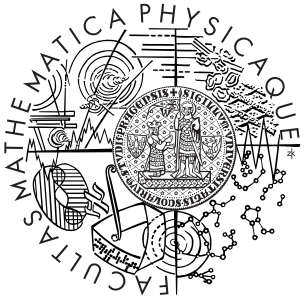
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Tomáš Bartoš

Informační systém pro obchodní firmu

Katedra aplikované matematiky

Vedoucí bakalářské práce: RNDr. Tomáš Tichý
Studijní program: Informatika, programování

2007

Na tomto mieste by som chcel poďakovať pánovi Ing. Ladislavovi Bartošovi za všetky odborné rady, ktoré mi poskytol, za čas strávený konzultáciami riešených problémov a za poskytnuté materiály, pánovi RNDr. Tomášovi Tichému za vedenie mojej práce a firme Team - SOFT s.r.o. za poskytnutie technických a technologických prostriedkov na realizáciu projektu.

Prohlašuji, že jsem svou bakalářskou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 28.5.2007 Tomáš Bartoš

Obsah

Název práce: *Informační systém pro obchodní firmu*
Autor: *Tomáš Bartoš*
Katedra (ústav): *Katedra aplikované matematiky*
Vedoucí bakalářské práce: *RNDr. Tomáš Tichý*
e-mail vedoucího: *tichy@math.cas.cz*

Abstrakt: Cieľom práce je implementácia informačného systému založeného na architektúre *klient-server*, ktorý bude poskytovať skladového hospodárstvo určené pre malé a stredné obchodné spoločnosti. Systém je navrhnutý tak, aby zodpovedal požiadavkám z praxe a poskytol dve používané metódy oceňovania skladu - oceňovanie váženým priemerom a oceňovanie *FIFO* (z angl. *first-in-first-out*). Dôraz je kladený na grafické užívateľské prostredie a jednoduchosť ovládania na strane klienta a na efektívnosť, dodržiavanie normálnych foriem a zabezpečenie integrity dát na strane servera.
Projekt je napísaný v jazyku *C#* s podporou *.NET Framework 2.0*.

Klíčová slova: informačný systém, skladové hospodárstvo, klient-server, .NET

Title: *Bussiness information system*
Author: *Tomáš Bartoš*
Department: *Department of Applied Mathematics*
Supervisor: *RNDr. Tomáš Tichý*
Supervisor's e-mail address: *tichy@math.cas.cz*

The aim of bachelor thesis is to implement information system based on *client-server* architecture, which offers storehouse management designed for small and medium bussiness companies. System is designed to meet the requests from practice and to offer two common used principles of calculation of cost price - method of weighted mean and FIFO method (*first-in-first-out*). The accent is put on graphical user interface and handling simplicity on client side and efficiency, respecting normal forms and ensuring data integrity on server side.
Project is written in *C#* based on *.NET Framework 2.0*.

Keywords: information system, storehouse management, client-server, .NET

Kapitola 1

Úvod

Informačné systémy (IS) v podnikoch sú základným prvkom pri spracovaní a uchovaní rozličných dát. Rôzne systémy sa od seba odlišujú v závislosti od spôsobu využitia. Užívateľ má väčšinou jednoduchú predstavu, ako má taký systém fungovať. Postupne bude zadávať nejaké údaje, opravovať ich a ukladať, analyzovať, niekedy ich bude chcieť spätne získať alebo úplne vymazať. Požaduje však, aby tieto údaje boli konzistentné a korektné. Väčšinou však nemá prehľad o tom, koľko vecí je potrebných k tomu, aby mu systém vyhovel vo všetkých oblastiach. Preto je dôležité spraviť dôkladnú analýzu problému a požiadaviek ešte pred začiatkom samotného navrhovania systému.

Vplyv na vývoj programu má aj rozhodnutie o cieľovej skupine užívateľov. Iný charakter bude mať program, ktorý bude jednoužívateľský, a iný charakter bude mať systém, v ktorom môže pracovať viacej užívateľov súčasne. V multiužívateľskom režime treba zabezpečiť konzistenciu dát tak, aby si rôzni užívatelia navzájom nemohli prepisovať údaje bez toho, aby o tom vedeli.

Nemenej dôležitá je aj voľba užívateľského prostredia. Neskúsení užívatelia kladú väčší dôraz na kvalitnejšie grafické prostredie a jednoduchosť ovládania programu, kým skúsenejší uprednostňujú skôr funkcionalitu a efektívnosť programu vzhľadom k práci, ktorú vykonávajú.

Mojou snahou bolo navrhnúť užívateľsky prívetivé prostredie, skombinovať ho s efektívnosťou a vytvoriť informačný systém, ktorý bude vyhovovať užívateľom začiatočníkom aj pokročilím. Zameral som sa predovšetkým na základnú časť obchodovania, ktorou je *skladové hospodárstvo* - príjem a výdaj tovaru zo skladu, rôzne spôsoby oceňovania vydaného sortimentu[1]...

Okrem vytvárania dokladov k základným skladovým pohybom akými sú príjem a výdaj sortimentu zo skladu, program umožňuje aj vystavovanie ponúk, objednávok zákazníkom, či evidenciu niektorých prijatých dokladov. Užívateľský komfort dopĺňajú možnosti evidencie obchodných partne-

[1] sortiment môže označovať nielen tovar, ale aj materiál, služby, popr. vlastné výrobky

rov, kontaktných osôb, miest a obcí, štátov a iné. Jedným z kritérií bolo vytvorenie takého prostredia, aby sa užívateľ mohol plne sústrediť na prácu a nemusel skúmať funkcionalitu programu. Samozrejmosťou bolo ovládanie programu myšou aj klávesnicou.

Celý systém je postavený na architektúre *klient-server* a navrhnutý pre operačný systém *Windows XP*. V aktuálnej verzií je možné používať na uloženie údajov databázu *SQL Server 2005*, ale v budúcnosti sa počíta aj s podporou ďalších bežne používaných databáz akými sú napríklad *Oracle, PostgreSQL, MySQL atď.*

Vývoj projektu prebiehal pomocou integrovaného vývojového prostredia *Visual Studio 2005* a *Visual C# Express Edition.*

Projekt bol navrhnutý a vyvíjaný tak, aby dodržiaval zákony platné na území Slovenskej republiky. Preto je možné, že v iných krajinách (napr. v ČR) môže byť legislatíva v niektorých oblastiach odlišná.

Kapitola 2

Sklad a skladové hospodárstvo

2.1 Charakteristika

Zdroj príjmov obchodných spoločností tvorí väčšinou poskytovanie služieb a obchodovanie s tovarom, materiálmi, vlastnými výrobkami. Sortiment, s ktorým chce firma obchodovať, si musí najskôr zaobstarať a vo väčšine prípadov nie je okamžite predaný zákazníkom, poprípade je predaná iba časť a zvyšok sortimentu skončí na *sklade*.

Presná definícia, čo je to *sklad*, je v tomto smere obtiažná, pretože závisí od uhla pohľadu. Intuitívna predstava skladu je, že ide o nejakú miestnosť alebo budovu, v ktorej sú uložené zásoby sortimentu. Z pohľadu účtovníctva *sklad* môže pokrývať viacero takýchto budov, miestností (viacero "virtuálnych" skladov[1]), ktoré majú spoločné vlastnosti.

Obchodná spoločnosť môže mať k dispozícií viacero skladov. Podľa zákona je potrebné viesť samostatnú evidenciu pre každý sklad. Ide o uchovávanie skladových dokladov (príjemka, výdajka) a skladových kariet[2] sortimentu. Práve skladové doklady sú neskôr potrebné v účtovníctve v procese účtovania.

Evidencia dokladov (nielen skladových) prebieha vo zvolenej mene, ktorá je v účtovnom období nemenná a považovaná za základnú. Znamená to, že ak firma napríklad prijala na sklad tovar a príjemka je vystavená v inej mene ako je základná mena, je potrebné prepočítať nákupné ceny na základnú menu podľa kurzu, ktorý je platný v deň uskutočnenia účtovného prípadu[3].

Existuje aj obmedzenie týkajúce sa položiek dokladov. Každý sortiment má určenú základnú mernú jednotku (MJ), v ktorej sa vedie napr. skladová karta. Preto je nutné prepočítať množstvo na základnú MJ, ak bol sortiment prijatý (vydaný) v odlišnej MJ.

[1] V ďalšom texte sa budem pridržiavať intuitívnej predstavy a za sklad budem považovať práve "virtuálny" sklad. *Sklady* budú predstavovať množinu virtuálnych skladov.

[2] viď strana 12

[3] viď [7] §24 odsek 2

Medzi ďalšie požiadavky, ktoré sa v praxi kladú na informačný systém zabezpečujúci prácu so skladom, patrí možnosť inventúry skladu k danému dátumu (aj spätne), zobrazenie informácií o objednanom a nedodanom tovare, stav zásob na sklade... Takéto nároky patria k štandardným a doplnené sú špecifickými prianiami konkrétnych používateľov systému.

Skladové hospodárstvo definuje spôsob, ako sa bude zaobchádzať so sortimentom na sklade. Každá firma si tento spôsob upravuje podľa vlastných kritérií a sama si určuje, koľko zásob chce mať aktuálne na danom sklade, aké je minimálne množstvo tovaru na sklade, aký spôsob určenia ceny sortimentu sa použije pri výdaji zo skladu a iné. Tieto záležitosti sú zväčša závislé od konkrétnej spoločnosti a teda nie je možné ich unifikovať. Napriek tomu existujú určité zákonné opatrenia, ktoré túto benevolentnosť regulujú a tie je potrebné dodržiavať. Medzi takéto nariadenia patrí metóda oceňovania výdaju sortimentu zo skladu, ktorá nesmie byť ľubovoľná. Je potrebné si vybrať z vopred určených, zákonom stanovených, možností takú metódu, ktorá bude v danom prípade pre podnik najvýhodnejšia.

2.2 Metódy oceňovania

Existujú viaceré spôsoby, ako sa určuje cena sortimentu pri výdaji zo skladu[4]. V slovenskej legislatíve[5] je možné používať dva druhy oceňovania skladu:

- oceňovanie cenou zistenou váženým priemerom
- oceňovanie spôsobom *FIFO* (z angl. *first-in-first-out*)

Metóda oceňovania je viazaná na sklad, ale firma môže viesť v systéme evidenciu viacerých skladov s rozdielnymi spôsobmi oceňovania. Metódu výpočtu ceny pri výdaji je potrebné určiť pri založení skladu a nemôže sa zmeniť, ak už existujú nejaké doklady naviazané na daný sklad.

Kým príjem sortimentu na sklad nezávisí od zvolenej metódy oceňovania a v podstate zostáva jeho charakter v obidvoch prípadoch nemenný, na výdaj tovaru má oceňovanie výrazný vplyv.

2.2.1 Oceňovanie skladu váženým priemerom

Po zvolení tejto metódy sa pri výdaji tovaru vypočíta skladová cena ako vážený aritmetický priemer. Tento priemer si môžeme predstaviť ako podiel:

$$SC = \frac{stav\ skladovej\ karty\ v\ mene}{stav\ skladovej\ karty\ v\ MJ}$$

[4]tento proces sa tiež nazýva *oceňovanie* a výsledná cena sa nazýva *skladová cena*
[5]viď [7] §24 odsek 3

kde skladová karta je jednoznačne určená tovarom a skladom, z ktorého sa výdaj uskutoční, *SC* je výsledná (skladová) cena tovaru a mena je základná mena, v ktorej sa systém vedie. Akékoľvek navýšenie skladových zásob ovplyvní aktuálnu skladovú cenu.

2.2.2 Oceňovanie skladu metódou FIFO

Podľa zákona[6] je skladová cena v tejto metóde určená nasledovne:

> *Prvá cena na ocenenie prírastku príslušného druhu majetku sa použije ako prvá cena na ocenenie úbytku tohto majetku.*

Znamená to, že výdaj zo skladu zásob sa oceňuje vždy cenou najstarších zásob postupne smerom k najnovším. V tomto prípade príjem sortimentu tiež ovplyvní skladovú cenu, ale až po určitom čase, keď sa na sklade minie sortiment prijatý skôr než nastal tento príjem.

V praxi sa používajú dva rôzne prístupy vedenia skladu s týmto druhom oceňovania. Prvý spôsob spočíva v tom, že pre každú nákupnú cenu je vytvorená nová skladová karta. Ja som si vybral druhý spôsob, ktorý je založený na tom, že program uchováva na skladovej karte nákupné ceny aj zostatky v merných jednotkách (pre danú cenu), čo znamená, že pri výdaji sa vyhľadajú a použijú sa tie položky, kde je kladný zostatok. Preto môže v skutočnosti vzniknúť viac položiek výdajky než užívateľ v skutočnosti zadal[7]. V prvom prípade by bolo nutné zadať explicitne viacej položiek, čo môže byť pre užívateľa nepohodlné.

2.3 Skladové doklady

Na evidenciu pohybov na sklade slúžia skladové doklady - *príjemka* a *výdajka*. Kedykoľvek nastane príjem alebo výdaj sortimentu, poprípade medziskladový presun (všeobecne skladový pohyb), treba túto akciu zaevidovať.

Skladové doklady môžu existovať samostatne, ale často sú naviazané na iné doklady (dodacie listy, faktúry, pokladničné doklady). Súvisí to s tým, že práve dodací list (prípadne faktúra alebo pokladničný doklad) môže byť tým impulzom pre vykonanie pohybu na sklade. Typickým príkladom je prijatie faktúry a na základe toho vytvorenie príslušnej príjemky.

Podobne ako iné doklady, aj tie skladové sa skladajú z dvoch častí - *hlavička* a *položky*. V hlavičke sú uvedené napr. detaily o dodávateľovi (odberateľovi), dátume príjmu (výdaja), kým položky obsahujú napr. prijaté (vydané) množstvo sortimentu, skladovú cenu a iné.

[6] viď [7] §24 odsek 3

[7] viď výdajka na strane 11

Keďže všeobecne sa môže v systéme vyskytovať viacero skladov, netreba zabúdať, že skladové doklady sú naviazané na sklad, kde sa tento pohyb uskutočnil. Aj keby mali rovnaké číslo[8], ešte stále sú identifikované skladom.

2.3.1 Príjemka

Tento typ dokladu sa vytvára pri prijatí sortimentu na sklad. Prirodzeným dôsledkom je navýšenie stavu zásob vybraných tovarov o príslušné hodnoty a aktualizácia skladovej karty.

Môže dôjsť aj k situácií, keď užívateľ vytvorí príjemku, zadá ju do systému, ale až spätne zistí, že spravil pri zadávaní chybu (napríklad v množstve tovaru, keď prijaté množstvo nezodpovedalo skutočnosti). Ak nie je možné položku dokladu zmeniť[9], treba napraviť túto chybu iným spôsobom, aby stav skladu bol konzistentný. Na tento účel existuje *storno príjem*.

Situácia v prípade stornovania je trošku zložitejšia, pretože treba skontrolovať, či je na danom sklade dostupné také množstvo, ktoré chce užívateľ stornovať (resp. vrátiť späť, čo znamená zníženie stavu zásob daného skladu). Takisto by mal užívateľ zadať správnu nákupnú cenu, aby bolo možné zistiť, ktorý príjem sa má stornovať. Aj v tomto prípade ale môžu nastať komplikácie, ak sa jedná o oceňovanie skladu metódou FIFO, pretože už mohlo prebehnúť vydanie všetkých tovarov s danou nákupnou cenou. Znamená to, že na sklade už nie je požadované množstvo sortimentu s danou cenou a teda storno príjem vedie k nekonzistencií dát.

2.3.2 Výdajka

Pri výdaji tovaru treba byť opatrný. Nielen pre to, že pri výdaji tovaru sa určuje skladová cena, ktorá je závislá od zvolenej metódy oceňovania, ale aj pre to, že treba skontrolovať stav zásob na sklade, či je možné vydať toľko tovaru zo skladu, ako užívateľ požaduje (podobne ako storno príjem). Navyše užívateľ nemôže explicitne zadať skladovú cenu, pretože tá je určená až pri prevedení výdaja zo skladu. Preto sa môže stať, že pri oceňovaní metódou FIFO sa môže jedna položka výdajky zmeniť na viacej položiek[10].

Podobne ako pri príjemke, aj pri výdajke existuje spôsob korigovania zle zadaných položiek dokladov formou *storno výdaja*.

Storno výdaj sa mierne odlišuje od storno príjmu, pretože je nutné okrem predajnej ceny zadať aj skladovú cenu, aby bolo možné vyhľadať položku,

[8] v praxi sa väčšinou používa rozličný spôsob číslovania pre rozdielne sklady, aby sa predišlo nedorozumeniam v účtovníctve

[9] napr. kvôli tomu, že je už zaúčtovaný alebo z iných dôvodov uzamknutý

[10] Typický príklad: Užívateľ chce vydať 20ks nejakého tovaru. Na sklade je k dispozícií 10ks s nákupnou cenou c_1 a 10ks s nákupom za c_2, kde ceny c_1 a c_2 sú navzájom odlišné. Užívateľ zadá jednu položku, ktorá sa po vytvorení výdajky rozdelí na dve.

ktorej sa stornovanie týka[11].

2.3.3 Skladová karta

Skladová karta poskytuje históriu jednotlivých pohybov pre daný sklad a sortiment. Ide o prehľad akcií, ktoré sa na zvolenom sklade vykonávali s vybraným sortimentom (postupná evidencia zásob). Tvoria ju chronologicky zoradené položky príjemok, výdajok (tiež storno príjem a storno výdaj), dodacích listov, faktúr a iných dokladov. Tieto údaje sú potrebné pri *inventarizácií*[12].

Okrem histórie môže skladová karta poskytovať aj aktuálne informácie. Ide o aktuálny stav skladu v základnej mene, aktuálny stav v základných merných jednotkách pre vybratý sortiment (potrebné napríklad v metóde oceňovania váženým priemerom), množstvo objednaného, nedodaného tovaru atď.

[11] Predpokladá sa, že ak užívateľ robí storno výdaja, má k dispozícií pôvodnú výdajku (v nejakej forme), aby mohol presne určiť skladovú cenu.

[12] proces, v ktorom účtovná jednotka overí, či stav majetku, záväzkov a rozdielu majetku a záväzkov v účtovníctve zodpovedá skutočnosti [7] § 29 odsek 1

Kapitola 3

Architektúra aplikácie

Zvolenie správnej architektúry pre aplikáciu býva jednou z najťažších častí. Treba zobrať do úvahy všetky aspekty, ktoré môžu rozhodovanie v zásadnej miere ovplyvňovať - *Kto bude koncový užívateľ? Aké sú jeho požiadavky a nároky? Kde bude aplikácia nasadená? Aké sú nároky na výkon, efektívnosť a udržiavanie chodu programu?* Toto je len náznak toho, čo sa očakáva od práce analytika, pretože systém postavený na zlých základoch je pohromou nielen užívateľov (nefunkčnosť), ale aj správcov starajúcich sa o správny chod aplikácie, či programátorov, ktorí majú za úlohu systém udržiavať a aktualizovať. Nie je preto vhodné podceňovať analýzu, aj keď jej trvanie (skúmanie dostupných riešení) zaberie veľmi veľa času. Výber *správneho modelu* môže uľahčiť prácu na viacerých úrovniach vývoja a vedenia projektu (dizajnér, programátor, testér, konzultant) a vedie k jednoduchšiemu nasadeniu[1] aplikácie u zákazníkov. Pri dodržovaní niektorých ďalších smerníc je možné opakovane dosahovať úroveň kvality a zlepšovať tak konkurencieschopnosť programu.

V dnešnej dobe poznáme niekoľko druhov architektúr od jednovrstvových až po viacvrstvové [1]. *Jednovrstvové* architektúry zodpovedajú požiadavkám nenáročných klientov, ktorí nemajú príliš veľa financií a zdrojov na vytvorenie a prevádzkovanie siete s viacerými počítačmi. Tým úplne postačuje, ak informačný systém beží na jednom počítači a pracuje na ňom niekoľko málo užívateľov (väčšinou nie súčasne). Naopak väčším spoločnostiam takýto sytém práce nemusí vyhovovať, pretože obmedzuje prácu užívateľov (nemožnosť paralelnej práce). Okrem toho pri takejto firmej sa predpokladá uchovávanie väčšieho objemu údajov, čím stúpajú nároky nielen na efektívnejšie spracovávanie dát, ale celkovo aj na robustnosť aplikácie. Preto býva vhodné nasadiť do takýchto podnikov aplikácie založené na *viacvrstvovej* architektúre.

[1] angl. *deployment*

3.1 Voľba architektúry

Na začiatku mojej práce som musel zvoliť cieľovú skupinu podnikov, pre ktoré bude aplikácia vyvíjaná, aby vykonaná analýza splňovala požiadavky, ktoré takéto firmy na skladové hospodárstvo (všeobecne informačný systém) majú. Vybral som si malé a stredné podniky. Nielen pre to, že tieto firmy sú veľmi rozšírené, ale aj pre to, že vývoj systému pre veľké spoločnosti trvá niekoľko mesiacov (až rokov) a na takomto projekte nepracuje jediný človek, ale skupina analytikov, dizajnérov, programátorov...

Ďalej som potreboval vybrať vhodnú architektúru pre program. Moje rozhodovanie ovplyvňovali v najväčšej miere dve oblasti. Na jednej strane boli možnosti, ktoré má aplikácia užívateľom ponúkať, na druhej strane stáli požiadavky projektu na zdroje, ktoré by nemali byť prehnané a prijateľné pre väčšinu odberateľov. Zvažoval som použitie jednovrstvovej architektúry alebo architektúry *klient-server* (dvojvrstvová). Obidve možnosti mali svoje kladné aj záporné stránky, no nakoniec som zvolil práve dvojvrstvovú architektúru *klient-server* a to z niekoľkých dôvodov.

Prvý dôvod mojej voľby bola požiadavka na multiužívateľský režim. Zabezpečenie paralelného prístupu k dátam som mohol vyriešiť aj v prostredí jedného počítača, na ktorom budú pracovať viacerí užívatelia. Bolo by to však náročnejšie na vývoj programu, pretože by to znamenalo starať sa okrem aplikačnej vrstvy aj o zabezpečovanie konzistencie dát. Vytvárať vlastnými prostriedkami databázovú logiku je najmä v súčasnosti dosť neefektívne, pretože existujú viaceré relačné DBMS[2], ktoré sú pre túto úlohu určené, sú efektívne aj pri práci s väčším objemom dát a dostatočne prepracované, aby splňovali požadované kritéria.

Pri presune zodpovednosti za uloženie dát na stranu servera som získal ďalšiu výhodu vo forme kontroly prístupu k dátam. Niektoré časti aplikácie by nemali byť prístupné ľubovoľnému užívateľovi a preto je potrebné definovať, kto má a kto nemá k vybraným dátam prístup. Na serveri je možné definovať prístupové práva nielen k databáze, ale aj k jednotlivým tabuľkám. Túto možnosť som využil takým spôsobom, že užívateľ sa musí pred prácou s programom prihlásiť a vtedy zistím, či má oprávnenie na prezeranie alebo zmenu údajov v databáze. Vyriešenie prístupových práv v programe, ak by boli dáta uložené na disku v nejakom súbore a pristupovali by k ním rôzni užívatelia, by bolo v systéme Windows obtiažné.

3.2 Analýza a vývoj projektu

Po zvolení cieľovej skupiny užívateľov a architektúry aplikácie som začal s hlbšou analýzou požiadaviek na informačný systém.

[2]DataBase Management System - systém riadenia bázy dát (databázový systém)

Podľa mnohých prieskumov prevažuje v sfére malých a stredných podnikov operačný systém Windows. Bežní užívatelia pracujú práve s týmto operačným systémom a to bolo dôvodom, prečo som aplikáciu začal vyvíjať pre (dnes ešte pomerne rozšírený) systém Windows XP. V budúcnosti samozrejme počítam aj s podporou Windows VISTA, ale to všetko závisí od ďalšieho využívania programu. Riešenie založené na systémoch Linux som zamietol, pretože by to znamenalo pre zákazníkov ďalšie náklady spojené s prechodom na iný operačný systém a preškolenie užívateľov (keďže iba malý počet z nich vie pracovať s iným systémom ako je spomínaný Windows). Takisto vývoj programu a jeho ladenie by sa stalo zložitejším v závislosti od distribúcie, pretože mnohé oblasti v systémoch Linux nie sú unifikované a v rozličných distribúciach fungujú niektoré veci rôzne.

Po vyriešení voľby operačného systému som začal s procesom analýzy dát. Potreboval som zistiť, čo všetko sa musí v databáze uchovávať, aké údaje budú užívatelia zadávať v konkrétnych dokladoch, ako sú tieto doklady medzi sebou previazané, ktoré z nich sú povinné, ktoré voliteľné atď. Na analýzu dát a vytvorenie príslušných tabuliek v databáze som použil nástroj *CASE*, ktorý slúži práve na modelovanie databázy, určovanie závislostí medzi tabuľkami, voľbu primárnych kľúčov. Tieto nástroje ponúkajú aj ďalšie možnosti, ktoré už sú pred užívateľom väčšinou skryté - tvorba pohľadov nad tabuľkami, vytváranie indexov, určovanie menového priestoru[3], kde sa tabuľka v databáze vytvorí, vytvorenie alternatívnych kľúčov pre tabuľku, či zobrazenie a upravovanie relácií, v ktorých sa tabuľka nachádza. Pri navrhovaní databázy som kládol dôraz na to, aby výsledný model spĺňoval 1. a 2.normálnu formu[4] [4]. Predišiel som tak nielen redundancií dát, ale mohol som preniesť kontrolu integrity dát čiastočne aj na databázový server.

Na uchovanie dát som musel zvoliť nejakú databázu. Moja voľba padla na databázu *SQL Server 2005* poprípade jej voľnú verziu *SQL Server 2005 Express Edition*. V budúcnosti možno pribudne podpora aj pre iné používané databázy.

Vypracovaním databázového modelu som získal základ pre ďalší vývoj aplikácie. V priebehu vývoja som síce spravil ešte nejaké malé zmeny tohto modelu, ale tie sa týkali vo veľkej miere doplňovaniu atribútov do tabuliek a zväčšovaniu rozsahu existujúcich atribútov. Od databázového modelu som pristúpil k navrhovaniu užívateľského prostredia. Mojou úlohou bolo vytvoriť také grafické prostredie, ktoré by bolo čo najjednoduchšie, ale zároveň by zachovalo možnosť zobraziť užívateľovi všetky požadované dáta. Tak vznikli okná pre zobrazovanie, editovanie údajov.

Po navrhnutí grafického užívateľského prostredia vznikol prototyp aplikácie, ktorá síce fungovala (užívateľ si mohol zobrazovať okná, skúšať prácu

[3]angl. *namespace*

[4]V niektorých prípadoch však bola druhá normálna forma porušená kvôli efektívnosti programu.

s nimi) ale nezobrazovala žiadne dáta. Táto *alfa* verzia mi umožnila doladiť požiadavky užívateľov na ovládanie, množstvo zobrazených informácií a iné podrobnosti týkajúce sa vzhľadu.

Keď už bolo jasné, že užívateľské prostredie je dostatočné stabilné (zmien už nebude veľa pribúdať), začal som s analýzou programového modelu systému. Potreboval som vyriešiť spôsob pripojenia k databáze, prenos údajov medzi serverom a klientom, zobrazovanie údajov na strane klienta, zabezpečenie konzistencie dát a mnoho ďalších vecí. Okrem toho som bral do úvahy možnosti rozširovania projektu v budúcnosti a systém som navrhoval tak, aby sa prípadné zmeny a doplnenia v programe dali spraviť pomerne jednoducho. Z toho všetkého vznikol tzv. programový model aplikácie.

3.3 Programový model

Po rozhodnutí zvoliť architektúru *klient-server*, upresnení operačného systému, pre ktorý budem aplikáciu vyvíjať, a vytvorení grafického užívateľského prostredia prišlo na rad jadro programu (komunikácia s databázou). Vytvoril som model, ktorý určoval, ako bude prebiehať komunikácia klienta so serverom (resp. aplikácie s databázou). Ako si môžeme pozrieť na obrázku 3.1, tento model sa skladá z viacerých častí[5].

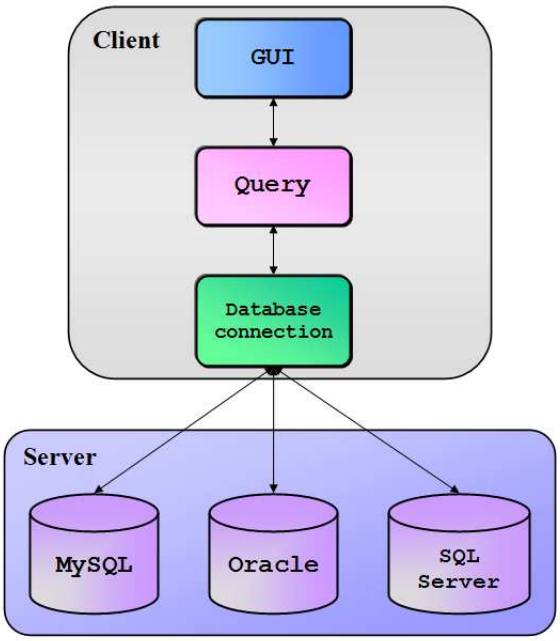
Jednotlivé časti modelu majú v procese komunikácie rozličnú úlohu. Kým jedna časť zabezpečuje priamo spojenie programu s databázou a prenos dát z/do databázy, iná sa stará o komunikácia s užívateľom.
Pre porozumenie, prečo je model navrhnutý práve takýmto spôsobom, je treba dôkladne popísať funkcie týchto častí:

1. Na strane **klienta** sa nachádzajú 3 úrovne, čiže tri rozličné uzly, kde prebieha výmena dát. Komunikácia je obojsmerná, ale iba pre susedné uzly[6]. Charakter prenášaných dát medzi uzlami je vopred definovaný, aby sa predišlo nedorozumeniam na akejkoľvek úrovni.

    - **GUI**[7] (grafické užívateľské prostredie)
      Pre užívateľa predstavuje základnú časť pre prácu s programom (jedinú, ku ktorej má priamy prístup). Ide o zobrazované okná, dialógy na zadávanie hodnôt, potvrdzovanie výberu a iné. V tejto časti si užívateľ vyberá, ktoré dáta chce zobraziť, upraviť alebo vymazať. Ak sa rozhodne pre akúkoľvek zo spomínaných akcií, urobí to definovanou formou (kliknutím na tlačítko alebo stlačením klávesy). Program udalosť analyzuje, vyhodnotí a ak

[5] najmä na strane klienta
[6] podľa orientácie šípiek na obrázku 3.1
[7] z angl. *Graphical User Interface*

Obrázok 3.1: Model aplikácie



prebehlo všetko korektne a nevyskytla sa žiadna chyba, tak vytvorí požiadavku, ktorú pošle ďalej do nižšej úrovne (bližšie k databáze) a čaká na odpoveď.

Odpoveď mu príde v nejakom dohodnutom formáte, tú grafické prostredie akceptuje a podľa nej sa zachová - zobrazí požadované dáta, vykoná nejakú akciu alebo informuje užívateľa o výskyte chyby.

- **Query** (tvorba požiadavky do databázy)
  Tento uzol prijíma požiadavky z užívateľského prostredia, transformuje ich do takej podoby, ktorej rozumie databázový server a posiela ďalej poslednému modulu, ktorým je pripojenie k databáze. Odpoveď, ktorú dostane, vyhodnotí (môže ju opäť transformovať) a odošle vyššie užívateľskému prostrediu.

  Je namieste položiť otázku, prečo táto vrstva nie je súčasťou GUI? Dôvod, prečo bola vrstva oddelená spočíva v tom, že užívateľské prostredie by sa malo starať o správne zobrazovanie prijatých dát užívateľovi a nie o to, ako a odkiaľ tieto dáta získať. Táto úloha patrí niekomu inému a nie je nutne zviazaná s GUI. To umožní zmeniť užívateľské prostredie bez toho, aby sa musela meniť aplikačná logika. S tým súvisí aj rozširovanie systému a prípadne

doplnenie ďalších doplnkov ako je napr. webová aplikácia, ktorá bude zabezpečovať *online* komunikáciu so zákazníkmi[8]. Pri takomto modeli programu stačí vytvoriť webové rozhranie a na prenos dát a vykonávanie niektorých akcií použiť už existujúce aplikačné vybavenie.

- **Database connection** (pripojenie k databáze)
  Posledný "klientský" uzol zabezpečuje konektivitu programu s databázou. V podstate sa nestará o to, aké dáta sa budú prenášať, dôležité je overenie, či daný užívateľ má právo k týmto dátam pristupovať a či je možné požadované spojenie vytvoriť. Ak sa podarí vytvorenie spojenia, poskytne tento uzol komunikačný kanál na výmenu dát, kde na jednej strane bude konkrétna databáza a na druhej strane bude požiadavka užívateľa (uzol *Query*), ktorá posiela príkazy, čaká na ich vykonanie a prijíma odpoveď z databázy.
  Niekto by mohol namietať, že táto časť je zbytočná a je možné ju spojiť s predchádzajúcim uzlom na vytváranie požiadaviek. To však záleží od uhla pohľadu a charakteru aplikácie. Ak bude program zviazaný s jedným typom databázy, tak je vytváranie tejto časti úplne zbytočné, pretože neprinesie žiadne vylepšenie. Výhoda existencie samotného uzla, ktorý zabezpečuje pripojenie k databáze, sa prejaví až vtedy, ak sa bude od aplikácie požadovať, aby nebola zviazaná len s jedným (vopred určeným) druhom databázy, ale aby fungovala v spojení s viacerými databázami (firma môže preferovať určitý typ databázy, pretože má s ním dobré skúsenosti alebo má kvalifikovaných pracovníkov na administráciu takejto databázy).
  Rozličné databázy väčšinou majú rozdielny prístup k dátam[9] a bolo by nerozumné spojiť rozhodovanie o voľbe prístupu s tvorbou požiadavok. Jednak pre to, že tvorbu požiadaviek by nemala ovplyvňovať cieľová databáza, a aj pre to, že vytvorenie spojenia s databázou je proces, ktorý súvisí s požiadavkami iba nepriamo.

2. Na strane **servera** je situácia podstatne jednoduchšia. Nachádza sa tam databáza, v ktorej sú uložené dáta, ale aj časť aplikačnej vrstvy, ktorá je závislá na vybranej databáze[10] (procedúry, triggery na kontrolu integrity dát...).

Týmto môžem považovať analýzu projektu za kompletnú. Zvolený cieľový

[8] B2C - Bussiness to Consumer

[9] Existuje štandardizovaný prístup k databázam známy pod skratkou *ODBC - Open DataBase Connectivity*, ktorý ale nemusia podporovať všetky databázy a ktorý nemusí znamenať tú najefektívnejšiu voľbu pripojenia k databáze

[10] Norma *SQL 92* síce definuje pravidlá pre prácu s dátami, ale výrobcovia DBMS spravidla dodržiavajú iba niektoré jej časti

operačný systém a vytvorený programový model sú faktory, ktoré ďalej ovplyvňujú výber technológie, ktorú použijem pre realizáciu projektu.

Kapitola 4

Výber technológie

Súčasná doba poskytuje programátorom obrovské množstvo nástrojov pre vývoj. Niektoré sú nezávislé od zvolenej platformy, iné sa dajú používať iba vo vopred zvolenom operačnom systéme. Oproti minulosti, kedy existoval iba malý počet programovacích jazykov, v ktorých sa dali vyvíjať aplikácie, je to pokrok. Na jednej strane to prináša výhodu v tom, že analytik má možnosť rozhodnutia, ktorá technológia sa použije, aby sa využili všetky jej prednosti a uľahčila prácu programátorov. Na druhej strane, pri existencií širokej palety možností, je potrebné poznať charakteristiku jednotlivých vývojových nástrojov, aby výber technológie znamenal efektívnejšiu prácu na projekte (treba zobrať do úvahy aj to, či majú vývojári skúsenosti s daným jazykom, prostredím a spôsobom písania aplikácií).

Technológia by nemala obmedzovať programátorov, ale naopak. Mala by im ponúkať také možnosti, aby sa vývoj aplikácie nezaoberal riešením zbytočných problémov na nízkej úrovni (ako pristupovať a čítať z disku, ako naprogramovať zásobník...). Tieto problémy sú tu už dávnejšie a existujú riešenia, ktoré sú často efektívnejšie než tie, ktoré by si programátori narýchlo napísali sami. Ušetrí im to čas a ich pozornosť sa môže sústrediť na vytváranie dôležitých častí aplikácie tak, aby splňovala užívateľove kritéria.

4.1 Dostupné technológie

Ako som už spomínal, množstvo a dostupnosť technológií stále rastie a možností je čím ďalej, tým viac. Uvažovať budem iba o vývojových nástrojov, ktoré sú určené pre zvolený operačný systém[1] a ktoré umožňujú vytváranie aplikácií založených na architektúre *klient-server*.

Len v prostredí Windows existuje dostatok nástrojov, ktoré sa dnes bežne používajú. Rozhodoval som sa medzi nasledujúcimi jazykmi a tech-

[1] viď strana 14

nológiami[2]:

- Delphi
- C, C++ (v kombinácii s Win32 API)
- Java
- platforma .NET (jazyky C# ,VB.NET)

Ako to už býva, každý z týchto jazykov (vývojových prostredí) má svoje výhody aj nevýhody. Môj výber bol značne ovplyvnený tým, že s niektorými technológiami som pracoval viac, s niektorými menej, a vedel som odhadnúť na základe vlastných skúseností, ktorý jazyk bude pre vývoj aplikácie najlepší.

1. Jazyk *Delphi* je objektovo orientovaný a založený na známom Pascale. Spolu s vývojovým prostredím (nazývané tiež Delphi, aktuálne vo verzií *Delphi 2005*) tvoria údajne silný nástroj pre vývoj rozličných aplikácií (desktopových, databázových). Moje skúsenosti s prácou v tomto (objektovo orientovanom) jazyku boli malé a preto som nevedel reálne posúdiť silu jazyka v databázovej aplikácií. Najmä v súvislostí s tvorbou užívateľského prostredia a spojením s rozdielnymi databázami.

2. Prostredie *Win32 API* mi bolo známejšie, pretože som už pre danú platformu vyvíjal aplikáciu (aj s grafickým užívateľským rozhraním). Medzi výhody nesporne patrí unifikovanosť na rozličnýsh sytémoch Windows a fakt, že programy vytvorené na tejto báze sú pomerne rýchle, čo je pre užívateľa určite prínos. Lenže z pohľadu programátora je vývoj zdĺhavejší, pretože je treba riešiť rôzne veci, ktoré sú pre databázovú aplikáciu vedľajšie - komunikáciu s operačným systémom, vzájomnú komunikáciu okien, vytváranie okien a ovládacích prvkov a iné. Teda v mojom prípade, keď užívateľské rozhranie je veľmi podstatné, by to znamenalo viacej práce s grafickým rozhraním (okná na zobrazovanie dát) a menej času by zostalo na tvorbu aplikačnej logiky. To bol hlavný dôvod, prečo som túto technológiu zamietol.

3. Obľubenosť jazyka *Java* v súčasnosti rastie. Rozličné balíky jazyka sú určené pre rôzne formy použitia (J2ME - mobilné telefóny a zariadenia PDA, J2SE - domáce použitie, J2EE - enterprise edícia atď.). Návrh užívateľského prostredia je však trošku komplikovanejší. Naopak podobne ako pri platforme .NET, aj pre aplikácie napísané v Jave je

[2]Vymenované technológie tvoria výber z toho, čo sa aktuálne používa, určite netvoria kompletný zoznam možností.

potrebné mať nainštalovaný nejaký framework[3], aby bolo možné program spustiť. Dôvody odmietnutia tejto technológie boli dva. Prvým bol komplikovanejší vývoj grafického užívateľského prostredia, druhý v nutnosti inštalácie JVM. Hoci v dnešnej dobe ešte .NET nie je súčasťou všetkých operačných systémov Windows, v budúcnosti s tým pravdepodobne môžeme počítať. Teda nebude potrebné inštalovať framework, aby užívateľ mohol spustiť program napísaný v prostredí .NET.

Nakoniec teda voľba prostredia padla na platformu .NET.

4.2 .NET

Technológia *.NET* v sebe skrýva prostredie na vývoj a beh aplikácií [6]. Skladá sa z dvoch hlavných častí:

- *CLR*[4] - prostredie potrebné na beh aplikácií
- *FCL*[5] - knižnice určené pre vývoj aplikácií, poskytujúce rôzne základné funkcie (vytváranie okien, ovládacích prvkov), ale aj pokročilé možnosti (komunikácia s databázou, práca s XML)

Podstata je v tom, že generovaný kód nie je určený priamo pre skutočný stroj, ale pre nejaký *abstraktný stroj* (v tomto prípade je abstraktným strojom CLR). Samozrejme to v sebe skrýva výhody a nevýhody. Keďže je potrebné interpretovať kód (konverzia na inštrukcie procesora), môže sa zdať, že programy budú pomalé. To je čiastočne vyriešené pomocou kompilácie JIT[6]. Výhodou je zvýšenie bezpečnosti (možnosť "kontrolovať" vykonáne inštrukcie). Naviac *.NET* ponúka rovnaké možnosti vývoja pre všetky dostupné jazyky vo forme FCL.

Na vytvorenie aplikácie som mohol použiť ktorýkoľvek jazyk prostredia *.NET*. Zvažoval som výber jedného z troch najpoužívanejších jazykov: *managed C++*[7]*, Visual Basic .NET a C#*. Keďže som bol zvyknutý písať skôr v jazyku C (C++), nie v jazyku Visual Basic, a vedel som, že v jazyku *managed C++* sa niektoré veci robia zložitejšie, rozhodol som sa pre jazyk C# (ktorý ako jediný z tejto trojice bol navrhnutý priamo pre prostredie .NET), aby som sa mohol plne sústrediť na vývoj aplikácie a nemusel zbytočne riešiť technické problémy.

[3] *JVM - Java Virtual Machine*, ktorá existuje pre rôzne operačné systémy
[4] z angl. *Common Language Runtime*
[5] z angl. *Framework Class Library*
[6] z angl. *Just in time*
[7] jazyk C++ v prostredí .NET s *riadeným kódom* (z angl. *managed code*)

Kapitola 5

Implementácia

Prvá časť mojej práce spočívala v analyzovaní požiadaviek potenciálnych užívateľov systému. Toto skúmanie kritérií, ktoré sa bežne vyskytujú v praxi, mi pomohlo v tom, že program som vyvíjal už od začiatku "pre užívateľa", s ohľadom na jeho potreby tak, aby sa mohol plne sústrediť na svoju prácu a nemusel stráviť veľa času skúmaním, ako program funguje. Dôležité bolo ponúknuť užívateľovi čo najviac možností, aby mu program jeho prácu zjednodušil. To znamenalo, okrem iného, že som musel vytvoriť prehľadné, užívateľsky prívetivé[1], grafické prostredie. Samozrejme som zabezpečil, aby sa program dal ovládať klávesnicou aj myšou.

5.1 Klient

Práca s informačným systémom by sa dala z pohľadu ľudí, ktorí s ním pracujú, rozdeliť na niekoľko častí : prihlásenie, práca so systémom (zadávanie nových záznamov, úprava existujúcich atď.), odhlásenie. Ide o jednoduchú predstavu, ktorá v skutočnosti až taká jednoduchá nie je. Užívateľ sa prevažne nestará o to, čo sa deje na pozadí, zaujíma ho len výsledok jeho práce a to, čo vidí na obrazovke. Tomu treba aspoň čiastočne prispôsobiť chovanie programu.

Postupne popíšem niektoré dôležité procesy, ktoré prebiehajú na klientskej stanici. Ide skôr o taký pohľad "do zákulisia", ktorým vysvetlím, ako je program prispôsobený chovaniu, aké od neho užívatelia očakávajú.

5.1.1 Prihlásenie do programu

Pred prácou s programom je potrebné overiť, že daný užívateľ môže pracovať so systémom. Neautorizovaný užívateľ by nemal vstupovať do systému, aby

[1] z angl. *user friendly*

tam nenapáchal zbytočné škody. Musel som teda vyriešiť, ako realizovať prihlásenie užívateľa a zabránenie prístupu neoprávneným osobám. Rozhodol som sa využiť možnosť, ktorú ponúka databázový server, v podobe prihlasovania do konkrétnej databázy. Nastavovanie prístupových práv prebieha na strane servera, kontroluje ich správca databázy a obyčajný užívateľ ich nedokáže zmeniť. Takže na strane klienta som nemusel riešiť nič. Pred spustením programu sa akurát spýtam užívateľa na jeho prihlasovacie meno[2] a heslo, skúsim sa s takýmito údajmi prihlásiť do databázy a zachovám sa podľa odpovede zo strany servera. Ak server umožní takémuto užívateľovi prístup, program sa spustí a užívateľ môže začať prácu so systémom. Ak server odmietne požiadavku na spojenie, môže to byť z rôznych dôvodov - nesprávne užívateľské meno alebo heslo, nesprávne zvolená databáza, server môže byť dočasne odpojený a iné. Preto by nebolo vhodné, aby sa aplikácia okamžite ukončila, ale umožnil som opakovať pokus o pripojenie. Opakované pripojenie môže byť s rovnakými údajmi, alebo ich môže užívateľ pred ďalším pokusom nastaviť na iné hodnoty.

5.1.2 Grafické prostredie

Tvorbou grafického prostredia som strávil pomerne dlhý čas. Jednak pre to, že okná, ktoré sa zobrazujú užívateľovi, by mali byť aspoň trošku podobné[3], pretože tak si užívatelia ľahko zvyknú na ovládanie okien. S tým súvisí aj fakt, že ovládanie podobných okien by nemalo byť odlišné. Ďalším dôvodom bolo, že som musel nájsť správne a prehľadné rozmiestnenie ovládacích prvkov v oknách a vybrať tie podstatné dáta, ktoré sa zobrazia ihneď pri otvorení okna. Niekedy je ťažké vybrať také dáta, pretože tie, čo jeden považuje za dôležité, sa inému môžu zdať nepodstatné. Ide o subjektívny názor a tak som vytvoril možnosť, aby si užívateľ mohol v niektorých oknách sám vybrať, ktoré dáta chce zobraziť a ktoré pre neho nemajú význam a môžu zostať skryté. Sú to prevažne okná zobrazujúce podrobnosti o doklade, kde je veľké množstvo informácií.

Aplikovaním predchádzajúcich kritérií na podobnosť okien som dospel k záveru, že bude potrebné použiť určitú hierarchiu okien. Využil som poznatky z objektovo orientovaného programovania a realizoval som hierarchiu formou dedičnosti. Základom bolo vytvorenie spoločného predka všetkých okien. Jeho potomkovia potom zdedia spoločné črty a doplnia vlastné metódy, ak sú potrebné (väčšinou ide o špecifické chovanie). Týmto krokom som dosiahol aj to, že som v programe mohol uplatniť používanie klávesových skratiek. Väčšinu z nich som umožnil už v spoločnom predkovi a tak sa chovanie potomkov stalo štandardizovaným.

[2] z angl. *login*

[3] mali by zachovávať určitý štandard v oblasti vzhľadu okna a rozmiestnenia ovládacích prvkov v okne, ak to je možné

Jeden predok však v hierarchií nestačil a tak som potreboval doplniť ďalších podľa *typu okna*. Nasledujúci zoznam ukazuje prehľad najpoužívanejších typov okien aj s popisom.

- Okno pre zobrazovanie dát
  Ide o klasické okno na zobrazovanie informácií z tabuliek, číselníkov, kde nie je veľa informácií a typicky ani veľa záznamov.

- Okno pre zobrazovanie dát s filtrom
  Jedná sa o rozšírenie predchádzajúceho typu okna. Ak sa zobrazuje väčšie množstvo dát, je vhodné ich niekedy filtrovať podľa určitých kritérií, čo umožní jednoduchšie vyhľadávanie záznamov. Vytvoril som okná s vopred danými atribútmi, podľa ktorých je možné filtrovať (atribúty sú závislé od typu zobrazovaných dát).

- Dialógové okno
  Predok pre všetky dialógové okná, ktoré zabezpečujú pridávanie a modifikovanie údajov, aby ich chovanie bolo unifikované.

- Okno dokladu
  Ide o najkomplexnejšie okno, v ktorom sa nachádza veľké množstvo ovládacích prvkov a nastavení. Základnú časť okna som vytvoril podľa toho, že doklady majú podobnú štruktúru. Všetky spoločné vlastnosti sú presunuté sem, ostatné, ktoré sú špecifické pre jednotlivé doklady sú realizované až v potomkoch.

Vytvorenie dedičnosti a jej aplikácia mi umožnila efektívne nastaviť aj meniť vzhľad a chovanie jednotlivých okien. Prípadné zmeny som neupravoval na viacerých miestach, stačilo iba pozmeniť chovanie predka.

Zobrazovanie okien som vyriešil metódou MDI[4]. Je to spôsob ako zobraziť viacej okien (zobrazované okná sa nazývajú *MDI children*) naraz v jednom hlavnom okne (*MDI parent*). Prináša to množstvo výhod (užívateľ si môže otvoriť viacej okien naraz a pracovať s nimi "súčasne"), ale aj nevýhod (chovanie okien je trošku iné ako keby boli samostatné, naraz môžu byť zobrazené dve menu,čo môže pomýliť aj skúseného užívateľa). V mojom prípade to znamenalo prevažne prínos, preto som zvolil práve túto metódu[5]

V budúcnosti možno pribudne spôsob, ktorý bude fungovať na princípe MDI, ale iba jedno okno bude aktívne a ostatné budú síce otvorené, ale zobrazené vo forme *záložiek*[6].

[4]z angl. *Multiple Document Interface*

[5]Existuje aj metóda nazývana *SDI - Single Document Interface*, kde je každé okno aplikácie zobrazované zvlášť a tak sa k nemu správa aj operačný systém.

[6]angl. *tabs*

Pri navrhnutom programovom modeli[7] sa mi zdalo neefektívne, aby sa správnosť niektorých prenášaných dát kotrolovala až v databáze. Znamenalo by to často prenos zbytočného množstva dát po sieti a zaťažovanie nielen sieťovej linky ale aj servera. Preto som zvolil metódu čiastočnej kontroly už na strane klienta a prispôsobil som jej aj grafické prostredie. Zadávanie nových údajov a opravovanie existujúcich prebieha formou dialógových okien. Ak sa rozhodne užívateľ akceptovať zadané zmeny, prebehne validácia niektorých vstupných dát, ktoré je možné kontrolovať už na strane klienta (napríklad vyplnenie atribútov, ktoré majú v tabuľke status *NOT NULL*). Ak je všetko v poriadku, dáta sú poslané do databázy. To ešte nezaručí, že budú akceptované, ale moja metóda ponúka aspoň čiastočnú istotu. Obmedzí to prenos takých dát, ktoré by určite nemohli byť v databáze uložené. Tým znížim aspoň čiastočne zaťaženie servera aj prenosovej linky a práca servera bude efektívnejšia. Dáta aj napriek tomu nemusia byť akceptované z rôznych dôvodov. Medzi najbežnejšie patria:

- Porušenie integrity dát
  Napríklad, ak zvolený primárny kľúč už v databáze existuje.

- Dáta boli medzičasom zmenené alebo úplne vymazané
  Tento prípad môže nastať v prostredí viacerých užívateľov pomerne často, záleží od spôsobu, ako sa pristupuje k editácií dát. Existujú dva navzájom odlišné spôsoby. Prvý z nich sa nestará o to, aké dáta užívateľ prepisuje alebo odstraňuje. Ide o *pesimistický* prístup a výsledok záleží na tom, kto dáta upraví ako posledný. Na druhej strane sa nachádza prístup *optimistický*, ktorý je postavený na tom, že umožní úpravu záznamu iba vtedy, ak ešte nebol modifikovaný. Znamená to testovanie nielen zhodnosti primárneho kľúča, ale aj zvyšných atribútov a ich porovnanie s pôvodnými hodnotami. Ak jeden z užívateľov chce záznam opraviť a pokúsi sa o to aj ďalší užívateľ, iba jeden z nich môže zmenu vykonať. Je to ten, kto upraví záznam ako prvý (chyba pri opravovaní nenastane, pretože záznam má pôvodné hodnoty). Ostatním sa zmena záznamu už nepodarí. Aktualizáciou dát z databázy získajú záznamy, ktoré už môžu potom upravovať.

V systéme som použil kombináciu spomínaných spôsobov prepisovania dát, pretože používanie výhradne optimistickej metódy znamenalo skomplikovanie príkazov. Preto je väčšina modifikácií založená na čiastočne optimistickom, v niektorých prípadoch dokonca na úplne optimistickom spôsobe (záleží to od konkrétnej tabuľky v databáze).

[7] viď obrázok 3.1

5.1.3 Komunikácia s databázou

Zvolením technológie .NET som získal možnosť pripojenia k databáze pomocou *ADO.NET*. Architektúru tejto technológie tvoria dve hlavné časti, ktoré umožňujú jednoduchú komunikáciu s databázou [2]:

- *Poskytovateľ dát*[8]
- *Množiny dát*[9]

Prvú časť tvoria triedy, ktoré zabezpečujú pripojenie ku konkrétnej databáze, vytváranie objektov, ktoré reprezentujú príkazy (spolu s možnosťou zadávať parametre príkazom) a objektov, ktoré umožňujú spracovanie dát pomocou transakcií. Súčasťou je aj tzv. *Data Adapter*, ktorý slúži ako sprostredkovateľ prenosu dát medzi zdrojom dát a množinou dát, poprípade *Data reader* na postupné čítanie rozsiahlych dát.

Množiny dát sú objekty, ktoré uchovávajú nielen záznamy z databáze, ale môže ukladať aj prípadné integritné obmedzenia[10]. Trieda *DataSet* sa skladá z viacerých častí. Sú to *DataTable* (v množine sa môžu nachádzať dáta z viacerých tabuliek), *DataRow* a *DataColumn* (jednotlivé riadky a stĺpce zvolenej tabuľky), *DataView* (filtrovanie riadkov) a iné [5]. V projekte som na prístup k dátam a ich filtrovanie využíval práve spomínané objekty.

Problém s technológiou ADO.NET spočíva v tom, že rôzne databázy majú zvyčajne rôzneho poskytovateľa dát (triedu, ktorá zabezpečí komunikáciu s vybranou databázou). To je výhodné pre aplikácie, kde sa požaduje iba jeden, vopred daný, druh databázy (používajú sa konkrétne triedy zvoleného poskytovateľa). Môj návrh však počítal v budúcnosti s podporou viacerých databáz. To znamenalo, že som musel nejakým spôsobom vyriešiť rozdielnosť medzi jednotlivými poskytovateľmi dát. Riešením bolo vytvorenie vlastnej triedy, ktorá zabezpečovala pripojenie a prácu s databázou. Požiadavkou bolo, aby využívanie tejto triedy v programe už nebolo závislé od voľby databázy. Na začiatku síce užívateľ zvolí typ databázy, s ktorou chce komunikovať a vytvorí spojenie, ale program sa už viac nestará o to, s akou databázou komunikuje. O to sa stará práve vytvorená trieda. Ponúka aplikácií unifikované rozhranie na výmenu dát s rozličnými databázami. Vyriešenie konkrétnych rozdielov rôznych poskytovateľov je umiestnené vo vnútri triedy a navonok je práca s dátami stále rovnaká.

V prípade dokladov som musel využiť na vykonávanie príkazov aj transakcie. Spôsobila to štruktúra dokladov - doklad je rozdelený na hlavičku a položky. V hlavičke sú údaje, ktorými je doklad identifikovaný a ďalšie hodnoty (odberateľ, dátum vystavenia dokladu, atď.), kým položky, ktoré

[8] z angl. *Data provider*
[9] z angl. *Data sets*
[10] angl. *constraints*

bývajú uložené v inej tabuľke, sú jednotlivé záznamy tvoriace doklad. Ak som chcel upraviť nejakú položku, znamenalo to zároveň aj upraviť *kontrolný súčet*[11] v hlavičke dokladu. Samozrejme táto operácia by mala byť *atomická*. Na zaistenie atomicity som použil práve rôzne úrovne transakcií.

5.2 Server

Úloha databázového servera je vcelku jednoduchá a predsa veľmi dôležitá. Umožňuje pripojenie do databázy, poskytuje užívateľom uložené dáta, stará sa o konzistenciu uložených dát. Samozrejme zabezpečuje aj kontrolu prístupu do databázy. V tomto smere by mal byť server dostatočne chránený pred neoprávnym používaním, o čo sa stará databázový administrátor.

Na strane servera môže byť časť aplikačnej vrstvy systému, ktorá by sa v klientskej časti riešila veľmi zložito alebo by sa vyriešiť nedala. K tomuto účelu ponúkajú databázy rôzne možnosti od integrovaných funkcií, cez užívateľsky definované procedúry až po tzv. *spúšťače*[12], ktoré zabezpečujú integritu a kontrolu vkladaných, upravovaných alebo odstraňovaných dát. Problémom pri používaní spúšťačov je, že môžu obsahovať príkazy, ktoré nie sú *prenositeľné* medzi jednotlivými typmi databáz a preto sú závislé od zvoleného poskytovateľa. To je hlavný dôvod, prečo som si vybral konkrétnu databázu, pre ktorú bude systém (aspoň na začiatku) určený. V mojom prípade je to *SQL Server 2005*.

Ako som už spomínal v časti o prihlasovaní do programu[13], databázový server som využil na určovanie prístupových práv aj v klientskej časti. Preto je veľmi dôležité správne nastaviť prístupové práva do databázy iba pre tých užívateľov, ktorí s ňou budú aktívne pracovať. Zvolenie skupiny takých užívateľov záleží od konkrétnej politiky podniku, ktorý bude systém využívať.

[11] v hlavičke dokladov väčšinou ukladám aj záznam o celkovej hodnote dokladu, ktorý zodpovedá súčtu hodnôt jednotlivých položiek

[12] z angl. *triggers*

[13] viď strana 23

Kapitola 6

Práca s programom

V nasledujúcej časti stručne popíšem, ako program z pohľadu užívateľa funguje a vysvetlím spôsob práce so sytémom. Nepôjde o rozsiahly popis všetkých funkcií a možností, ktoré program ponúka[1], ale prehľad základných prvkov, na ktorých je pogram postavený.

Postupne sa dostanem od inštalácie a nastavenia programu, cez užívateľské prostredie až po problematiku tvorby dokladov.

6.1 Inštalácia

Program je určený pre platformu Windows a inštaluje sa pomocou jednoduchého inštalátora, ktorý prevedie užívateľov celým procesom inštalácie. Pred spustením inštalácie je potrebné skontrolovať, či pracovná stanica spĺňa aspoň základné požiadavky (softwarové aj hardwarové) pre správny chod programu. Medzi tieto požiadavky patrí napríklad dostatočný voľný priestor na pevnom disku, či nainštalovaný *.NET Framework 2.0*. V prípade, ak sa na cieľovom počítači nenachádza *.NET Framework 2.0*, môže ho užívateľ nainštalovať buď priamo z priloženého CD alebo stiahnuť jeho aktuálnu verziu z internetu.

Ak sú všetky podmienky splnené, užívateľ môže začať inštalovať program. Počas inštalácie je potrebné zadať umiestnenie programu, ostatné časti sa prevedú automaticky. Po inštalácií sa vytvorí odkaz v ponuke Štart a na ploche, ktorý umožní okamžité spustenie programu.

6.2 Nastavenie a spustenie programu

Po úspešnom nainštalovaní programu môže užívateľ začať program používať. Po prvom spustení prgramu je vhodné nastaviť vlastnosti pripojenia (adresu

[1] Tie sú k dispozícií v užívateľskej dokumentácií, kde sa nachádzajú aj obrazové prílohy pre ľahšiu orientáciu.

servera, názov databázy, spôsob autentifikácie) a tieto nastavenia uložiť, aby ich u+z9vate+l nemusel zadávať pri každom ďalšom prihlasovaní. Správne hodnoty závisia od použitého databázového servera.

Ak konfigurácia pripojenia k databáze nie je zadaná správne, program síce bude možné spustiť, ale nebude možné s ním plnohodnotne pracovať.

6.3 Užívateľské prostredie

Pre začínajúcich užívateľov je grafické prostredie zväčša rozhodujúcim faktorom pri výbere programu. Nestarajú sa príliš o konkrétne výhody vybraného systému, ktoré ho väčšinou robia výnimočným oproti konkurencií, ale zaujíma ich predovšetkým, ako program vyzerá po spustení. Na druhej strane užívatelia, ktorí si už vyskúšali prácu s viacerými aplikáciami a teda majú skúsenosti s ich každodenným používaním, uprednostňujú skôr funkčnosť a efektívnosť pred grafickým prostredím. Ideálnym riešením je teda vytvorenie efektívneho a graficky prívetivého systému. Tu nastáva problém, pretože na jednej strane každý užívateľ je zvyknutý na nejaký iný spôsob ovládania programu, zobrazovania okien..., na strane druhej zvyšovanie efektívnosti je obmedzené zvolenou technológiou.

Preto som sa snažil vytvoriť program tak, aby sa príliš neodlišoval od iných programov určených pre platformu Windows a umožnil tak jednoduchú prácu pre užívateľov, ktorí s týmto systémom pracujú. K tomu som využil viaceré možnosti, ktoré prostredie *.NET* ponúka.

Zabezpečiť som musel podmienku pre reálne využívanie programu, aby užívateľ mohol ovládať program myšou aj klávesnicou. Obidva spôsoby by mali byť porovnateľné a rovnocenné (z hľadiska efektivity práce) a nemalo by sa stať, že niektoré možnosti programu sú dostupné iba pre jeden spôsob. Aj v dôsledku toho je v programe veľké množstvo klávesových skratiek, ktoré prinášajú väčšie pohodlie pri práci užívateľom, ktorí sú zvyknutí na ovládanie programu klávesnicou.

6.3.1 Ovládanie programu

Program je založený na princípe *MDI*[2], ktorý umožňuje vytvorenie a zobrazenie viacerých okien súčasne. Zobrazené okná si môže užívateľ rozložiť na obrazovke ľubovoľne alebo môže použiť niektoré zo základných možností zobrazenia viacerých okien - horizontálne, vertikálne alebo kaskádovo. Následne sa môže prepínať medzi jednotlivými oknami, vykonávať zmeny v jednom okne a sledovať ich dôsledok v ďalších oknách.

V multiužívateľskom režime je vhodné aktualizovať dáta z databázy čo najčastejšie, aby sa predišlo konfliktom, ktoré veľmi ľahko nastanú, ak budú

[2]viď strana 25

napríklad viacerí užívatelia naraz a nezávisle na sebe upravovať rovnaký záznam. V takom prípade môže dôjsť k tomu, že jeden z užívateľov záznam opraví, ale ostatní ešte zmenu nevidia. Na druhej strane aktualizácia by nemala byť až príliš častá, aby sa veľmi nezaťažovala sieťová linka a databázový server.

Samotné vytváranie a zobrazovanie okien prebieha dvoma spôsobmi a záleží len od užívateľa, ktorej voľbe dá prednosť:

- Z hlavného menu kliknutím na požadovaný doklad, tabuľku alebo číselník.
- Dvojitým kliknutím na požadovaný uzol v stromovej hierarchií v tzv. *rýchlom menu*, ktoré sa nachádza v ľavej časti hlavného okna (môže byť skryté, ak ho užívateľ nevyužíva).

V závislosti od charakteru okna[3] sa zobrazí požadované okno a je možné s ním pracovať - prezerať, vytvárať, opravovať a vymazávať záznamy. Niektoré okna disponujú filtrom pre jednoduchšie vyhľadávanie záznamu podľa viacerých, vopred určených, atribútov. Umiestnenie filtra záleží od typu okna, väčšinou sa nachádza v hornej alebo v pravej časti okna (pri zozname dokladov je na ľavej strane). Filter reaguje na zmenu hodnôt, podľa ktorých sa filtruje, okamžite a zobrazí iba tie záznamy, ktoré vyhovujú aktuálnemu nastaveniu.

Pridávanie nových záznamov a editácia existujúcich prebieha vždy formou dialógových okien, ktoré sa zobrazujú viacerými spôsobmi buď z menu, kontextového menu, ktoré sa zobrazí, ak užívateľ klikne pravým tlačítkom na zoznam záznamov, alebo klávesovými skratkami. Podobná manipulácia je aj s položkami a zoznamom dokladov.

6.4 Tvorba dokladov

Vytváranie a evidencia dokladov patrí k základným možnostiam, ktoré informačné systémy ponúkajú. Je potrebné uchovávať doklady v takom stave, ako boli vytvorené a uložené do databázy, aby bolo možné k nim pristupovať aj spätne. V niektorých prípadoch je nutné kvôli tomu ukladať nejaké údaje duplicitne.

Manipulácia s dokladmi v multiužívateľskom režime so sebou prináša niekoľko problémov. Ide nielen o to, ako uchovávať dáta v konzistentnom stave, ale aj ako správne zabezpečiť číslovanie novovytvorených dokladov.

[3] viď strana 24

6.4.1 Číslovanie dokladov

Problematika číslovania dokladov je pomerne rozšírená a existuje niekoľko rôznych spôsobov, ako doklady číslovať. V podstate je možné vybrať akýkoľvek spôsob, ale musí dodržiavať zákonom určené pravidlá. Napríklad pri vystavených faktúrach treba zabezpečiť, aby boli číslované chronologicky podľa dátumu vytvorenia.

Ja som zvolil spôsob číslovania dokladov, ktorý má možnosť užívateľ ovplyvňovať. Niektoré doklady môžu mať viacero kategórií (napríklad príjemky a výdajky, kde kategória je číslo skladu a znamená naviazanosť dokladu na sklad). Každá kategória môže mať odlišný spôsob číslovania alebo sa môžu číslovať všetky kategórie rovnako. V praxi sa odporúča číslovať každú kategóriu, resp. sklad, individuálne a odlišne od ostatných, aby bolo potom jednoduchšie určiť príslušnosť dokladu k zvolenej kategórií (pre skladový doklad príslušnosť k skladu).

Takisto som umožnil užívateľovi, aby obmedzil množstvo vytvorených dokladov v danej kategórií tým, že určí maximálne číslo dokladu.

Najdôležitejšie bolo zabezpečiť správne vytváranie a očíslovanie nových dokladov. Môj prístup spočíva v tom, že pokiaľ užívateľ neprenesie zmeny do databázy, žiadny doklad nie je vytvorený, užívateľ ho vidí iba "lokálne"vo svojom programe, iní užívatelia ho nevidia. Zabezpečím tak chronologické číslovanie, pretože dokladu je pridelený jeho identifikátor až pri uložení do databázy. Keďže vytvorenie dokladu prebieha vrámci transakcie (je potrebné uložiť hlavičku dokladu a následne položky dokladu), o konzistenciu dát sa stará databázový server podľa zvolenej úrovne transakcie.

Po vytvorení sa číslo dokladu stáva jedinečným a nemenným vrámci danej kategórie dokladov. V niektorých prípadoch má zmysel aj prečíslovanie dokladov, ale to je vhodné až po skončení účtovného obdobia.

6.4.2 Evidencia dokladov

Keďže všeobecne môže k dokladu pristupovať v jednej chvíli viacero užívateľov, ľahko sa môže stať, že si budú navzájom prepisovať údaje. Podľa zvoleného prístupu môžu nastať dve situácie:

- Každému z užívateľov sa oprava dokladu podarí, pretože pri vyhľadávaní dokladu, ktorý sa má opravovať, sa neberú do úvahy všetky atribúty, ale iba identifikátor dokladu. To, aké hodnoty budú nakoniec uložené v databáze, záleží od toho, ktorý užívateľ ich prepíše ako posledný. Nevýhodou takéhoto prístupu je, že užívateľ nebude vedieť o tom, že prepisuje dáta, ktoré možno prepísať nechcel a môže si myslieť, že ostatní užívatelia budú brať ohľad na jeho zmeny, čo tiež nemusí byť pravda.

- Druhý spôsob, na základe ktorého som vytvoril svoje riešenie, je založený na tom, že sa kontrolujú aj iné atribúty než len primárny kľúč záznamu. Všeobecne sa odporúča porovnávať všetky atribúty, čo môže byť miestami príliš komplikované. Preto som zvolil porovnávanie niekoľkých (nie všetkých) atribútov. Vybral som si také atribúty, ktoré sú dôležité, naopak do úvahy som nebral nepodstatné atribúty (poznámky a pod.).

Samozrejmosťou je, že akákoľvek modifikácia dokladu musí prebiehať v transakcií, pretože je nutné opraviť najskôr hlavičku dokladu a následne vykonať zmeny v položkách dokladu. Počas tohto procesu je nutné zamknúť upravované záznamy (na úrovni databázy), aby ich iní užívatelia nemohli počas vykonávania transakcie zmeniť. Zamykanie a odomykanie záznamov je ponechané na databázový server, ktorý to väčšinou robí automaticky podľa zvolenej úrovne izolácie transakcie. Niektoré databázy však umožňujú aj explicitné uzamykanie vybratých záznamov, tabuliek atď.

6.5 Rozšírenia programu

Počas vytvárania informačného systému som prišiel na niekoľko možností, ako by sa mohol systém v budúcnosti rozširovať. Ide nielen o pridávanie ďalších doplnkov, ktoré by umožnili užívateľom pohodlnejšiu prácu s programom, ale aj o vytváranie nových komponent, ktoré by ponúkali ďalšie možnosti.

Z užívateľského hľadiska by bolo príjemné doplniť tlač dokladov, prípadne ich export do elektronickej formy (XML, popr. mail) spolu s následnou možnosťou elektronickej komunikácie so zákazníkmi. Keďže by išlo o vopred definovaný formát prenášaných dát, uľahčilo by to prácu užívateľom, ktorí zadávajú dáta do systému, pretože by sa ich práca dala automatizovať. Kontrola vkladaných dát a následne ich vloženie priamo do databázy by mohla vykonávať nejaká komponenta aplikácie. Užívateľ by iba určil vstupné, resp. výstupné dáta, o ostatné by sa postaral systém.

Na zjednodušenie vytvárania podobných dokladov by mohli byť prístupné klasické operácie - kopírovať, vystrihnúť, vložiť a to nielen pre prácu s položkami ale aj s celými dokladmi. To je výhodné pri častom vytváraní dokladov s veľkým množstvom položiek.

S tým súvisí aj možnosť odvolávania zmien (*undo*). Momentálne je možné sa vrátiť iba k pôvodnému stavu dokladu, nie je možné sa vrátiť k východziemu stavu postupnými krokmi.

Ďalšími doplnkami by mohlo byť ukladanie obrázkov k vybraným produktom alebo ukladanie naskenovaných prijatých dokladov.

Aplikácia je navrhnutá tak, aby ju bolo možné doplniť aj o ďalšie rozhranie na zadávanie a zobrazovanie dát, napr. vo forme webových stránok.

Vytvorilo by sa tak webové rozhranie, prostredníctvom ktorého by mohla prebiehať komunikácia so zákazníkmi (B2C) alebo *online* obchodovanie, či kontakt s dodávateľmi (B2B). Táto možnosť je k dispozícií vďaka tomu, ako je členená klientská časť aplikácie[4].

Okrem spomenutých rozšírení treba spomenúť aj prechod na iný databázový server než je aktuálne podporovaný *SQL Server 2005*. Existuje množstvo iných databázových serverov, ktoré sa bežne používajú a niektoré z nich sú voľne prístupné, takže rozšírenie v tomto zmysle by prinieslo možnosť vybrať si databázový server, ktorý bude najlepšie vyhovovať nárokom užívateľov.

[4]viď obrázok 3.1, ktorý znázorňuje model aplikácie

Kapitola 7

Záver

Hlavným cieľom bakalárskej práce bolo vytvorenie plnohodnotného informačného systému, ktorý bude poskytovať evidenciu a prácu so skladovými dokladmi (príjemka, výdajka, dodacie listy, faktúry) spolu s možnosťou vystavovať, resp. prijímať ponuky a objednávky. Cieľovými používateľmi mali byť malé a stredné podniky, preto som pri vývoji aplikácie bral do úvahy najmä požiadavky takýchto spoločností na funkčnosť a efektívnosť programu, ktorá je všeobecne závislá od množstva spracovávaných dát. Samozrejmosťou bolo vytvoriť užívateľsky prívetivé grafické prostredie spolu s jednoduchým a prehľadným ovládaním. K zvýšeniu užívateľského pohodlia som vytvoril množstvo vylepšení, ktoré prácu s programom podstatne zjednodušujú (napr. rýchle menu, číselníky a iné).

V niektorých prípadoch bolo potrebné v programovom modeli porušiť pravidlá z teórie databáz (porušenie normálnej formy a ukladanie duplicitných údajov), aby som dosiahol efektívnejšiu prácu s dátami.

Jedným z hlavných prínosov bolo uplatnenie teoretických znalostí z rôznych oblastí programovania v praxi. Išlo o jednoduché aj zložitejšie programátorské techniky v závislosti od komplexnosti riešeného problému (použitie dedičnosti na vytvorenie hierarchie, transakcie pre korektné spracovanie a zabezpečenie konzistencie dát a iné) tak, aby použitý spôsob riešenia zodpovedal rozsahu problému.

Počas vytvárania aplikácie som získal aj nové poznatky z oblastí účtovných zákonov a skladového hospodárstva. Musím poznamenať, že zabezpečenie, aby sa program správal v súlade so zákonom a dodržiaval všetky príslušné zákony a normy, bolo v niektorých častiach systému komplikovanejšie (napr. pri metóde oceňovania *FIFO*).

Do prílohy som umiestnil ukážky grafického prostredia (okná pre prácu s dátami) a obrázok práce s programom pri otvorení viacerých okien súčasne.

Literatúra

[1] Adámek P.: Ročníkový projekt, VUT Brno
`http://www.herkules.cz/doc/rp/node7.html`

[2] Wikipedia
`http://en.wikipedia.org/wiki/ADO.NET`

[3] MSDN Library
`http://msdn2.microsoft.com/en-us/library/default.aspx`

[4] Pokorný, Halaška: Databázové systémy, skripta FEL ČVUT, 2003

[5] Price J.: *C# - programování databází*, Grada, Praha, 2005

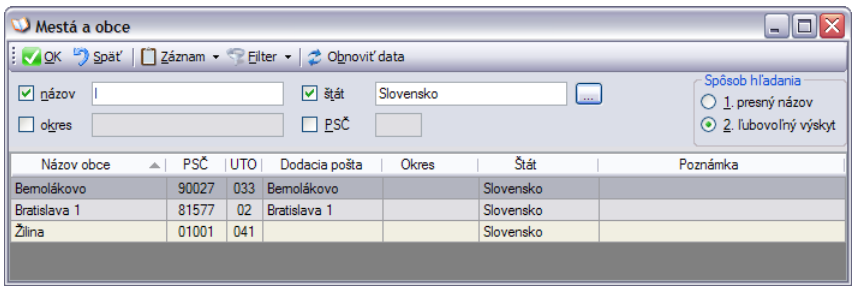
[6] Prosise J.: *Programming Microsoft .NET*, Microsoft Press, Redmond, 2002

[7] Zákon o účtovníctve 431/2002 Z.z.
`http://www.minv.sk/legislativa/431uctovnictvo.htm`

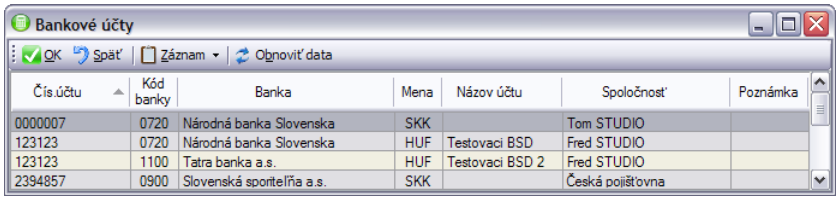
Dodatok A

Prílohy

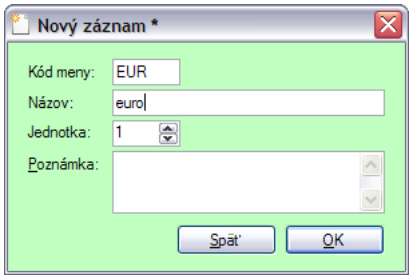
Obrázok A.1: Okna s filtrom



Obrázok A.2: Okna bez filtra



Obrázok A.3: Dialógové okno



Obrázok A.4: Práca s programom