnota atribútu príslušného stĺpca nie je rovná hodnote null.

f) Práca s filtrovacími podmienkami

Filtre sú veľmi dôležitou súčasťou väčšiny SQL príkazov. S ich pomocou je možné špecifikovať len určitú skupinu záznamov z tabuľky, pre ktoré sa má daný SQL príkaz vykonať. Oblasť webového rozhrania, kde užívateľ s týmito filtrami pracuje, je na obrázku 4.11 označená ako „Práca s filtrami“ a nachádza sa v ľavom dolnom rohu..

Prácu s filtrami obsluhujú tri prvky DropDownList, jeden prvok TextBox a tri tlačidlá:

- *DropDownList AND/OR* – slúži na zadanie SQL operátora, ktorý určuje, akým spôsobom bude filtrovacia podmienka vyhodnocovaná. AND je výraz pre logický súčin, OR pre logický súčet.
- *DropDownList Column* – obsah tohto prvku sa generuje automaticky na základe názvov jednotlivých stĺpcov v tabuľke.
- *DropDownList Condition* – určuje podmienku filtrovania, EQUALS značí podmienku rovnosti, NOT_EQUALS podmienku nerovnosti, GREATER značí podmienku „väčší ako“, LESS podmienku „menší ako“ a podmienka LIKE umožňuje nájsť všetky záznamy, ktoré obsahujú zadaný reťazec znakov (nemusia sa mu presne rovnať ako v prípade EQUALS).
- *TextBox Value* – slúži na zadanie konkrétnej hodnoty, ktorá má byť použitá pri filtrovaní záznamov.
- *tlačidlo Add Filter* – zavolá metódu addFilter, kde ako vstupné parametre použije údaje zadané v spomenutých prvkoch. Filter sa uloží ako textový reťazec atribútu filter(dátový typ ArrayList) triedy GBO-[názov_tabuľky].
- *tlačidloRemove Filter* – odstraňuje filter zavolaním metódy removeFilter. V prípade, že chce užívateľ odstrániť prvú filtrovaciu podmienku, aplikácia ignoruje zadanú hodnotu v prvku AND/OR, keďže táto hodnota je nepodstatná (pred prvou filtrovacou podmienkou nikdy nie je SQL operátor AND alebo OR) a nezobrazuje sa ani v TextBoxe CurrentFilter.
- *tlačidlo Remove All* – odstráni všetky zadané filtrovacie podmienky zavolaním funkcie removeAllFilters().
- *TextBox Current Filter* – zobrazuje zadané podmienky ako textový reťazec. Vlastnosť TextBoxu ReadOnly je nastavená na hodnotu True, to znamená, že užívateľ nemôže zobrazený reťazec meniť.



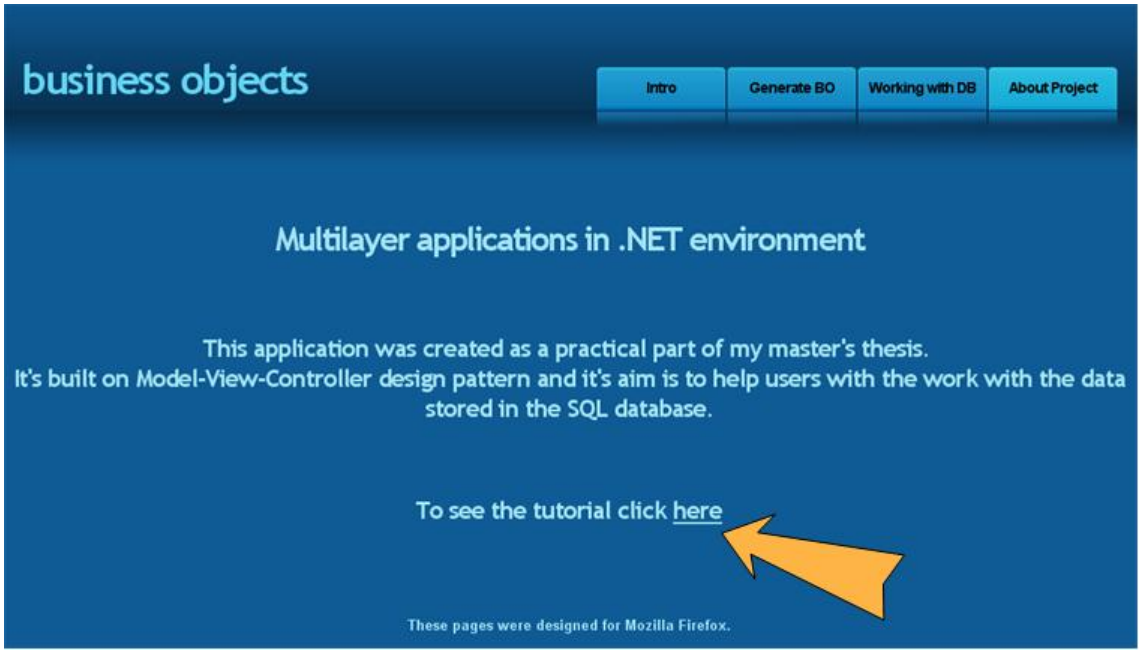
Obr. 4.14: Použitie filtrovacích podmienok

g) Mazanie záznamov

Kliknutie na tlačidlo Delete Entries umožňuje užívateľovi zmazať z tabuľky všetky záznamy, ktoré vyhovujú zadaným filtrovacím podmienkam, zavolaním metódy delete().

4.2.7 Informácie o aplikácii

Kliknutím na tlačidlo About Project v hlavnom menu v pravom hornom rohu webových formulárov, sa zobrazí formulár about.aspx. Tento formulár je len informačný a obsahuje len základné informácie o aplikácii. Kliknutím na odkaz (viď Obr. 4.15) sa zobrazí elektronická verzia tejto práce vo formáte PDF otvorená na strane, na ktorej sa v práci začína pojednávať o aplikácii.



Obr. 4.15: Informačný webový formulár

4.3 VÝHODY APLIKÁCIE

V predchádzajúcich kapitolách bola popísaná funkčnosť aplikácie, táto kapitola je venovaná jej výhodám so zameraním sa na uľahčenie práce užívateľom.

4.3.1 Práca s dátami

Štandardný postup

S dátami v databáze sa pracuje pomocou SQL príkazov. Tieto SQL príkazy majú presnú syntaktickú štruktúru, ktorá ak nie je dodržaná, SQL príkaz sa neprevedie a server vypíše chybové hlásenie. Písanie týchto príkazov je pomerne časovo náročné, najmä v prípade hľadania eventuálneho preklepu v dlhšom SQL príkaze. SQL server môže mať problémy aj s vykonaním príkazov, ktoré síce sú syntakticky správne napísané, no sémanticky sú chybné – napríklad ak užívateľ nezadá hodnotu primárneho kľúča tabuľky, prípadne chce vložiť hodnotu primárneho kľúča, ktorá sa už v tabuľke nachádza.

V prípade, ak by užívateľ chcel napríklad vložiť nový záznam do tabuľky Cities, SQL príkaz by musel vyzerať nasledovne:

```
INSERT INTO Cities (ID, name, country, inhabitans) VALUES ('22', 'Prague', 'CZE', '2200000')
```

Vykonávanie SQL príkazov pomocou aplikácie

Aplikácia pracuje s dátami v databáze pomocou členských premenných triedy, ktorá bola pre danú tabuľku vygenerovaná. Názvy jednotlivých atribútov generovaných tried sú odvodené z názvov stĺpcov v tabuľke a ich dátové typy sú premapované z príslušných SQL dátových typov. Následne sú v atribúte dictColumns, ktorá je typu Dictionary, pridané hodnoty kľúčov, ktoré taktiež odpovedajú stĺpcom tabuľky. Ak bude chcieť užívateľ vykonať nejakú prácu s dátami, potrebné hodnoty sa nastavia ako hodnoty k jednotlivým kľúčom. Aplikácia potom pracuje už len s týmito hodnotami.

SQL príkazy sú automaticky generované aplikáciou, to znamená, že užívateľ nemusí nič ručne písať a tým pádom odpadá možnosť preklepov vyúsťujúcich do syntaktických chýb. Aplikácia taktiež analyzuje zadané hodnoty a vypíše varovné hlásenie v nasledujúcich prípadoch:

- ak chce užívateľ vkladať záznamy do tabuľky a nezadá hodnotu primárneho kľúča v prípade, že vlastnosť Identity daného stĺpca nie je nastavená na hodnotu True,
- ak chce užívateľ vkladať záznamy do tabuľky a zadá hodnotu primárneho kľúča, ktorá sa už v tabuľke nachádza,
- ak chce užívateľ vkladať alebo aktualizovať záznamy v stĺpci, ktorý má vlastnosť Identity nastavenú na hodnotu True (zobrazí sa len informačné okno a záznam pre daný stĺpec nie je pridaný do SQL príkazu),
- ak chce užívateľ aktualizovať záznamy v stĺpci, ktorý je nastavený ako primárny kľúč tabuľky,
- ak chce užívateľ mazať alebo aktualizovať dáta, ale nezadal žiadnu filtrovaciu podmienku,
- ak chce užívateľ pridať filtrovaciu podmienku, ktorá už existuje,

- ak chce odobrať filtrovaciu podmienku, keď žiadna nie je zadaná.

Nasledujúci postup ukazuje, ako je rovnaký SQL príkaz na vloženie dát, ako bol použitý v predchádzajúcej podkapitole, vykonaný aplikáciou:

1.) užívateľ zadá údaje do príslušných TextBoxov a klikne na tlačidlo Insert (viď Obr. 4.11).

2.) vytvorí sa nová inštancia triedy GBO_Cities:

```
GBO_Cities myObject = new GBO_Cities("Cities");
```

Názvy atribútov, ako aj dátové typy, sú odvodené z tabuľky, s ktorou chce užívateľ pracovať.

3.) údaje zadané v TextBoxoch sa nastavia ako hodnoty atribútov triedy:

```
if (tb_ID.Text != "")
{
    myObject.ID = Convert.ToInt32(tb_ID.Text);
}
if (tb_name.Text != "")
{
    myObject.name = Convert.ToString(tb_name.Text);
}
if (tb_country.Text != "")
{
    myObject.country = Convert.ToString(tb_country.Text);
}
if (tb_inhabitans.Text != "")
{
    myObject.inhabitans = Convert.ToInt32(tb_inhabitans.Text);
}
```

Tým sa automaticky nastavia aj ako hodnoty k príslušným kľúčom v atribúte dictColumns:

```
public System.Int32 ID
{
    get
    {
        return (System.Int32)this.dictColumns["ID"];
    }
    set
    {
        this.dictColumns["ID"] = value;
```

```
            _ID = value;
        }
    }

    public System.String name
    {
        get
        {
            return (System.String)this.dictColumns["name"];
        }
        set
        {
            this.dictColumns["name"] = value;
            _name = value;
        }
    }
    public System.String country
    {
        get
        {
            return (System.String)this.dictColumns["country"];
        }
        set
        {
            this.dictColumns["country"] = value;
            _country = value;
        }
    }
    public System.Int32 inhabitans
    {
        get
        {
            return (System.Int32)this.dictColumns["inhabitans"];
        }
        set
        {
            this.dictColumns["inhabitans"] = value;
            _inhabitans = value;
        }
    }
```

4.) aplikácia zavolá metódu insert():

```
myObject.insert();
```

5.) vytvorí sa spojenie s databázou

```
string connectionString =
ConfigurationManager.AppSettings["constring"];
SqlConnection con;
con = new SqlConnection(connectionString);
con.Open();
```

6.) získajú sa hodnoty názvu stĺpca s vlastnosťou primárneho kľúča, stĺpca s vlastnosťou Identity a všetky hodnoty primárnych kľúčov uložených v tabuľke:

```
string primaryKey = "";
primaryKey = this.getPrimaryKey(con);
string identity = "";
identity = this.getIdentity(con);
SqlDataReader PKValues = this.getPKValues(con, primaryKey);
```

7.) pre každý záznam v dictColumns sa pomocou cyklu (`foreach (KeyValuePair<string, object> entry in this.dictColumns)`) vykonajú nasledovné operácie:

7.1) ak stĺpec nie je ani primárnym kľúčom, ani nemá nastavenú vlastnosť Identity, tak sa do premenných typu StringBuilder vložia údaje o práve spracovávanom zázname v dictColumns:

```
columnsList.Append(entry.Key);
columnsList.Append(",");
valuesList.Append("'");
valuesList.Append(entry.Value);
valuesList.Append("',");
```

7.2) ak je stĺpec primárnym kľúčom a nemá nastavenú vlastnosť Identity na True, tak aplikácia zistí, či sa záznam, ktorý chce užívateľ vložiť, už nachádza v tabuľke (ak áno, aplikácia vygeneruje chybové hlásenie a premenná execute sa nastaví na hodnotu False). Ak sa záznam ešte v tabuľke nenachádza, aplikácia otestuje, či užívateľ nechce vkladať nulovú hodnotu a ak nie, tak do premenných typu StringBuilder vloží spracovávané údaje. Ak má stĺpec nastavenú hodnotu Identity, aplikácia tento záznam ignoruje:

```
if (identity != entry.Key && primaryKey != entry.Key)
{
    columnsList.Append(entry.Key);
    columnsList.Append(",");
    valuesList.Append("'");
    valuesList.Append(entry.Value);
    valuesList.Append("',");
}
else if (identity != entry.Key && primaryKey == entry.Key)
{
    PKexists = false;
    while (PKValues.Read())
    {
        if (PKValues.GetValue(0).Equals(entry.Value))
```

```
            {
                PKexists = true;
            }
        }
        if (PKexists == true)
        {
            MessageBox.Show("The Insert command cannot be executed. The
column '" + entry.Key + "', which is the Primary Key for the table '"
+ this.tableName + "', already contains the value '" + entry.Value +
"'.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            execute = false;
        }
        else
        {
            if (entry.Value == null)
            {
                MessageBox.Show("The Insert command cannot be executed.
The primary key(column '" + primaryKey + "') for the table '" +
this.tableName + "' must be set!", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                execute = false;
            }
            else
            {
                columnsList.Append(entry.Key);
                columnsList.Append(",");
                valuesList.Append("'");
                valuesList.Append(entry.Value);
                valuesList.Append("',");
            }
        }
    }
    else if (identity == entry.Key)
    {
        if (entry.Value != null)
        {
            MessageBox.Show("Property of the column " + entry.Key + " in
the table '" + this.tableName + "' is set to AutoIncrement. The value
you have set will be replaced automatically.", "Warning",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
```

8.) ak je hodnota premennej execute True, v premenných columnsListLength a valuesListLength je odstránená čiarka za posledným záznamom a aplikácia prevedie vykonanie SQL príkazu INSERT:

```
int columnsListLength = columnsList.Length;
columnsList.Remove(columnsListLength - 1, 1);
int valuesListLength = valuesList.Length;
valuesList.Remove(valuesListLength - 1, 1);

string insertString = "INSERT INTO " + this.tableName + "(" +
columnsList + ") VALUES (" + valuesList + ")";
SqlCommand insertCmd = new SqlCommand(insertString, con);
insertCmd.ExecuteNonQuery();
```

9.) napokon sa uzavrie spojenie s databázou:

```
con.Close();
```

Ako je z uvedeného postupu vidieť, užívateľ musí vykonať len prvý krok – to znamená zadať hodnoty, ktoré sa majú do tabuľky vložiť. Samotné zostavenie SQL príkazu, kontrolu správnosti zadaných údajov a napokon aj vykonanie tohto SQL príkazu už vykoná aplikácia.

Popis ostatných možností práce s dátami v databáze je uvedený v kapitole 4.2. Princíp procesu ich vykonávania je podobný s popisom uvedeným v tejto kapitole.

4.3.2 Ďalšie výhody aplikácie

Okrem už spomínaného uľahčenia práce užívateľovi pri vytváraní SQL príkazov a kontroly ich logickej správnosti, aplikácia poskytuje aj iné výhody:

- aplikácia je nezávislá na dátovom type dát v databáze,
- dátové typy atribútov generovaných business objektov sú mapované z dátových typov odpovedajúcich stĺpcov v tabuľke, čo umožňuje užívateľom vytvoriť metódy na následnú prácu s atribútmi triedy. S dátami uloženými ako číselné typy je možné pracovať ako s číslami, s dátovými reťazcami je možné pracovať ako s dátovými reťazcami a podobne,
- princíp partial classes umožňuje hociktorú zdedenú metódu v generovaných business objektoch prepísať presne podľa potrieb užívateľa,
- aplikácia ponúka aj užívateľské rozhranie, ktoré užívateľom ešte viac prácu s databázami uľahčuje.

4.4 POUŽITÉ SOFTWAROVÉ NÁSTROJE

Pri vytváraní aplikácie Viacvrstvové aplikácie v prostredí .NET boli použité nasledujúce softwarové nástroje:

- *Visual Studio 2005 + Service Pack 1* – vývojové prostredie,

- *SQL Server 2005* – databázový server,
- *SQL-Server-Management-Studio-Express* – program slúžiaci na správu SQL databáz.

5 ZÁVER

Hlavným cieľom tejto práce bolo popísať technológie využité pri vývoji praktickej časti diplomovej práce, teda aplikácie vyvinutej na princípoch architektúry Model-View-Controller v prostredí .NET Framework. Ako prvý bol podrobne popísaný návrhový model aplikácie Model-View-Controller, ktorý umožňuje rozdeliť aplikáciu na tri navzájom nezávislé vrstvy. Modifikácia zdrojového kódu v jednej vrstve by nemala vôbec ovplyvniť ostatné vrstvy. Tento návrhový vzor taktiež umožňuje vytváranie nových rozhraní bez nutnosti modifikácie aplikácie. Cieľom práce bolo popísať činnosť jednotlivých vrstiev architektúry, popísať princíp fungovania konceptu MVC na najbežnejšom príklade použitia tejto architektúry v praxi – na modeli prístupu k databáze pomocou webového rozhrania. Časť kapitoly bola taktiež venovaná výhodám a nevýhodám architektúry a jej implementácii v praxi.

Ďalšia téma, ktorej sa táto práca venuje, je platforma .NET Framework od firmy Microsoft určená pre operačné systémy Windows. Poskytuje rozhranie pre vývoj aplikácií založených na princípoch objektovo-orientovaného programovania. Jej hlavnými časťami sú jazykovo neutrálna knižnica Framework Class Library obsahujúca kolekciu zdrojových kódov podporujúcich najbežnejšie programátorské problémy a rozhranie na spúšťanie vytvorených aplikácií pomocou Common Language Runtime. Cieľom kapitoly bolo popísať architektúru .NET Frameworku, vysvetliť činnosť jednotlivých častí prostredia, poskytnúť prehľad doteraz vydaných verzií .NET Frameworku. Ďalšia časť kapitoly je venovaná jazyku C#, ktorý bol firmou Microsoft vyvinutý zároveň s platformou .NET. Záver kapitoly je venovaný technológii ADO.NET, ktorá je v diplomovej práci taktiež využtá. Jedná sa o technológiu na prístup k dátam uloženým v databáze a ich následnú modifikáciu.

Posledná kapitola práce popisuje samotný návrh riešenia praktickej časti diplomovej práce – aplikáciu „Viacvrstvové aplikácie v prostredí .NET Framework“. Základom vytvorenej aplikácie je dynamická knižnica napísaná v jazyku C#, ktorá tvorí akýsi obecný business objekt. Tento objekt generuje business objekty nad tabuľkami databázy, ktoré si užívateľ zvolí. Objekty implementujú metódy na prácu s databázou (insert, update, select, delete), ktoré automaticky generujú SQL príkazy a to znamená,

že užívateľ môže príslušné príkazy vykonávať len kliknutiami myši. Dátové typy atribútov generovaných business objektov sú z SQL dátových typov premapované na C# dátové typy. Užívateľské rozhranie, tvoriace vrstvu View, je vytvorené v ASP.NET. Na vývoj aplikácie bola použitá platforma Microsoft .NET Framework a vývojové prostredie Microsoft Visual Studio 2005. Aplikácia je navrhnutá pre databázový server Microsoft SQL Server 2005. Funkčnosť aplikácie je nezávislá na type dát v databáze.

LITERATÚRA

[1] ASP.NET QuickStart Príručka. [online]. 2007 [cit. 2008-04-21]. Dostupné na internete: <*http://quickstart.aspnet.sk/QUICKSTARTV20/aspnet/doc/pages/pages.aspx*>.

[2] EVJEN B. *C# 2005 Programujeme profesionálně*, Computer Press 2007, ISBN 80-251-1181-4

[3] Microsoft .Net Framework Framework Tutorials [online]. 2005 [cit. 2008-12-06]. Dostupné na internete: <*http://vb.net-informations.com/framework/framework_tutorials.htm*>.

[4] Model-View-Controller [online]. 2008 [cit. 2008-12-01]. Dostupné na internete: <*http://msdn.microsoft.com/en-us/library/ms978748.aspx*>.

[5] Model View Controller [online]. 2007 [cit. 2008-12-01]. Dostupné na internete: <*http://www.phpwact.org/pattern/model_view_controller*>.

[6] ObjectDataSource Web Server Control Overview [online]. 2008 [cit. 2008-12-08]. Dostupné na internete: <*http://msdn.microsoft.com/en-us/library/9a4kyhcx.aspx*>.

[7] Overview of ADO.NET [online]. 2008 [cit. 2008-12-07]. Dostupné na internete: <*http://msdn.microsoft.com/en-us/library/h43ks021(VS.71).aspx*>.

[8] Overview of the .NET Framework [online]. 2008 [cit. 2008-12-05]. Dostupné na internete: <*http://msdn.microsoft.com/en-us/library/a4t23ktk.aspx*>.

[9] RYDVAL, Slávek. Seznámení s .NET [online]. 2005 [cit. 2008-12-05]. Dostupné na internete: <*http://www.rydval.cz/phprs/view.php?cisloclanku=2005061313*>.

[10] TROELSEN, A. *C# a .NET 2.0 profesionálně*, ZONER Press 2007, ISBN 80-86815-42-0

ZOZNAM SKRATIEK

ADO.NET – ActiveX Data Object

ASP.NET – Active Server Pages

BCL – Base Class Library

CLI – Common Language Infrastructure

CLR – Common Language Runtime

CLS – Common Language Specification

COM – Component Object Model

CTS – Common Type System

DLL – Dynamic-Link Library

FCL – Framework Class Library

GAC – Global Assembly Cache

GC – Garbage Collecting

GUI – Graphical User Interface

HTML – HyperText Markup Language

LINQ – Language Integrated Query

MFC – Microsoft Foundation Classes

MS-DOS – Microsoft Disk Operating System

MSIL – Microsoft Intermediate Language

MVC – Model-View-Controller

ODBC – Open Database Connectivity

OLE DB – Object Linking and Embedding Database

OOP – Object Oriented Programming

PDA – Personal Digital Assistant

PDF – Portable Document Format

PHP – Personal Home Page

SQL – Structured Query Language

TCP/IP – Transmission Control Protocol/Internet Protocol

UML – Unified Modeling Language

VB.NET – Visual Basic

WCF – Windows Communication Foundation

WF – Windows Workflow Foundation

WML – Wireless Markup Language

WPF – Windows Presentation Foundation

XML – eXtended Markup Language

OBSAH CD

- elektronická verzia diplomovej práce vo formáte PDF
- adresár BusinessObjects so zdrojovými kódmi aplikácie