Bakalářská práce na téma

Replikace dat

Autor: Richard Kocman

Vedoucí bakalářské práce: Ing. Václav Novák CSc.

Rok: 2007/2008

Katedra Informatiky Pedagogické fakulty Jihočeské univerzity
v Českých Budějovicích

Prohlášení

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

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

V Českých Budějovicích dne

Anotace

Tato práce se zabývá problematikou replikace dat za pomoci moderních technologií, jako jsou Microsoft .NET Framework 3.5, Microsoft Sync Framework a nástroj pro tvorbu programů Microsoft Visual Studio 2008. Kromě teoretické části obsažené v práci, zde uvádím také několik názorných příkladů způsobů řešení synchronizace dat, a po seznámení čtenářů s moderními technologiemi replikace dat, překládám návod k vytvoření funkční aplikace za použití Visual Studia 2008, který zefektivní a zjednoduší vývoj aplikace.

Abstrakt

This work deals with the problems of replication of the data with the assistance of modern technologies such as Microsoft .NET Framework 3.5, Microsoft Sync Framework and even the tool for a program production Microsoft Visual Studio 2008. In addition to the theoretical part, which is also included, you can find here some illustrative examples as well and after getting acquinted with the modern technologies we'll create a functional application. During its development we are using the tools of Visual Studio 2008 for a greater efficiency.

Obsah

1 Úvod

1.1 K samotné práci

1.1.1 Replikace dat a její využiti

V dnešní době je při využívaní moderních počítačových technologií a internetu uchováváno na pevných discích počítačů a jiným způsobem zálohováno obrovské množství dat. Tato data, která pro nás mají hodnotu, je nutné bezpečně uchovávat a chránit, abychom nebyli vystaveni nebezpečí jejich ztráty či zničení.

Další nutností pak může být možnost využívat přístupu k datům na více místech najednou. Zvlášť pokud jsme obchodními zástupci potřebujícími mobilních zařízení při zaměstnání, spravování internetového obchodu či organizací s velkým množstvím poboček, využívajících jednu velkou společnou informační síť.

V jakémkoli z výše uvedených případů využíváme replikaci dat. Například při tvorbě záloh, při stahování dat z centrální databáze do lokální, nacházející se na našem klientu, kterým může být osobní počítači, Smartphon, Pocket PC a jiné mobilní zařízení.

1.1.2 Význam a cíl bakalářské práce

Tato bakalářská práce si klade za cíl seznámit čtenáře s problematikou replikace dat takovým způsobem, aby mohl jako začínající programátor „nakouknout pod pokličku“ této tvorby. Po seznámení s teorií jsme tedy postaveni před úkol, vytvořit jednoduchou, ale funkční aplikaci. Při programování využijeme nejnovějších, a jak se přesvědčíme vysoce efektivních nástrojů, jež jsou součástí Visual Studia 2008.

Jako student se v tomto konkrétním případě setkávám s nedostatkem tištěných nebo elektronických materiálů, ze kterých bych mohl načerpat potřebné znalosti, protože se jedná a nástroje poměrně nové a manuály k nim nejsou pro začátečníky svou složitostí vhodné. Na internetu je pak možné najít několik zmínek a rad, ale ty dle mého názoru zcela vůbec nejsou ucelené a pro čtenáře, kterým se chce problematikou začít zabývat, nejsou nejvhodnější.

1.1.3 Členění práce

Samotná práce je členěna do několika částí. Na úvod se seznámíme se základním kamenem práce s daty, a to jazykem XML. Dozvíme se, jaké výhody má, v čem spočívá jeho kvalita, pro niž je podporován ve většině moderních jazyků a ve světě internetu. Dále si vysvětlíme pojem webová služba, jež při replikaci dat slouží spíše jen jako jejich zdroj. Nezapomene se zaměřit na teoretickou stánku replikace dat, a pustíme se i do seznamování s moderními technologiemi Sync a .NET Framework za pomocí Visual Studia.

Závěrem si dovolím shrnout obsah práce, a také uvádím seznam použité literatury a obsah přiloženého CD.

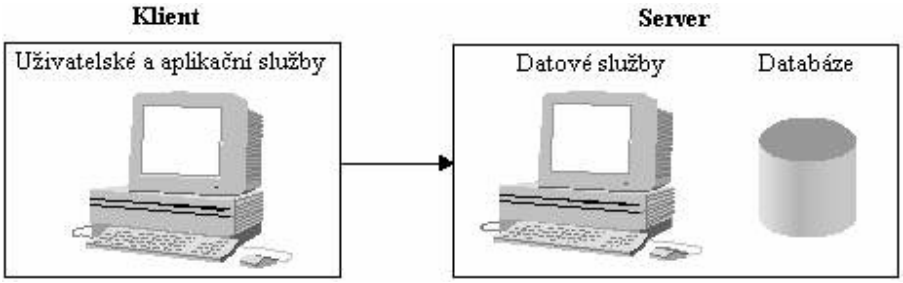
2 Síťová architektura klient - server

2.1 Klient a server

2.1.1 Fungování a teorie

Architektura klient - server je poměrně rozšířená síťová architektura, ve které klient a server jsou spojeni určitou síťovou technologií. Uzly sítě, anglicky nodes, pak označujeme servery a klienty, jež v síti můžeme najít. Dále rozlišujeme dvojvrstvé a vícevrstvé architektury.

V dvojvrstvé architektuře se nachází několik klientů komunikujících se serverem. Ve více vrstevné architektuře pak kromě několika klientů nacházíme i různé druhy serverů. Např. aplikační a databázové. Ty první mají na starost zpracovávat zprávy s požadavky, které od klientů obdrží, kdežto ty databázové slouží k uchovávání dat jako „datová skladiště“.



Samotné fungování architektury klient server není složité. Klienti posílají serveru žádosti a čekají od něj odpověď, jež od serveru obdrží, poté co jejich žádost zpracuje. Samotní klienti velmi často obsahují i grafické prostředí pro komunikaci s uživatelem a jsou připojeni k jednomu serveru, jenž obsluhuje více klientů.

3 XML neocenitelný pomocník

3.1 XML - o co se jedná?

3.1.1 Skvělá myšlenka

XML formát jako takový má v dnešní době velké možnosti využití v různých počítačových technologiích pracujících s daty, a to především jako kvalitní otevřený formát pro výměnu informací. Jeho základní myšlenkou je použití textového formátu, což je velkou předností a nemalou výhodou. Díky tomu je srozumitelný jak pro člověka, tak pro počítač. Mezi jeho další plus patří možnost použití téměř jakéhokoli světového jazyka díky kódování UNICODE.

3.1.2 XML a jeho syntaxe

Jazyk HTML, používaný při tvorbě internetových stránek je jazykem, jemuž se XML podobá. V souboru ve formátu XML se nachází atributy, elementy a hodnoty, které slouží k libovolné definici datových typů a různé struktuře dat. Struktura dokumentu XML se neobejde bez dokumentové hlavičky umisťované na jeho začátek. Někdy je označujeme jako XML deklarace. Určuje typ kódování dokumentu a jeho XML verzi. Dále pak jednoho tzv. kořenového elementu.

V XML dokumentu se mohou objevit také komentáře <! -- -- ->, prázdné elementy <videliJsteNekdySvestu = "Ano"/>. XML schémata pak mohou strukturu samotného dokumentu popisovat. Ve své podstatě nejsou zase nic jiného, než další XML dokument obsahují přesný popis struktury XML dokumentu, jež bývá podle nich formátován. Slouží tak k validaci syntaxe samotného dokumentu a jeho konkrétního obsahu. Jakýmsi předkem XML schémat, z nichž i vycházíjí, je tzv. DTP, Dokument Tipe Definition, jež se používalo u XML verze 1.0. Jeho velkou nevýhodu však byla nemožnost definice vlastních datových typů a proto se již nepužívá.

```
<?xml version="1.0" encoding="Windows-1250"?>
<!-- CEdecko --!>
<cedecko>
        <titul>Automatic for the people</titul>
        <interpret>R.E.M</interpret>
        <cena>560</cena>
</cedecko>
```

3.2 Práce s daty s pomocí XML

3.2.1 XML

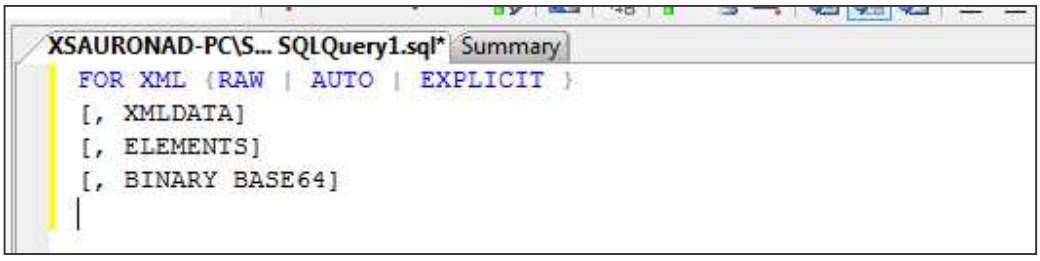
XML je pro svůj ideální formát skvělým nástrojem pro výměnu informací mezi aplikacemi a webem, jak může vidět v mnoha programech, a často i bohaté podpoře v programovacích jazycích a vývojových prostředích.

V této části práce bych rád probral celou problematiku použití XML a jeho podpory na straně SQL serveru pokud např. programujeme naší aplikaci v C#. Jak uvidíme, SQL databáze společně s XML soubory mohou být ideálními nástroji a velkými pomocníky při replikaci dat.

Řekneme si něco objektech XmlDocument a XmlDataDocument i jak jejich pomocí uložit soubor ve formátu XML. Nebo od SQL serveru získat data v XML formátu a dále možnost jak pomocí webu přistupovat a klást dotazy SQL serveru.

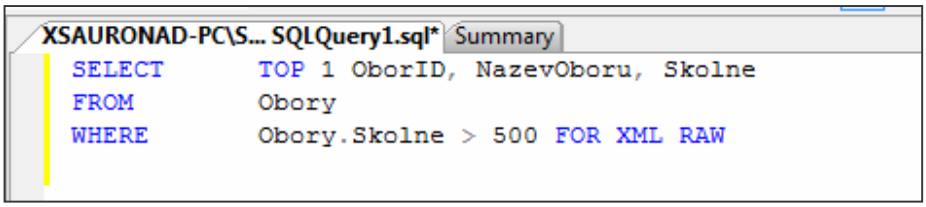
3.2.2 SQL server komunikuje v XML

Získat data od SQL serveru se nám podaří tehdy, pokud mu položíme správný dotaz. SQL server nám nabízí možnost vracení dat ve formátu XML. Vše co musíme udělat, je jen se správně zeptat. Klíčovým slůvkem je v tomto případě výraz FOR XML, jež má následující syntaxi:



3.2.2.1 FOR XML RAW

V hranatých závorkách najdeme klíčová slova určující mód zápisu XML. Při použití módu RAW se dočkáme XML souboru, v němž řádky výsledného dotazu budou v elementu RAW a sloupce budou atributy. Např. na SQL dotaz, který se ptá jaký první obor z tabulky Obory (obsahující informace o oborech nějaké školy) má školné vyšší než 500?



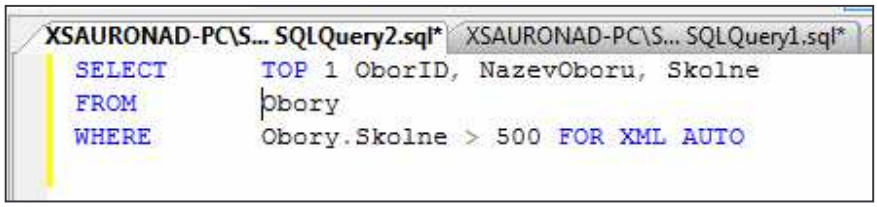
Dostaneme následující odpověď:

```
<row OborID="1025Q"NazevOboru="Mechanik" Skolne="1000"/>
```

3.2.2.2 FOR XML AUTO

Mód AUTO nám pomůže nastavit výpis tak, že SQL server vrátí každý řádek opět jako element a název tabulky je použit jako název elementu řádku.

Např. na dotaz:



Dostaneme odpověď:

```
<Obory OborID="1025Q" NazevOboru="Mechanik" Skolne="1000"/>
```

Je tedy jasně vidět, že značku < row > nahradil název tabulky.

3.2.2.3 FOR XML EXPLICIT

Další mód, podle kterého můžeme formátovat XML výstup, je mód EXPLICIT. Můžeme pomocí něho zajistit, že příkaz SELECT bude obsahovat relační vztah nadřízený - podřízený. Pokud chceme tento mód použít, musíme také použít dva řádky SELECT.

První příkaz určuje řádky nadřízené a druhý podřízené. Řádky, které jsme načetli pomocí těchto příkazů, jsou spojeny speciálními sloupci Tag a Parent. První z nich nastavuje číselnou hodnotu elementu a druhý říká, jaké číslo Tag má nadřazený element.

3.2.2.4 XMLDATA

XMLDATA je další položkou, která dokáže určit, aby definice XML schématu byla vrácena společně s XML zápisem. Obsahuje tedy názvy a hlavně typy jednotlivých atributů sloupců.

3.2.2.5 ELEMNETS

Pokud používáme mód Auto, je možné využít i další položku a to položku elements. Díky ní se nám hodnoty jednotlivých sloupců vracejí jako podelementy řádku a ne jeho elementy.

3.2.2.6 BINARY BASE 64

Poslední volitelnou položkou je položka Binary Base64. Může se nám hodit tehdy, pokud chceme získat pomocí XML informace v binární podobě.

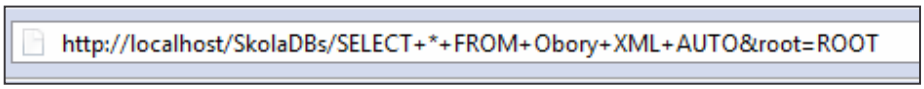
Když nepoužijeme Binary Base64, v XML výstupu najdeme jen výraz jazyka XPath, který místo toho odkazuje na binární data.

Práci s XML nám může výrazně ulehčit právě jazyk Xpath, pomocí kterého můžeme efektivně prohledávat a procházet XML dokument. Více informací si však neřekneme, protože to přímo s naším tématem nesouvisí.

3.2.3 XML, HTTP a SQL server

Existuje možnost přistupovat k SQL serveru pomocí prohlížeče. Můžeme tak nejenom vkládat dotazy, ale vkládat a upravovat data. SQL server musí být samozřejmě správně nastaven, aby něco takového povoloval. Při práci s vrácenými daty a při jejich pak můžeme využít výše zmíněný jazyk XPath, či formátovat výstup v internetovém prohlížeči pomocí XSL šablon.

Samotné zadání příkazu není nikterak složité. Provedeme jej pomocí URL adresy, jež příkaz obsahuje. Například:

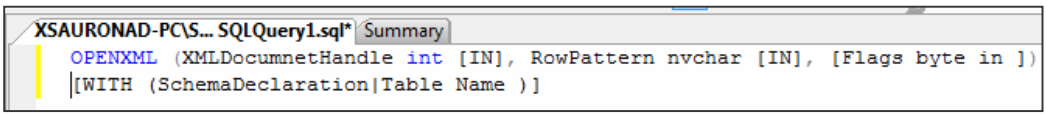


Místo mezer musíme používat znak plus. Parametr root pak slouží jako jméno kořenového elementu, ale můžeme používat cokoli. Pokud je vše správně nastaveno, zobrazí se nám v prohlížeči výsledek našeho dotazu.

Kromě mezery, které nelze v URL adrese použít, se můžeme setkat ještě s jinými znaky. Uvádím je i s náhradními znaky /- %2F, ?- %3F, %-%25, #-%23, &-%26.

Například pokud chceme použít v našem SQL příkazu tento výraz LIKE ‘C%’, pak napíšeme: LIKE+’C%25’.

Při práci s databází můžeme použít také funkci OPENXML(). Díky ní můžeme jednoduše načítat data z XML souboru, a získáme sadu řádků, které lze přímo ukládat do tabulek. Funkce OPENXML() má tuto syntaxi:



3.2.3.1 XMLDocumnetHandl

Je celočíselným ukazatelem na dokument a používá se jako odkaz na něj.

3.2.3.2 RowPattern

Tento výraz v jazyku XPath ukazuje na data v dokumentu, jež nás zajímají a chceme je načítat.

3.2.3.3 Flags

Jedná se o volitelnou položku typu byte, díky níž můžeme nastavit mapování mezi XML daty a hodnotami sloupců v databázi. Může nabývat hodnot 1 a 2. První hodnota nastaví mapování tak, že hodnoty sloupců se nachází ve vnořených atributech uzlů v XML dokumentu. Druhá pak říká, že hodnoty sloupců najdeme jako samostatné vnořené elementy.

3.2.3.4 SchemaDeclaration

Jak již název napovídá, pomocí této položky nastavíme schéma databáze, do které budeme vkládat data z XML dokumentu.

3.2.3.5 TableName

Určuje jméno tabulky, do níž budeme vkládat data. SchemaDeclaration používáme tehdy, pokud pracujeme s tabulkou, jež v databázi není. Pokud je, využíváme spíše položku TableName.

3.2.4 C#, Data a XML Dokument

Pokud při programování s NET.Framework píšeme kód v jazyku C#, můžeme využít objekt třídy XmlDocument. Tento objekt ve svých atributech typu XmlNode uchovává uzly XML. Pokud kupříkladu ve své aplikaci pracujeme s databázemi, anebo data jen uchováváme v objektech typu DataSet, můžeme jejich zápis pomocí tohoto objektu uložit.

Celá operace by mohla vypadat asi takto. Do objektu typu DataSet pomocí SqlDataAdapteru nahrajeme obsah nějaké tabulky z databáze. Data z DataSetu pak získáme pomocí metody GetXml(), jejíž výstup použijeme jako vstupní řetězec pro metodu LoadXml() objektu XmlDocument. Jeho XML obsah může uložit pomocí metody Save().

Ještě se zmíníme o třídě XmlDataDocument. Objekt této třídy můžeme propojit s objektem typu DataSet, pokud jej použijeme jako parametr v jeho konstruktoru. Pak můžeme požívat tento objekt k přístupu k objektům DataRow a XmlNode a XmlDataDocument provádí mezi těmito objekty synchronizaci. Vložíme li kupříkladu nová data ve formě objektu XmlNode do XmlDataDocument, stejný je vložen i do objektu DataSet.

4 Webové služby

4.1 Silný nástroj

4.1.1 Webová služba

Pokud mluvíme o řešeních typu klient - server s využitím XML zpráv, tak se jedná o technologie označované souhrnným názvem webové služby. Technologie webových služeb je rozšířením klasické RPC. Komunikace mezi klientem a serverem používá zprávy ve formátu XML. Klientem může být v tomto případě libovolný webový prohlížeč, aktivní webová stránka, desktopová aplikace a další. Serverem může být téměř jakýkoliv webový server.

Klient odešle zprávu ve formátu XML na server, ten ji zpracuje a v případě, že je potřeba odpovědět, odešle odpověď ve formátu XML. Formát XML zpráv je v případě webových služeb přesně definován. Stejně tak jsou přesně definována rozhraní, přes která se k webovým službám přistupuje. K definici webových služeb se používá takzvané WSDL.

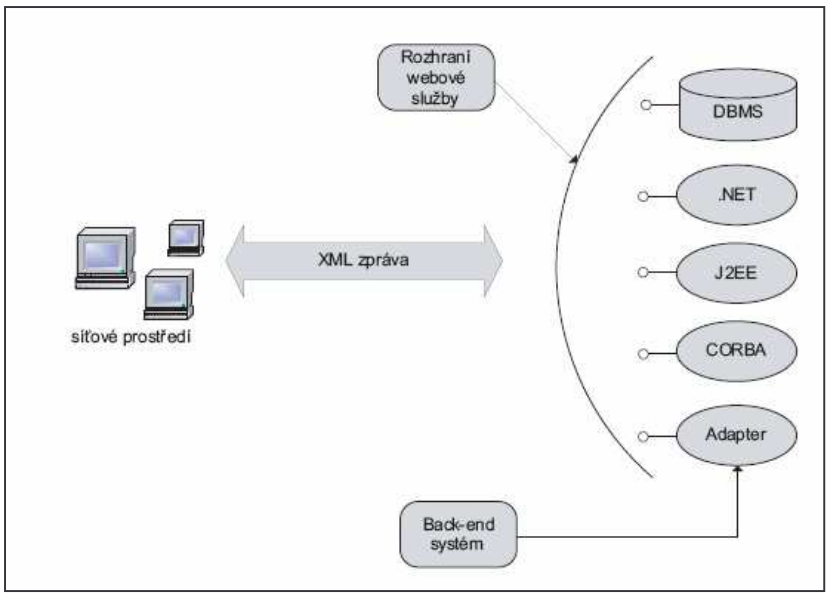
4.1.2 Publikování webových služeb

Pro vyhledávání a publikování webových služeb slouží takzvané UDDI. V případě webových služeb musí být přesně definované konverze datových typů. A to z toho důvodu, aby bylo možné ke službám přistupovat pomocí libovolného programovacího nástroje nebo rozhraní.

4.1.3 Webová služba – různé technologie

Webové služby jsou díky použití univerzálního XML formátu k výměně zpráv ideální ke kombinaci různých technologii. Jak ukazuje obrázek níže, základní myšlenkou bylo umožnit pomocí jednotné formy volání přístupu k různým back-end technologiím.

Tyto back-end technologie mohou být například relační databázové systémy, NET platforma, J2EE platforma, CORBA a mnoho dalších. Stačí nám k tomu pouze rozhraní pro webové služby, které leží mezi klientem a vlastním back-end systémem. Toto rozhraní může být vytvořeno v libovolném programovacím jazyce na libovolném operačním systému. Zde je již jasné, že webové služby nenahrazují samotný programovací jazyk, ale pouze k němu poskytují jinou formu přístupu. V případě webových služeb jde o unifikovaný přístup pomocí XML zpráv.



Díky tomu lze například i starší aplikace rozšířit o nové rozhraní pro podporu přístupu pomocí technologie webových služeb, a přitom ponechat původní kód a funkčnost programu.

4.1.4 Přenos dat - replikace a webové služby

Webové služby dělíme do dvou skupin, a to RPC orientované a dokumentově orientované. RPC orientované vychází z principu klasického RPC, ve kterém se pracuje se zprávami. Dokumentově orientované jsou takové, které jako základ používají dokumenty ve formátu XML. Tyto dokumenty jsou celé odeslány na server a na serveru je zpracováván celý tento dokument, než se nakonec vrátí uživateli.

Dokumentově orientovaný přístup si můžeme představit jako přihlášku do soutěže. Ta je celá zpracována, a výsledek nám přijde za několik dnů zpět. V praxi se ovšem jedná pouze o rozdíl ve voláních. Při RPC volání má zpráva přesně definovaný tvar, ale při dokumentově orientovaném volání může být XML zpráva v podstatě libovolně formátované XML.

Při získávání dat a při jejich replikaci je tedy možné webové služby používat jako zdroje informací. V ideálním případě i ve formátu XML, pro jejich rychlé a efektivní zpracování.

5 Teorie Replikace

5.1 Replikace – duplicita dat

5.1.1 Data v bezpečí

Často uchovávame data na více místech najednou z potřeby zálohy, tedy ochrany, či z nutnosti rychlého přístupu. My se budeme zabývat převážně problematikou databází a jejich replikací. Databáze jaké si datové sklady, jsou často umísťovány na síťové servery. Velkou výhodou toho umístění je snadná dostupnost pomocí různých síťových prostředků.

I zde však můžeme narazit, protože nám hrozí nebezpečí ztráty dat v případě výpadku nebo dokonce jejich zničení, jestliže je server vážně poškozen. Proto pokud chceme toto nebezpečí minimalizovat, uchováváme data, v tomto případě na serverech v sítích. Také musíme zajistit aktuálnost a správnost dat na více serverech, což znamená zajistit jejich vzájemnou komunikaci a synchronizaci dat, což není zrovna jednoduchý úkol.

Replikace je jedním ze způsobů, jak zajistit duplicitu dat. Přináší nám výhodu z lokalizačního hlediska (výhodná vzdálenost klienta od serveru s přihlédnutím k intenzitě komunikace a požadované rychlosti).

Distribuované datové prostředí, tedy jiný termín pro duplicitu dat, s sebou nese několik nemalých výhod:

1. Nezávislost jednotlivých datových serverů
2. Separace OLTP (Online Transaction Processing -, tj. online transakční zpracování s poměrně velkým poměrem aktualizací) a samotných datových skladů (zde pak s vysokým výskytem operace čtení).
3. Přiblížení dat uživateli a s tím i zvýšení rychlosti a možnosti snížení finančních nákladů.
4. Snížení konfliktů při aktualizaci, mazaní nebo čtení dat.

5.1.2 Distribuované transakce

Dalším možným přístupem k zajištění distribuovaného datového prostředí mohou být tzv. distribuované transakce. Oba dva způsoby jsou založeny na zcela jiném filozofickém systému, často se však v praxi z hlediska uplatnění překrývají. Jak replikace, tak distribuované systémy udržují vícenásobné kopie dat aktuální. Oba způsoby se však liší v časové synchronizaci..

Důležitou vlastností distribuovaných systémů je garance konzistentnosti uchovávaných dat v čase. Což s sebou však nese několik nevýhod.

Mezi hlavní nevýhodu tohoto řešení patří především nutnost všech serverů s daty být online, protože transakce se provádí na všech serverech najednou, nebo se neprovede vůbec. Je tedy jasné, že metoda je náchylnější k chybám, a chyba například s připojením u jednoho ze serverů zapříčiní problém u všech serverů, a tedy i stornování transakce. Proto se její využití omezuje jen na případy, kdy data musí být neustále synchronizována. Ne moc šťastným řešením by pak mohla být komunikace počítačů po internetu. Využití bychom mohli najít spíše ve vnitropodnikovém intranetu s vysokorychlostní sítí.

5.1.3 Replikace

Časovou synchronizaci, kterou používají distribuované transakce, replikace nevyžadují. Servery fungují zcela samostatně a provádí bez omezení všechny operace mazaní, vkládání a aktualizace. Samotná synchronizace dat se provádí až tehdy, kdy je vyžádána. Což může šetřit síťové prostředky a provádět se tak, aby synchronizace nenarušovala standardní chod sítě. Až při synchronizaci pak dochází k případným řešením problémů a ke konfliktům, vzniklým např. při aktualizaci totožných dat. Příkladem pak může pro nás být evidence přihlášených uživatelů na PC při školeních, když nemůžeme zaručit stále síťové připojení. Data se ukládají lokálně a po připojení se odesílají na server.

5.2 Replikace na serveru

5.2.1 Terminologie replikace

Jako nejlepší způsob přiblížení terminologie replikace může být analogie publisher (vydavatel) - distributor - subscriber (odběratel). Úkolem vydavatele je uchovávat zdrojovou databázi, tedy tu, která bude replikována. Změny v ní pak posílá vydavatel distributorovi. Distributor tyto změny v datech uchovává a nabízí odběratelům. Změny může

distributor posílat sám tzv. Push odběr, nebo si jej vyzvedávají odběratelé Pull odběr. Odběratel pak má kopii dat a i přijímá případné změny. Je také možné nastavit vše tak, aby odběratel data po příjmu modifikoval a posléze se stal vydavatelem pro jiného odběratele.

Tyto všechny části se mohou nacházet na jednom serveru. Otázkou pak zůstává, zda je tento způsob dostatečně praktický. Spíše využíváme možnost, kdy se každá část (odběratel, vydavatel, distributor) umisťuje na svůj vlastní server. Možných kombinací je však samozřejmě více. Kupříkladu není problém, pokud bude vydavatel a distributor na jednom serveru, ale nesmí hrozit nadměrné zatěžování serveru. Pokud by tomu tak bylo, vyplatí se vyhrazení dalšího serveru pro distributora. Pak tohoto distributora lze využít i na jiné vydavatele, kteří jsou součástí jiných replikacemi.

Pokud se nám zalíbila terminologie vydavatel, distributor, odběratel, zmíníme se ještě o dvou termínech. A to tzv. článcích a publikacích.

5.2.1.1 Článek

Článek je základním kámenem samotné replikace. Nese datový element, jež potřebujeme přemístit. Např. celá tabulka, řádek databáze, vybrané sloupce, vestavěná procedura nebo její vykonání.

5.2.1.2 Publikace

Představuje souhrn jednotlivých článků, které byly odebrány najednou. Můžeme ji tedy brát jako jednotku dat, obsahující kompletní data potřebná pro nějakou aplikaci. Zpravidla tedy v jeden konkrétní čas nemusí odběratel odebírat více než jednu publikaci od vydavatele. V publikaci se mohou nacházet jen data z jedné databáze, ale ta může obsahovat několik publikací.

5.2.2 Filtrování dat

SQL Server při replikaci dat nabízí i možnost tzv. filtrování. Jde o to, že je možné vybrat jen některé řádky, to jsou tzv. restrikce, nebo zvolíme horizontální filtrování. Při výběru sloupců (vertikálního filtrování) pak projekce. Můžeme použit obou možností současně. Pomocí horizontálního filtrování dat je možné zajistit jejich konzistenci, pokud je umožněno, aby se jednotlivé kopie replikace dat směly aktualizovat na více místech. Tímto předejdeme konfliktům při slučování dat. Jinak každá instance filtrování vytvoří samostatný článek.

Jistou alternativou za horizontální filtrování může být tzv. partitioning. Tato metoda vypadá tak, že se vytvoří více tabulek se stejnou strukturou, přičemž každá obsahuje určitou oblast dat. Tato metoda má ale i své nevýhody. Aplikace pracuje s více tabulkami a administrace dat je o dost náročnější.

5.2.3 Odběry

Za pojmem odběr můžeme vidět konfiguraci toho, jak odběratel získá publikaci od vydavatele. V praxi se můžeme setkat s velkým množstvím kombinací odběrů, ale my si zde povíme od dvou zásadních od sebe se lišících.

5.2.3.1 Push odběr

Push odběr, někdy také nazývaný vynucený odběr, je nadefinován na straně vydavatele. Každá publikace může mít nastaveno více odběratelů najedenou. Publikace jsou zasílány vydavatelem odběratelům jako nevyžádané. Proto tedy vynucený odběr. Tohoto typu odběru se využívá například tehdy, pokud bude u vydavatele provedena změna dat, jež je nutné dále distribuovat, a pokud dočasně zvýšené zatížení serveru vydavatele nehraje takovou roli. Replikační agenty i při tomto typu odběru můžeme najít u vydavatelů nebo distributora.

5.2.3.2 Pull odběr

Někdy také překládán jako odběr na vyžádání. Zde bychom toho v režii odběratele našli daleko více. Odběratel má u sebe pull odběr definován, spouští jej, rozhoduje také o samotných publikacích, které od vydavatele chce získat, ale i o čase, kdy mu budou zaslány. Aby tento typ odběru byl možný, musí jej publikace povolovat. Tento typ odběru je preferován tam, kde není tak důležitá bezpečnost a je upřednostňována větší volnost odběratelů. Využití samozřejmě nachází třeba na internetu. (Např. je možné využívat FTP protokol.) Zde se k nim může dostat velké množství odběratelů a využijeme je tehdy, pokud je chceme co nejvíce zpřístupnit. Vyžádané odběry pak ještě můžeme dělit na anonymní a registrované. Odběry anonymní jsou zcela nastaveny u počítačů odběratelů, kdežto registrované částečně i na straně vydavatelů. Replikační agenti jsou spuštěni na serveru odběratele.

5.3 Typy replikací

Samotný typ replikace se týká vždy konkrétní replikace, z čehož vyplývá, že typů replikací v jedné databázi může být i více. Proces implementace replikačních typů se nazývá replikační agent, a dále se o něm zmíníme.

5.3.1 Snímková replikace

U tohoto typu replikace jsou všechna kompletní data použita k nahrazení dat na cílovém počítači buďto na základě požadavku, nebo periodicky. Z toho u snímkové replikace vyplývá několik závěrů. Jedná se vlastně o fyzické kopírování dat platných v daném okamžiku.

1. Může docházet k poměrně vysokému stupni zpoždění.
2. Stupeň autonomie serverů je vysoký.
3. Replikované tabulky nepotřebují primární klíče.
4. Zátěž procesoru je menší, protože není nucen stále monitorovat transakce.
5. Nehodí se pro velmi rozsáhlé transakce.

Tzv. snímkový agent pročítá databázi vydavatele a snímky ukládá v souborech v distribuční pracovní složce na serveru distributora. Do distribuční databáze u distributora se neukládají data, ale pouze stavové informace. Distribuční agent pak zajistí, aby se data zkopírovala od distributora odběrateli. Agent běží buďto na distributorovi nebo na odběrateli v závislosti na tom, o jaký odběr se jedná. (push odběr – distributor, pull odběr – odběratel)

5.3.2 Transakční replikace

U tohoto typu replikace se distribuují pouze změněná data. Ke změně dat tak dochází pouze na jednom místě, a to u vydavatele. Díky tomu nedochází k žádným konfliktům. Z toho u transakční replikace vyplývá několik závěrů.

1. Většinou nedochází k velkým zpožděním.
2. Nižší stupeň autonomie serverů. (pokud chceme nízké zpoždění).
3. Replikované tabulky se neobejdou bez primárních klíčů.
4. Výhodného publikace povolné velikosti.
5. Procesor musí neustále monitorovat transakce v replikovaných tabulkách. A jen transakce, jež byly úspěšně provedeny a v jistém pořadí aplikovány na vydavateli, budou distribuovány v tomtéž pořadí.

Nejdříve je provedena snímková replikace. Soubory zkopírované pomocí snímkového agenta využije distribuční agent k založení transakční replikace na odběrateli. Transakční replikace pak pomocí Log leadera čte transakční log u vydavatele a ukládá informace do distribuční databáze. Distribuční agent se opět stará o přenos těchto změn na odběratele. Změny se odehrávají na distributorovi nebo na odběrateli v závislosti na typu odběru.

5.3.3 Možnost okamžitých změn

Obvyklým předpokladem je, že data odběratelé nebudou aktualizovat data. Předpokládá se tedy, že data potečou směrem od vydavatel k odběratel. Nikoli naopak. A to u obou výše zmíněných replikací. Nebo pomocí výše zmíněného partitioningu dojde k vytvoření několika tabulek s různými oblastmi. A to tak, že každá oblast je vydavatelem své vlastní tabulky a odběratelem tabulek z jiných oblastí. Je zde však ještě jedno

řešení. Řešením je možnost okamžitých změn odběratelem. Při této možnosti přichází na řadu dvoufázový potvrzovací protokol.

Odběrateli se pak nabízí možnost libovolně modifikovat data, ale jen tehdy, pokud budou okamžitě modifikována i u vydavatele. Protože se jedná o replikaci a ne distribuovanou transakci, bude změna k ostatním odběratelů zaslána až při příští replikaci. Jak tedy vyplývá z výše zmíněných poznatků, toto řešení je závislé na spolehlivém spojení, protože pokud není zdárně provedena aktualizace u vydavatele, musí být celá operace zrušena i na straně odběratele, jež transakci vytvořil.

Ale je třeba si ještě uvědomit a dát si pozor na jednu situaci. Aktualizace se nezdaří, pokud aktualizovaná data nejsou totožná u odběratele a vydavatele. Může k tomu dojít tehdy, pokud už se provedla změna záznamu u vydavatele nebo již jiný odběratel provedl změnu pomocí okamžitých změn. K této situaci může dojít u snímkových replikací, kdy dobrým preventivním řešením je partitioning. U transakčních replikací k této situaci téměř nedochází, protože data jsou posílána na všechny odběratele téměř okamžitě.

5.3.4 **Slučovací replikace**

Slouží ke změně dat na různých serverech. Ke slučování dochází na vyžádání, a nebo ve stanovený čas. Konflikty se pak řeší pomocí uživatelsky nastavitelného mechanismu. Tento typ replikace je vhodný pro data, která jsou nějakým způsobem segmentována, ale nezaručuje transakční integritu.

1. Zpoždění může být jakékoli
2. Servery mají vysokou autonomii
3. Je vhodného pro publikace libovolné velikosti
4. Musí monitorovat transakce v replikovaných tabulkách
5. V replikovaných tabulkách se při vytváření slučovací replikace generují spouště, označující data k replikaci.
6. Nepodporuje vertikální filtrování
7. Nezaručuje transakční konzistenci, ale zaručuje, že všechny tabulky budou mít podobný obsah.

Celý proces začne snímkovou replikací a zkopírované soubory použije slučovací agent k založení slučovací replikace na odběrateli. Slučovací agent se dále stará o přesun změn na odběratele, který běží na odběrateli, či distributorovi v závislosti na typu odběru. Stavové informace jsou uchovávány distribuční databázi. Slučovací agent pracuje se změnami z vydavatele a předá je odběrateli. Pak aplikuje na vydavatele data od všech odběratelů a řeší případné konflikty.

6 Replikace dat s .NET Frameworkem

6.1 Nekontinuální síťové spojení, replikace dat - řešení

6.1.1 Práce s daty

Pokud pracujeme s daty a používáme k tomu převážně databázové aplikace. Tyto aplikace jsou dosti závislé na kvalitním síťovém spojení proto, aby servery mohly mezi sebou data vyměňovat. Může se dostat do situace, kdy síťové připojení není realizovatelné. A to například z důvodu nekvalitního spojení, jež má za následek jeho výpadky, nebo pro potřeby offline práce mimo dostupnost sítě (zaměstnanci se pohybují v terénu) a pracují s daty na mobilních zařízeních.

Pokud chceme vytvářet aplikace pracující s daty, které nejsou stále připojené k centrální databázi, a mají tedy nestálý, občasný databázový přístup, musí tyto aplikace mít lokální databázi, jež bude fungovat i jako vykoná cache. Po úspěšném navázání spojení dojde pomocí synchronizačních služeb k synchronizaci dat s centrální databází.

6.1.2 Co nabízí .NET Framework a Microsoft

Pokud budeme uvažovat o výše zmíněném scénáři, a budeme jej chtít aplikovat do praxe, nabízí několik kvalitních nástrojů k řešení platforma .NET. Tím prvním je prostředí .NET Compact Framework 3.5 určen pro mobilní zařízení a s ním i mobilní SQL Server 2008 Compact Edition 3.5. Tato verze SQL serveru se dá využit, jak u mobilních aplikací tak i u desktopů. Pokud neklademe moc vysoké požadavky na server.

Compact Framework 3.5 Mobile obsahuje podporu, programovací ho jazyka LINQ, který může efektivně sloužit k přístupu k databázím. V mobilní verzi Compact Frameworku 3.5 je podporován LINQ to Object, LINQ to XML a LINQ to DataSet. LINQ to Entities není zatím podporováno. Ve mobilní verzi Frameworku lze pak najít celou řadu nových nástrojů CLR Profiler, který umožňuje profilovat .Net kód, pracovat s certifikáty na straně klienta, využívat možnosti komprese.

Při samotném programování oceníme také Visual Studio 2008, díky němuž dokážeme velice efektivně využít možnosti Frameworku 3.5 a SQL Serveru.

6.1.3 SQL Server Compact Edition 3.5

Tento malý integrovaný server dokáže spravovat databázi až o velikosti 4GB. Tato edice je nejen přístupná pro mobilní zařízení typu Smartphone a Pocket PC, ale lze ji instalovat i na desktopy a notebooky. SSCE 3.5 využívají například aplikace jako Windows Media Player 11 library nebo Windows Media Center.

Verze SSCE 3.5 nám umožňuje používat T-SQL dotazy i dotazy vnořené v klauzuli SELECT FROM. Pro práci se SQL Serverem můžeme využít Visual Studio 2008, v němž je možné provádět většinu důležitých operací, jež se nám při práci s databází hodí. Můžeme vytvářet tabulky, pracovat s daty, vytvářet připojení mezi naší aplikací a serverem. To vše v přehledném grafickém režimu.

Pokud chceme přistupovat k datům, můžeme využít například SqlCeDataProvider pro ADO.NET. Ten obsahuje SqlCeDataReader, DataSet, LINQ nebo SqlCeResultSet, jež je nejméně náročný na využití paměti. Tento objekt k datům přímo přistupuje a nepoužívá žádný buffer. SQLCE může být nainstalován zvlášť na cílovém zařízení. (ne současně s aplikací) Pak může být SQLCE aktualizován nezávisle na aplikaci. Nebo aplikace kromě dat obsahuje i jeho knihovny. Pak je ale SQLCE součástí aplikace, a po její odinstalaci je odinstalován i on. Aktualizace musí být prováděny v rámci aktualizace. Pokud již máme malý lokální

server, musíme nějakým způsobem provést aktualizaci s „velkým“ SQL serverem. Těch možností je několik.

První možností je využití slučovací replikace, kde je nutné správně nastavit pravidla a možnosti synchronizace. Pokud využíváme SSCE 3.5 , je možné využít replikaci jen s MS SQL serverem.

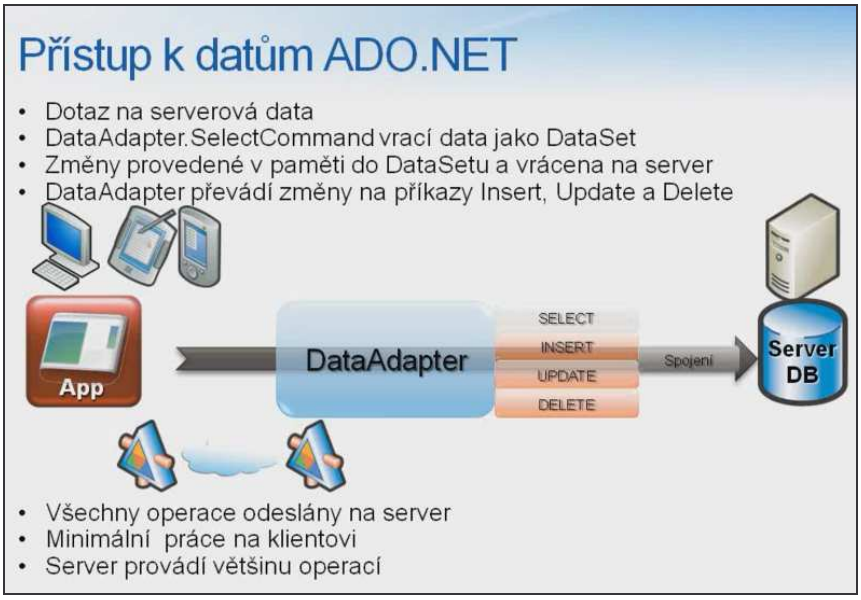
Další možnosti přenosu a replikování dat je tzv. Remove Data Access. Ten využívá přístup přes ISS server. Musíme však počítat s omezenými možnostmi pro škálovatelnost řešení a pro distribuované řešení více aplikací.

Novinkou pro práci při přenosu dat je služba Sync Services pro ADO.NET. Vývojář používá pouze runtime pro synchronizaci a provider pro ADO.NET, který zajišťuje synchronizaci dat mezi lokálním a velkým SQL serverem.

Platforma je založena na synchronizačních poskytovatelích, což jsou komponenty, různé distribuční kanály a technologie pro různé typy informací a zpráv, které zajišťují samostatnou synchronizaci našich informací mezi lokálním a centrálním serverem. Microsoft Sync Framework obsahuje synchronizační služby pro ADO.NET, soubory, RSS i pro ADO.NET pro mobilní zařízení.

6.1.4 Standardní přístup k datům

Standardní přístup k datům při využití ADO.NET vypadá asi takto. Mezi aplikací a serverem máme DataAdapter. Který v rámci spojení posílá příkazy na server. Aplikace tedy posílá data přes DataAdapter, například příkaz SELECT a server jej pomoci SQL příkazu vykoná posléze aplikaci zašle data v DataSetu. Aplikace provede v DataSetu změny a úkolem DataAdapteru je tyto změny převést na SQL příkazy, který je pošle serveru. Velkou výhodou tohoto řešení je že klient provádí minimum práce na rozdíl od serveru provádějícího většinu operací. Velkým problémem toho řešení je, však závislost na kvalitě a stabilitě spojení a samotné sítě.



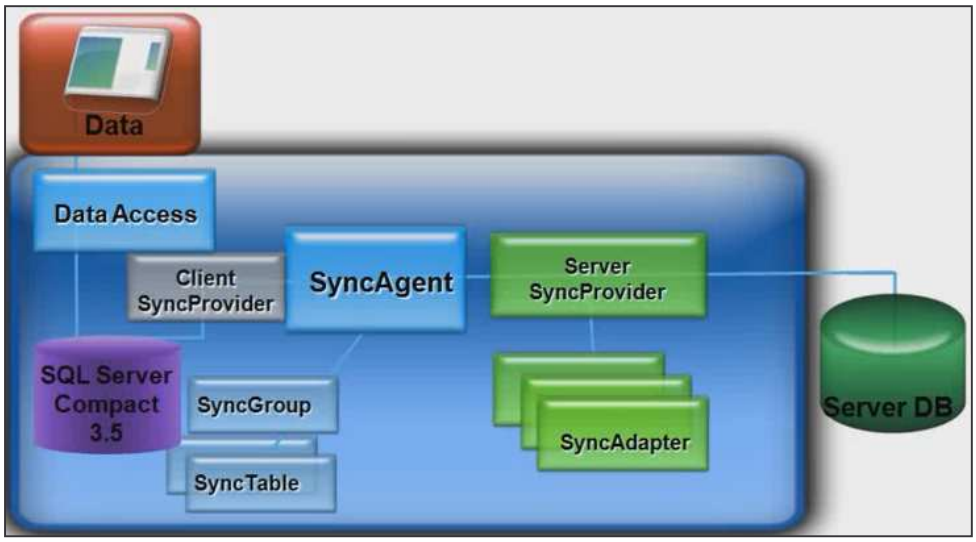
6.1.5 Služba SyncServices

Služba SyncServices pro ADO.NET má se standardním přístupem ADO.NET hodně společného, ale v několika důležitých věcech se liší. Samotné synchronizační služby se nacházejí za DataAdapterem. Najdeme zde synchronizační adapter, který kromě standardních SQL příkazů, přidává i tzv. inkrementální selenové příkazy, dále pak příkazy pro řešení kolizí, ke kterým může dojít při aktualizaci či mazání. A samozřejmě dokáže pracovat s DataSety stejně jako ADO.NET. Svým způsobem Sync Servisis pouze rozšiřuje ADO.NET o synchronizační služby.

Při tvorbě aplikace sloužící k práci s daty můžeme využít následující řešení. Abychom se nedostali do situace, kdy při komunikaci aplikace s centrální databází dojde k přerušení spojení, umístíme mezi aplikaci lokální databázi SSCE 3.5, a mezi lokální a centrální databázi SyncAdapter. Aplikace tedy pracuje beze změny s daty z lokální databáze a je nezávislá na síťovém připojení. Synchronizační služby na pozadí se pak starají o výměnu dat mezi oběma servery.

6.1.6 Architektura synchronizační služby

Na jedné straně, jak již víme, máme svojí aplikaci s daty, a na druhé straně server. Aplikace přistupuje pomocí Data Accessu ke svému lokálními SQL serveru. Jak na straně hlavního serveru, tak na straně klienta najdeme tzv. synchronizační providery. Mezi nimi se nachází synchronizační agent vykonávající práci a zajišťující synchronizační logiku. Na straně serveru běží synchronizační adaptery dávající přístup k jednotlivým tabulkám, jež synchronizujeme. Tabulky se nacházejí ve skupinách podle samostatných transakcí.



Existuje již větší množství synchronizačních služeb, ale pro naše účely zcela postačuje SyncServises verzepro ADO. NET. Tato služba je přímo navržena pro komunikaci přes ADO. NET s databázovými servery. Podporuje přímou synchronizaci založenou na službách. Je možné synchronizovat více tabulek či provádět obousměrnou synchronizaci. Ta je založena na poskytovateli, jímž může být libovolný ADO.NET poskytovatel. Součástí těchto synchronizačních služeb je SqlCeProvider.

6.1.6.1 SyncServices.- požadavky na server

Server by měl zvládnout zpracovat změny a nově vložené řádky pomocí tzv. timestampu (časových známek) a smazané řádky pomocí tabulky tombstone. Pokud to SQL server neokáže, měl by alespoň podporovat takzvané triggers (spouště). Díky nimž by se informace o změnách daly získat.

Nejnovější SQL Server 2008 podporuje tzv. change cracking umožňující vrátit klientovi jen jeho data, která byla změněna. Dále obsahuje funkci changeTable() umožňující vrátit tabulky. Tuto funkci můžeme používat ve FROM klauzuli, kde není problém zjistit informace o změnách.

6.1.6.2 Nastavení synchronizace a Visual Studio

Co se týče nastavení samotné synchronizace, je možné ve Visual Studiu 2008 použít Sync Designer. Ten nám pomáhá při konfiguraci Sync services pro ADO.NET nebo při konfiguraci SQL serveru pro konkrétní poskytovatele. Dokáže generovat příkazy pro aktualizaci dat a úpravy tabulek. Příkazy však nespouští, musíme je spustit sami nebo je používat na straně serveru. Designer obsahuje i podporu synchronizace tabulek.

6.1.7 Různé modely synchronizace

Samotných modelů synchronizace je několik. První možností může být okamžitá synchronizace dat, kdy odesíláme lokální změny nebo oboustrannou synchronizaci dat. Jinak synchronizaci provádíme také po dávkách, což nám umožňuje detekovat, zpracovávat a řešit kolize.

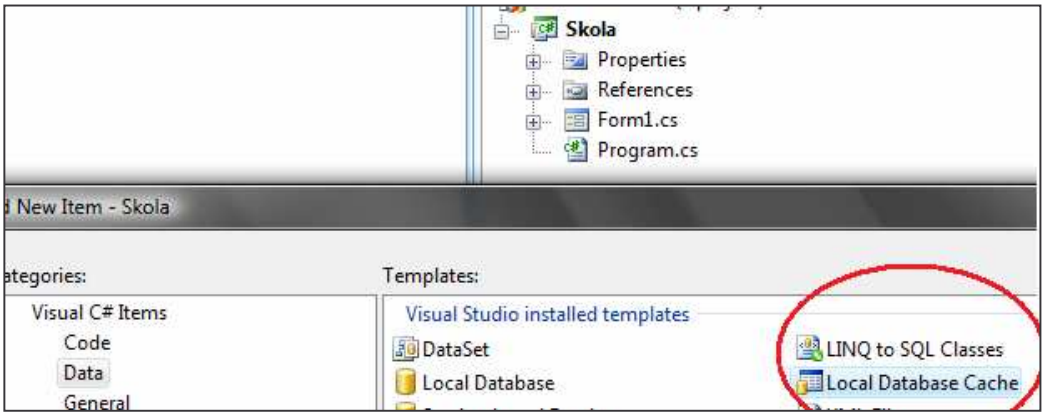
Další možností je využití logické fronty provedených změn, což se hodí, pokud je třeba použít tzv. „workflow“, nebo jestliže chceme data aktualizovat ihned po změně.

6.1.8 Příklad slučovací replikace ve Visual Studiu 2008

Nyní si ukážeme na příkladu synchronizaci slučovací replikace. Jako příklad slouží jednouchá aplikace vytvořená v prostředí Visual Studia 2008. Tato aplikace bude obsahovat lokální databázi a synchronizační nástroje, díky kterým bude lokální databáze aktualizovaná se serverem. Aplikace bude obsahovat data o studentech a oborech školy. Spustíme si Visual Studio a vytvoříme nový projekt WinForm.

6.1.8.1 Vytvoření lokální databáze

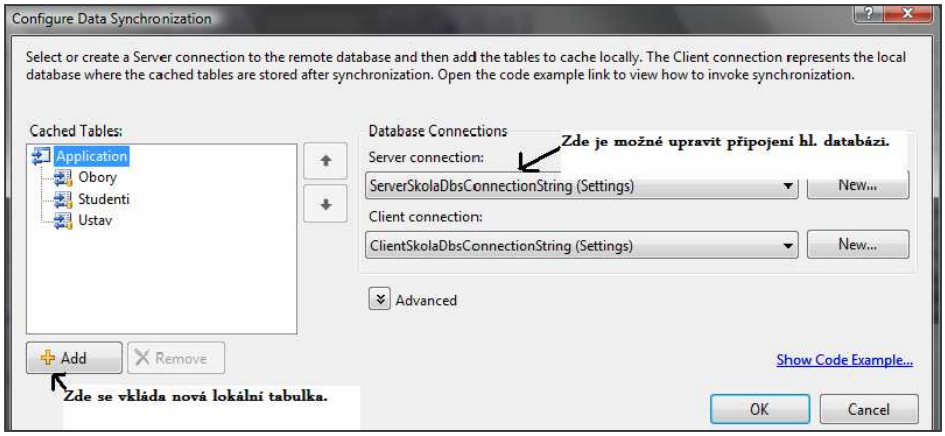
Další věcí, kterou uděláme, bude vložení lokální databáze (Cache) do našeho projektu. Z oblasti data vybereme Loacal Database Cache.



6.1.8.2 Konfigurace synchronizace

Nyní se nám otevře okno, ve kterém můžeme nastavit konfiguraci samotné synchronizace. Předtím však musíme provést nastavení připojení k hlavnímu serveru, s nímž bude synchronizace probíhat. Visual Studio nabízí možnost připojení jak k SQL Serveru SQL diabasovému souboru. Jakmile nastavíme správný connection string, vybereme si tabulku, kterou budeme používat v lokální databázi. Jakmile provedeme výběr tabulky, dostaneme možnost konfigurace synchronizace. Pokud v nabídce

synchronizaci tabulky zvolím, po potvrzení se provede první synchronizace mezi velkým serverem a lokální cache. Následně budou vygenerovány potřebné soubory jako Cache, Dataset. V našem příkladu jsem vybral tabulku Studenti a Obory.



6.1.8.3 Vytvoření formuláře

Poté vytvoříme formulář, v němž můžeme data z tabulek zobrazovat. Ve Visual studiu lze operaci provést velice rychle. Po otevření Data Source zvolím tabulku, jejíž položky bych chtěl editovat, popřípadě jinak upravovat v okně aplikací. Tuto tabulku přetáhnu do designéra grafického prostředí. Všechny objekty formuláře pro práci s databází se vygenerují.

6.1.8.4 Provedení replikace

Pokud chceme provádět synchronizaci s hlavním serverem, můžeme tuto operaci provést následujícím způsobem. Vytvořím tlačítko, jehož stisknutí zavolá posluchače s kódem této synchronizace. Musíme vytvořit synchronizačního agenta, kterého použijeme pro synchronizaci.

Dále zavolám replikaci na tabulky s daty. V našem případě na tabulky Studenti a Obory. Funkčnost si můžeme vyzkoušet například tak, že na straně hlavního serveru přidáme do tabulky nového studenta. Pokud máme aplikaci stále spuštěnou, a provedeme synchronizaci, data se objeví i u ní. Metoda Merge() pak sloučí tabulky Datasetů. Tato miniaplikace není závislá na kontinuálním spojení.

```
private void toolStripButton1_Click(object sender, EventArgs e)
{
    SkolaCacheSyncAgent syncAgent = new SkolaCacheSyncAgent();
    Microsoft.Synchronization.Data.SyncStatistics syncStats = syncAgent.Synchronize();

    this.skolaDbsDataSet.Obory.Merge(this.oboryTableAdapter.GetData());

    this.skolaDbsDataSet.Studenti.Merge(this.studentiTableAdapter.GetData());
}
```

7 Microsoft Sync Framework

7.1 Neocenitelný pomocník

7.1.1 Synchronizujeme téměř vše

Microsoft Sync Framework nabízí mnoho dalších možností synchronizace. Často se můžeme setkat s potřebou synchronizovat soubory, či adresáře na různých zařízeních.

7.1.2 Synchronizace souborů

Při synchronizaci souborů ukládáme informace o verzích těchto souborů, které porovnáváme a provádíme na nich různé operace podle naší potřeby. Naprogramovat vytváření a správu těchto pomocných informací není nic zábavného. Microsoft Sync Framework obsahuje několik objektů, které nám práci mohou ulehčit.

7.1.3 Teorie a důležité pojmy

Nyní si vymezme několik základních pojmů, mohou nám někdy pomoci. Základním kamenem je účastník. Identifikuje jej provider a jeho

obraz (replica), což není nic jiného než úložiště s informacemi, respektive jeho identifikace, jež se budou synchronizovat. Může se jednat o webovou službu, soubor či adresář na přenosném zařízení.

Podle schopností, jejich zdrojů a zařízení, rozdělujeme účastníky do několika kategorií:

1. Mezi tzv. úplné účastníky může počítat zařízení, která umí u sebe ukládat a mohou se na nich spouštět různé aplikace. Klasickým představitel je desktop, notebook nebo PDA.

2. Mezi další, někdy nazývané částečné účastníky, řádíme ta zařízení, která umí u sebe ukládat data společně s metadaty využitelných při synchronizaci. Představitelem pak může být datová karta, MP3 přehrávač atd.

3. Posledním účastníkem je tzv. jednoduchý účastník. Data může pouze poskytovat a nejde na něm data uchovávat. Může to být kupříkladu webová služba nebo RSS kanál.

Pro správnou funkci synchronizace musí být alespoň jeden s účastníků účastníkem úplným.

7.1.4 Metadata

Metada ukládáme například do souboru pro metadata a nebo do sloupce v databázi. Sync Framework se o ukládaní těchto metadat stará sám. Standartě je uchovává v SQL Servru Compact Edition.

7.1.4.1 Náhrobky, verze a znalosti

V metadatech můžeme najít informaci o verzi synchronizovaných dat. Ukládá se creation version a update vision. Každá z výše zmíněných verzí je složena s identifikátoru obrazu a logických hodin. Kromě verze pak metadata obsahují takzvané znalosti a náhrobky. Znalosti jsou poměrně úsporné informace, které jsou vyměňovány mezi účastníky,

pokud dojde k samotné synchronizaci. Snižují množství informací, které mezi sebou účastníci vyměňují a tak zvyšují efektivitu synchronizace. Poslední složkou metadat jsou tzv. náhrobky, které slouží jako seznamy dat, které byly odstraněny. Tyto informace neustále narůstají, a tak není od věci je čas od času promazat.

Jak synchronizace vypadá už víme. Poté, co zdroj požádá cíl o informace, zjistí, jaké informace se budou replikovat, a pomocí nich informuje cíl. Cíl zjistí konflikty, ke kterým může docházet. Pokud se konflikty vyřeší nebo odloží, dojde k posílání dat a jejich uložení.

7.1.4.2 Konflikty

Způsobů řešení konfliktů je celá řádka. Některá řešení je nutné provést ručně, jiná jsou téměř automatická. Může jimi být takzvané zalogování nebo sloučení. V některých případech je možné využít i sloučení.

7.1.5 Praktický příklad synchronizace adresářů

Nyní si vyzkoušíme synchronizaci adresářů na praktickém příkladě, neboť tato operace se nám může často hodit. Využijeme k tomu samozřejmě Microsoft Sync Framework. Samotné řešení je jednoduché a poměrně efektivní.

7.1.5.1 Instalace Sync Framework

Samotný Sync Framework není standardně součástí Visual Studia 2008. Proto nám nezbývá nic jiného, než ho ze stránek Microsoft stáhnout a nainstalovat. Po spuštění Visual Studia a vytvoření nového projektu musíme do assembly přidat reference na důležité knihovny Microsoft.Synchronization. Nesmíme zapomenout ani na namespacy.

7.1.5.2 Zdrojový kód

Pro synchronizaci použijeme FileSyncProvider. Kód programků můžeme vidět v metodě provedSynchro() níže:

```
private void provedSynchro()
{
    const string zdrojovyAdresar = @"E:\straka";
    const string cilovyAdresar = @"E:\racek";
    SyncId replicaId1 = new SyncId(new Guid(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1));
    SyncId replicaId2 = new SyncId(new Guid(2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2));

    using (
            FileSyncProvider zdroj = new FileSyncProvider(replicaId1.GetGuidId(),
                                     zdrojovyAdresar, null, FileSyncOptions.None,
                                     zdrojovyAdresar, "meta1.sync", @"E:\temp", @"E:\konflikt"),
                           cil = new FileSyncProvider(replicaId2.GetGuidId(),
                                     cilovyAdresar, null, FileSyncOptions.None,
                                     cilovyAdresar, "meta2.sync", @"E:\temp", @"E:\konflikt"))

    {
        SyncOrchestrator agentSych = new SyncOrchestrator();
        agent.LocalProvider = zdroj;
        agent.RemoteProvider = cil;
        agent.Direction = Microsoft.Synchronization.SyncDirectionOrder.Upload;
        agent.Synchronize();
    }

}
```

Nastavíme si cestu k adresářům, s kterými budeme provádět synchronizaci. Pak si vytvoříme identifikaci replik pro každý adresář. ID musí být vždy stejné, ale můžeme je klidně generovat. Nic nám nebrání si ID ukládat do souboru v synchronizačním adresáři.

Při vytváření objektů providerů musíme do jejich konstruktoru jako parametry předat identifikace replik, jednu pro každý z adresářů společně s názvem adresáře. Dalším parametrem je tzv. FileSyncScopeFilter, který jak již název napovídá, slouží k filtrování souborů nebo adresářů, které replikujeme. FileSyncOptions opět nevyužijeme, ale díky němu lze ovlivnit několik nastavení synchronizace. Jako třeba mazání souborů do koše. Pak ještě vybíráme adresář pro ukládání metadat a jeho název. My jsme zvolili umístění ve stejných adresářích, v nichž pracujeme. Poslední dvěma parametry je nastavení adresáře pro dočasné soubory a případné konflikty.

Na řadu pak přichází SyncOrchestrator, jež bude celou synchronizaci organizovat. Pak nastavíme providery, tak jak si představujme. A to na toho, kterého chceme vybrat jako zdroj, druhého, jenž bude cílem. Vybereme směr toku dat. Poslední klauzule pak operaci synchronizace spustí.

Funkci programu si můžeme vyzkoušet. Pokud je vše správně nastaveno a na nic jsme nezapomněli, proběhne synchronizace. V námi synchronizovaných adresářích vznikly soubory s metady. Sami jsme si to tak nastavili. Všimneme si, že tyto soubory se nestaly součástí synchronizace.

7.1.5.3 Posluchači objektů využitých pro synchronizaci

Pro SyncOrchestratora je možné nastavit i posluchače události SessionProgress. Tato událost nás může informovat o stavu, v němž se synchronizace zrovna nachází.

Jiné události je však možné registrovat i u samotných providerů. Například události AppliedChange, ApplyingChange a SkippedChange. První dvě nám dovolují reagovat na změny, které se provádí během synchronizace. Obsluha by mohla být řešena například tímto způsobem.

```
switch (e.ChangeType)
{
    case ChangeType.Create:
    Console.WriteLine("Provádím *vytváření* souboru: {0}", e.NewFilePath);
    break;

    case ChangeType.Delete:
    Console.WriteLine("Provádím *mazaní* souboru: {0}", e.OldFilePath);
    break;

    case ChangeType.Overwrite:
    Console.WriteLine("Provádím *přepisování* souboru: {0}", e.NewFilePath);
    break;

    case ChangeType.Rename:
    Console.WriteLine("Provádím *přejmenovíní* souboru: {0} (OldName: {1})",
                      e.NewFilePath, e.OldFilePath);
    break;
}
```

Při zpracování události SkippedChange lze zvolit podobný postup, jaký je vidět na tabulce výše. Tato událost informuje o důvodu přeskočení nějakého souboru. Tím důvodem může být například nemožnost přečíst soubor. Událost CopyingFile nás informuje o stavu průběhu kopírování.

Konflikty, ke kterým může při synchronizaci dojít, řeší poměrně efektivně FileSyncProvider. Snaží se, aby nedošlo při replikaci dat a synchronizaci mezi účastníky ke ztrátě dat. Ve standardních situacích lze pomocí Configuration.ConflictResolutionPolicy jednoduše nastavit správné chování, a nadefinovat, jaká data z jakého umístění mají větší váhu.

7.2 LINQ

7.2.1 Zásadní novinka v jazyku C#

Již dlouho používáme objektově orientované přístupy při tvorbě aplikací. Technologie, jako jsou zaobalení, dědičnost a polymorfismus, jsou dnes poměrně běžné.

Vedle objektově orientovaného přístupu existuje celá řada neobjektových datových zdrojů a informací, jako jsou relační databáze či XML. Bylo by výhodné vytvořit dotazovací jazyk, který by nebyl specifický k programovacímu jazyku, k datovému zdroji, jenž by bylo možné při integraci do jazyka kontrolovat po syntaktické stránce. A samozřejmě ho integrovat natolik dobře se současnými moderními vývojovými technologiemi, aby fungovaly takové pomůcky, jako je například třeba IntelliSence.

Projekt jazyka LINQ (Language Integreted Query) tyto požadavky plní. Definuje obecně použitelnou množinu operátorů nezávislou na datových zdrojích, nezávislou na typu programovacího jazyka, která nám umožňuje procházení, filtrování a projekci dat.

Důležité také je, že datové typy, nad kterými lze provádět takové to dotazování, musí implementovat generický typ i generické rozhraní IEnumerebe<T>.

V neposlední řadě je důležité říci, že projekt jazyka LINQ, a jazyk, který takhle vznikl, není uzavřený, množinu operátorů je možné rozšířit, a ty existující nahradit pomocí svého vlastního kódu.

7.2.2 Zdroje jazyka LINQ

Jazyk LINQ je dostupný pro všechny programovací jazyky, které přicházejí s Visual Studiem 2008, ale protože je jazykově agnostický

a dostupný na úrovni platformy .NET Framework, je možné dotazy pomocí tohoto jazyka vytvářet i v jazycích ostatních.

Samotný projekt jazyka LINQ byl rozdělen do tří samotných částí. První, která se dotazuje přímo nad objekty v paměť naší aplikace, dále na LINQ to ADO.NET, jež dokáže vytvářet dotazy nad relačními datovými zdroji, a část LINQ to XML vytvářející dotazy nad hierarchicky strukturovaným XML dokumentem. Samotný LINQ pro ADO.NET byl rozdělen do třech samotných částí, které odrážejí způsoby modulování relačních databází v paměti.

7.2.3 Příklad využití LINQ

Pro přiblížení dotazů LINQ můžeme využít následujícího kódu. Vytvoříme třídu ObjectQuery, která má metodu Test(). Ve třídě ObjectQuery najdeme další vnořenou třídu s názvem Osoba, která uchovává informace o člověku, tedy jeho jméno a věk. Následně v metodě Test () vytvoříme generický seznam osob a provedeme inicializaci.

V tuto chvíli můžeme s takovýmto seznamem vytvořit jednoduchý dotaz v jazyce LINQ. Využijeme nové klíčové slovo var pro definici návratové hodnoty z dotazu a vyfiltrujeme si například všechny uživatele, které můžeme označit jako mladistvé. Vybereme tedy z našeho seznamu osob všechny, pro které platí, že jsou starší než 15 a mladší než 19 let. Jakmile vytvoříme seznam mladistvých, můžeme pro každého vytisknout jeho jméno a příjmení.

```
class Osoba
{
    public int Age;
    public string Name;
}
```

```
Vysledek:
> Jmeno = Petr, Vek = 16
> Jmeno = Karel, Vek = 18
```

```
public static void Test()
{
    var osoby = new List<Osoba>() {
        new Osoba { Age=16, Name="Petr" },
        new Osoba { Age=22, Name="Richard" },
        new Osoba { Age=18, Name="Karel" }
    };

    //dotaz
    var mladistvi =
        from p in osoby
        where p.Age > 15 && p.Age < 19
        select p;

    Console.WriteLine("Vysledek:");
    foreach (var val in mladistvi)
    {
        Console.WriteLine("> Jmeno = {0}, Vek = {1}", val.Name,
                          val.Age);
    }
    Console.ReadLine();
```

7.2.4 Další novinky jež přinesl .NET Framework

7.2.4.1 Anonymní typ

Pokud se podíváme na jednoduchý dotaz v jazyce C# a projektu LINQ, kde z množiny mladistvých, pro něž platí, že jsou ve věku 15 až 20 let, vybíráme ty, kteří toto kriterium splňují, používáme klíčové slovo var. Toto slovo představuje tzv. anonymní proměnnou. Typ proměnné je zjištěn na základě výrazu nacházejícího se na pravé straně. Důležité je také říci, že v tomto případě deklarace je vždy zachována typová bezpečnost, protože typ je vytušen. Tento způsob tedy není identický s použitím typu objekt, který je společný pro všechny typy na .NET Framework. Inferenci typu lze použít pouze na lokální proměnné.

7.2.4.2 Lambda výrazy

Lambda výraz umožňuje jednoduchým způsobem psát funkci, kterou lze předat jako parametr nebo argument k dalšímu vyhodnocení. Lze pomocí něj pokládat dotazy podobně, jako jazykem LINQ. Vše můžeme vidět na příkladu níže. Vytvořili jsme si seznam ovoce, do kterého jsme vložili několik jeho představitelů. Dále jsme využili anonymní typ, do něhož pomocí lambda výrazu vybereme takové ovoce, jehož barva je modrá. Lambda výraz je tzv. tokenem (=>) odděleným od typového seznamu lambda výrazů.

```
public class Ovoce
{
    public string Jmeno{get;set;}
    public string Barva{get;set;}

    public Ovoce(string jmeno, string barva)
    {
        this.Jmeno=jmeno;
        this.Barva=barva;

    }
}

[STAThread]
static void Main()
{


    List<Ovoce> seznamOvoce = new List<Ovoce>();

    seznamOvoce.Add(new Ovoce("Švestka","Modrá"));
    seznamOvoce.Add(new Ovoce("Jablko","Červená"));
    seznamOvoce.Add(new Ovoce("Pomeranč","Oranžová"));

    var vybraneOvoce= seznamOvoce.FindAll(takoveOvoce=> (takoveOvoce.Barva=="Modrá"));

    foreach (var konkretniOvoce in vybraneOvoce)
    {
        Console.WriteLine(konkretniOvoce.Jmeno);
    }
```

7.2.4.3 Vlastnosti

Vlastnosti je možné využívat i ve starších verzí C#. V nové verzi je však možné využít zkrácené formy. Stačí napsat název vlastnosti a do závorek klíčová slova get a set, pokud nepotřebujeme žádnou dodatečnou logiku a nemusíme ani deklarovat členské proměnné, ke kterým takto přistupujeme, protože tyto proměnné jsou podle typu proměnné automaticky vytvořeny.

8 Tvorba aplikace

8.1 Zadání a podklady

8.1.1 Zadání

Nyní se pokusíme vytvořit aplikaci, která bude využívat výše zmíněné nástroje a neobejde se bez nutnosti replikace dat. Jak budeme postupovat?

Společnost Attavena zabývající se pořádáním kurzů pro výuku základů práce na PC, využívá po celém jihočeském kraji síť počítačů. Na těchto počítačích mohou účastníci cvičit, surfovat na internetu a plnit úkoly související s kurzem. Využívají toho všichni Ti, kteří nemají možnost pracovat na počítači ve své domácnosti, protože počítač nevlastní.

Společnost je částečně financována ze státních a evropských fondů a musí evidovat jednotlivé účastníky. Správce nemůže účastníkům vytvořit na počítačích uživatelské účty, protože není předem jasné, kdo možnosti využije. Prozatím jsou záznamy o osobách evidovány v tištěných materiálech. Někteří účastníci tyto informace nevyplňují. Zaměstnanci firmy vyplněné i nevyplněné formuláře sbírají a vyhodnocují.

8.1.2 Požadavky na aplikaci

Vytvoříme tedy aplikaci, která se po spuštění operačního systému otevře do nového okna a donutí uživatele, aby požadované informace vyplnili do formuláře v programu. Po té co uživatel informace vyplní, aplikace bude na pozadí stále zaznamenávat čas, po který je uživatel přihlášen, a zaeviduje tato data do lokální databáze, a zálohu do XML souboru.

Po určité době správce dotyčného počítače provede replikaci dat z lokální databáze na centrální SQL server. Popřípadě může některé záznamy upravit, či smazat.

8.1.3 Návrh aplikace

Využijeme prostředí Visual Studia 2008 a technologií ,NET (3.5) a Sync Framework. Je nutné mít je nainstalované na svém počítači. Při tvorbě využijeme SQL Server 2005 v Express edici, bude představovat centrální server. Aplikaci píšeme v programovacím jazyku C#.

Samotná aplikace bude pracovat ve dvou módech. První mód, realizovaný formulářem v okně Windows, bude sloužit ke sběru dat a poběží v pozadí, dokud uživatel nevypne počítač. Aplikaci vypne až ukončující se operační systém. Data o přihlášení se uloží do databáze a jejich záloha do XML souboru. Druhý mód aplikace pak využije správce ke správě záznamů v XML, popřípadě ke správě samotné lokální databáze. V této fázi je možné provést synchronizaci s centrálním serverem.

8.1.3.1 Třída Přihlášení

Protože programujeme v objektovém jazyce, vytvoříme si třídu Přihlášení, která bude obsahovat jako atributy všechny informace, jež chceme uchovávat, a metody, popřípadě vlastnosti, díky kterým tyto atributy můžeme pozměňovat. Instance této třídy využijeme i v jiných částech programu.

8.1.3.2 Třída Přihlášení s formulářem

Další třída bude představovat samotný formulář v okně Windows. Musíme tedy v ní vytvořit jednotlivé položky formuláře a dále mechanismy, které po vyplnění formuláře umožní data uložit do databáze, XML, a nadále evidovat čas přihlášení uživatele.

8.1.3.3 Třída Spravující záznamy

Umožňuje zpracování záznamu. Bude obsahovat mechanismy, díky kterým budeme moci záznamy v XML souboru popřípadě databázi měnit. Jednotlivé prvky formuláře propojíme s datovými zdroji.

8.1.3.4 Třídy pro prácí s datovými zdroji

Tyto třídy zrealizují správu datových zdrojů.

8.1.3.5 Třídy pro synchronizaci

Provádí a řeší poměrně složitou problematiku replikace dat a jejich synchronizaci. Musí zajistit komunikaci lokálního a centrálního serveru.

8.1.3.6 Psaní zdrojového kódu

Všechny třídy nebudeme psát ručně. Především ne v první fázi tvorby aplikace. Ukážeme si výhody nástrojů Visual Studia 2008, které nám některé třídy ušijí doslova „na míru“. Jejich efektivita a přehlednost bude mnohdy vyšší, než třídy ručně vytvořené začínajícím programátorem. Závěrem celý projekt zhodnotíme a zaměříme se na tos, co by se dalo vylepšit.

8.2 Vytváříme aplikaci - první fáze vývoje

8.2.1 Kde zdrojové kódy najdu?

Na přiloženém CD v adresáři Přihlašování najdeme projekt ve dvou fázích rozpracování. Složka s první fází vývoje aplikace obsahuje projekt naprogramován až k místu, kde vytváříme lokální databáze. Pokud tedy v této části bakalářské práce něčemu nerozumíte (např. chcete si třídy prohlédnout), máte možnost si projekt ve Visual Studiu 2008 z CD načíst. U většiny tříd a metod najdete komentáře. Nyní budeme pracovat s první fází projektu.

8.2.2 Třída Prihlasovani

8.2.2.1 Jak třída vypadá

Spustíme si Visual Studio a zvolíme možnost **File… ⇒ New… ⇒ Project…** Dále vybereme možnost **Visual C# ⇒ Windows ⇒ Windows Forms Application**. Aplikaci pojmenujeme jako Prihlasovani. Visual Studio vygeneruje třídu Form, které zatím nebudeme věnovat pozornost.

Pustíme se do programování třídy Prihlaseni. Instance této třídy představují souhrn informací o přihlášení. V přihlášení chceme evidovat jméno, příjmení a stavy uživatele. (Jestli je zaměstnán a v jaké věkové kategorii se nachází.) Atributy této třídy vycházejí z požadovaných informací. Co informace to atribut.

Do projektu vložíme **New Item** z menu **Project** typu **Class**. Vytvoříme si privátní atributy. Dále bychom měli k těmto atributům vytvořit metody, jež nám umožní nastavit jejich hodnoty. My však vytvoříme jejich vlastnosti, což nám .Net Framework umožňuje. Stavy uživatele budeme uchovávat v kolekci. Pro hodnoty stavů můžeme vytvořit vlastní výčtový typ, který využijeme v metodách sloužících k inicializaci stavů. Dále si vytvoříme konstruktor, nainicializujeme v něm základní hodnoty atributů. Více o kódu na CD.

8.2.3 Třída PrihlasForm

8.2.3.1 Jak třída vypadá

Tato třída bude obsahovat formulář, jenž se bude nacházet v okně Windows. Budeme tedy upravovat třídu Form, kterou jsme si vytvořili výše. Přejmenujeme ji na PrihlasForm a pustíme se do práce. Na CD v projektu si můžeme výsledný formulář prohlédnout a vytvořit jej podobně. Zobrazíme si tzv. ToolBox a přetáhneme jednotlivé komponenty na formulář.

Tato třída ještě obsahuje celou řadu metod, jejichž funkci je možné vydedukovat a nebo vyčíst z komentářů, které najdeme v jejich zdrojovém kódu.

Tato třída má atribut aktualniPrihlaseni typu Prihlaseni a používá jej k ukládání informací z formuláře. K uložení dojde po stisknutí tlačítka OK uživatelem. Metody zavolané posluchačem tlačítka provedou uložení údajů do atributu aktualniPrihlaseni, uložení do XML souboru, a spustí vlákno měřící čas. Toto vlákno po určitém časovém intervalu vyvolá

pomocí svého posluchače metodu, která data aktualizuje. (dobu, po kterou je uživatel stále přihlášen) Aplikace běží dále na pozadí, ale její okno se skryje. Všechny tyto operace jednoduše naprogramujete, pokud si prohlédnete zdrojový kód na CD.

8.2.3.2 Ukládaní do souboru XML

Protože ukládaní do souboru XML souvisí úžeji s problematikou práce s daty, prohlédneme si metodu ulozeniDoSouboru() pečlivěji. Také se přesvědčíme, že pokud budeme psát kód ručně, bude se jednat o práci nepříliš zábavnou a ani nikterak povznášející. Později si ukážeme, jak nám Visual Studio práci dokáže výrazně zjednodušit.

V první části metody dojde k vytvoření XmlDocumentu a jeho uložení. Do tohoto dokumentu musíme vložit jednotlivé elementy nesoucí informace o přihlášení. To provedeme tak, že vytvoříme objekty třídy XmlElemnt a XmlNode představující elementy a uzly v dokumentu. Do nich pak pomocí vlastností atributu aktualniPrihlaseni uložíme hodnoty. Elementy začleníme do dokumentu a dokument opět uložíme, tentokrát pod jedinečným jménem. To je složeno ze jména PC klienta, které nastavuje správce, a času, kdy k přihlášení došlo.

```
XmlDocument prihlaseni = new XmlDocument();

prihlaseni.LoadXml("<?xml version='1.0'?><prihlaseni xmlns='prihlaseni'></prihlaseni>");
prihlaseni.Save(jmenoSouboruPr);

prihlaseni.Load(jmenoSouboruPr);


XmlElement puvodniJmenoSouboru = prihlaseni.CreateElement("puvodniJmenoSouboru", "prihlaseni");
puvodniJmenoSouboru.InnerText = this.jmenoSouboruPr;

XmlNode jmenoPc = prihlaseni.CreateElement("jmenoPc", "prihlaseni");
jmenoPc.InnerText = aktualniPrihlaseni.JmenoPc;

XmlNode casPrihlaseni = prihlaseni.CreateElement("casPrihlaseni", "prihlaseni");
casPrihlaseni.InnerText = aktualniPrihlaseni.CasPrihlaseni;
```

Kód metody jsme museli napsat sami. Vidíte, že podpora práce s XML v C# je na dobré úrovni a nejedná se o nic obtížného. Na druhou stranu při práci s daty nám může být Visual Studio daleko nápomocnější.

8.2.4 Třída SpravcePrihlasForm

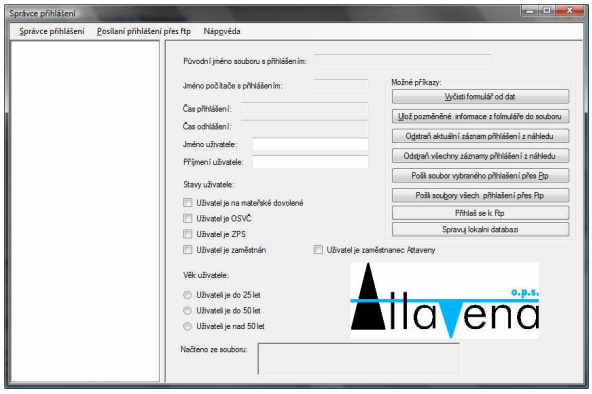
8.2.4.1 Funkce třídy

Tato třída slouží ke správě záznamů, ač pracuje poněkud jinak, než třída předchozí. Svou konstrukcí se jí v mnohém podobá. Opět se jedná o třídu, která obsahuje velké množství atributů představujících komponenty formuláře.

Pokud je tedy instance této třídy spuštěna, můžeme načítat XML soubory, prohlížet si je a upravovat jednotlivé záznamy. Poté provedené změny uložit. Soubory můžeme také posílat pomocí FTP protokolu. Všechny tyto funkce jsou realizovány pomocí několika metod.

8.2.4.2 Metody pro XML

Protože XML souborů může být načteno i více, obsahuje tato třída atribut typu kolekce, který v sobě uchovává data z XML souborů. Ta jsou načítána opačným způsobem, jakým byla ukládána ve třídě PrihlasForm. Jak asi tušíte, využívá se objektů typu Prihlaseni. Načtení realizuje metoda nactiPrihlaseni() a zpětné ukládaní pak metoda ulozDoSouboru(). Další metody pak realizují načtení dat z formuláře, a provedení operací, pokud uživatel aplikace stiskne nějaké tlačítko.



8.2.5 Ukončení první fáze tvorby aplikace

8.2.5.1 Zhodnocení

Po vytvoření této třídy ukončíme první fázi tvorby aplikace a provedeme malé zhodnocení:

1. Do této chvíle jsme většinu kódu psali ručně.

2. Pokud jsme vytvářeli formuláře, využili jsme možnosti designéra nabízeného nám Visual Studiem. Museli jsme však komponenty umísťovat a upravovat.

3. Vazby mezi datovými zdroji a formulářem jsme vytvářeli ručně.

V další části si ukážeme, že i většina těchto operací můžeme být pomocí Visual Studia značně zjednodušena.

8.2.5.2 Ostatní součásti projektu

V projektu dále najdeme třídy jako Napoveda, NastaveniForm a FtpForm. Všechny tyto jednoduché třídy obsahují třídy dědící ze třídy Form. Úkolem třídy NastavaníForm je ukládat a upravovat nastavení aplikace, kterým se ovlivňuje chod programu, pokud je spuštěn v módu přihlášení. Máme možnost upravit název klientského počítače, zvolit místo, kam se uloží XML soubory s informacemi o přihlášení a nastavit heslo do nastavení. (Do nastavení se dostaneme, pokud klepneme na tlačítko nastavení a do TextBoxu napíšeme heslo pro vstup, standardně 123)

Druhá třída má obdobnou funkci. Je umožněno nastavovat parametry pro posílání souborů pomocí FTP. Obě třídy dále používají pro ukládání informací o nastavení XML soubory, které musí být v adresáři s aplikací!

8.3 Vytváříme aplikaci - druhá fáze vývoje

8.3.1 Jak dál

V této fázi vytváření aplikace přidáme lokální databázi, dále třídy, které nám umožní zobrazovat a editovat informace v ní (stejně jako formuláře pro XML před tím) a třídy, jež zajistí správnou komunikaci formuláře a databáze.

A to nedůležitější. Třídy, díky kterým budeme moci provádět rychlou a efektivní synchronizaci dat mezi lokální databází a hlavním centrálním serverem, aby replikace dat proběhla bez problémů.

8.3.2 Rychlost, jednoduchost a efektivita!

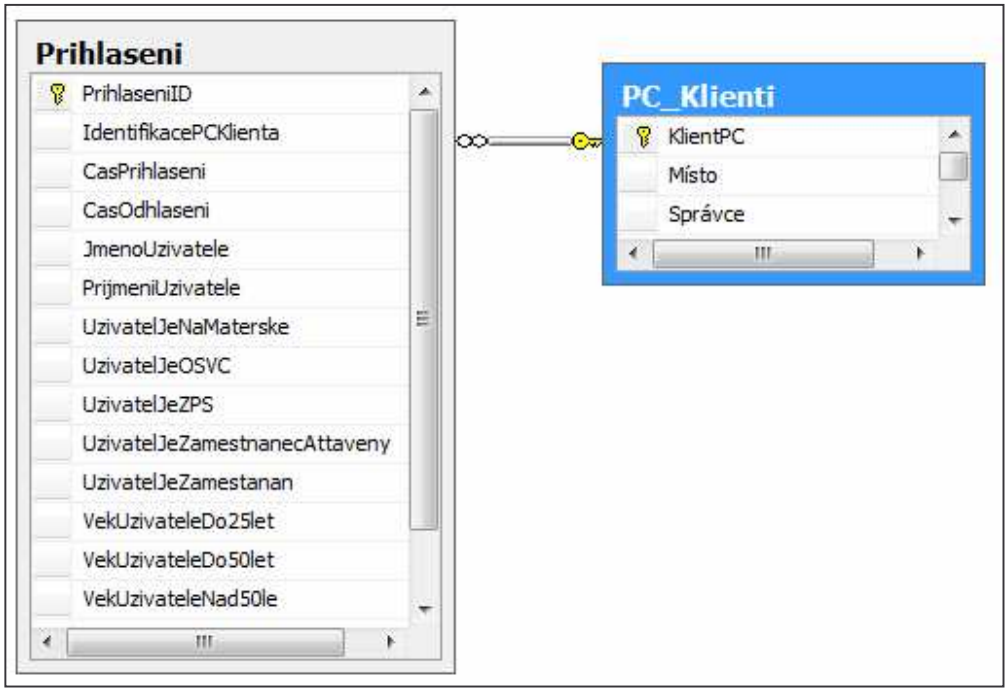
Budeme tedy pracovat s databázemi. Problematika práce s databázemi je poměrně složitá. Pokud budeme chtít propojit databázi a aplikaci, neobejdeme se bez potřebných znalostí tříd, jež najdeme v ADO.NET. Synchronizace je pak ještě daleko složitější. Pokud jsme však práci četli pozorně, budeme se nyní dobře orientovat.

Druhá fáze tvorby aplikace se nám může zdát až triviální. Napsání první části programu ručně podle předlohy nám mohlo zabrat až několik hodin. Druhá fáze ale zabere jen několik minut , přestože je mnohem složitější!

8.3.3 Databáze na serveru

Nyní si vytvoříme pro účely tvorby aplikace svojí centrální databázi na serveru. Jako server využíváme SQL Server 2005 a pro správu pak SQL Server Management Studio Express. Jednoduchou databázi postačující našim potřebám najdeme na přiloženém CD jako skript, což nám ulehčí práci.

Databáze vypadá asi takto:



Vidíme, že databáze není nikterak složitá, je doslova triviální, tedy pro naše účely naprosto ideální. Nyní máme na straně centrálního serveru vytvořenou databázi.

8.3.4 Vytvoření lokální databáze a další operace

Nyní si „držme klobouky“. V několika následujících krocích provedeme několik velice důležitých akcí.

1. Vytvoříme lokální databázi s SQL Serverem Compact Edition. Díky tomu bude naše aplikace nezávislá na centrálním SQL Serveru.

2. Nastavíme podmínky pro synchronizaci. To znamená vytvoření tabulek s náhrobky, vložení trigrů, které se po updatu či smazaní nějaké položky spustí.

3. Vybereme si, jaké tabulky bude obsahovat naše lokální databáze. Nemusíme se tedy snažit vytvářet stejnou strukturu tabulek jako na serveru. Vše se provede za nás.

4. Upravíme současný sběr dat.

5. A vytvoříme pomocí nástrojů Visual Studia dvě třídy pro editaci položek v lokální databáze s možností synchronizace.

6. Nastavíme typ synchronizace tabulek.

8.3.4.1 Vytvoření lokální databáze

V horní liště ve Visual Studiu vybereme menu **Project ⇒ Add New Items… ⇒ Data** a možnost **Local Database Cache**. Defaultní název LocalDataCache neměníme. Pak se objeví se okno **Configure Data Synchronization**.V oblasti **Database Connections** upravíme připojení k hlavnímu serveru. Klepneme na tlačítko **New…**, objeví se okno **Add Connectoin**. Jako **Data Source** zvolíme SQL Server pomoci tlačítka **Change…**. Server name nastavíme jménem našeho serveru. Pokud máme přístup k němu jištěn heslem, vyplníme jej a jako poslední operaci zvolíme databázi, kterou ze serveru budeme používat. Operaci potvrdíme tlačítkem **Ok**. Automaticky se vše nainicializuje k vytvoření lokální databáze, jak můžeme vidět v oblasti **Client Connection**.

8.3.4.2 Nastavíme podmínky pro synchronizaci

Ve stále otevřeném okně **Configure Data Synchronization** vybereme tlačítko **Add** a označíme tabulky, které chceme synchronizovat. V našem případě tedy obě, jak PC_Klienti, tak Prihlaseni. Na pravé straně okna je možné vybrat, jakým způsobem proběhne první synchronizace, a v jeho dolní časti trigery a skripty, jenž se mají v databázi vytvořit. My po vybrání tabulek vybereme tlačítko **Ok**, protože stávající nastavení nám vyhovuje. a opět vybereme tlačítko **Ok**, čímž se všechny námi výše vybrané operace provedou.

8.3.4.3 Upravení lokální databáze

Nyní se před námi otevřelo okno **Data Source Configuration Wizard**, ve kterém nastavíme podobu naší lokální databáze. Protože v databázi budeme používat obě tabulky z SQL serveru, vybereme je a pokračujeme dále pomocí tlačítka **Finish**.

Nyní jsme v několika málo krocích udělali pomocí Visaul Studia velký kus práce. V projektu aplikace již najdeme vazby mezi lokální databází a hlavním serverem. Pro práci s lokální databází se vygenerovala třída PrihlasovaniDataSet, díky níž lze vytvářet dotazy na lokální databázi, upravovat DataAdaptery k tabulkám a provádět mnoho jiných operací. Vše za pomoci grafického rozhraní. Pokud máme chuť, můžeme si tuto třídu prohlednout a uvidíme, kolik kódu obsahuje.

8.3.4.4 Upravení současného sběru dat

Pokud si vzpomínáme na třídu PrihlasForm, tak jistě víme, že stisknutí tlačítka Ok zavolalo posluchače tlačítka Ok a spustilo metody, které uložily data z formuláře do XML. Pak se spustilo vlákno, které v určitých intervalech volalo metody, jež záznamy v XML souboru aktualizovaly. Je tedy potřeba ještě vložit metody, které tyto operace provedou i do nově vytvořené lokální databáze. Jsou to metody ulozeniUdajuDoDatabaze() a updateLokalniDatabaze().

```
private void ulozeniUdajuDoDatabaze()
{
    PrihlasovaniDataSetTableAdapters.PrihlaseniTableAdapter prihlaseniTableAdapter = new Prihl
    PrihlasovaniDataSet mujDataSet = new PrihlasovaniDataSet();


   prihlaseniTableAdapter.Insert(vytvorJmenoS(false), aktualniPrihlaseni.JmenoPc, DateTime.Now
        aktualniPrihlaseni.JmenoUzivatele, aktualniPrihlaseni.PrijmeniUzivatele,
        aktualniPrihlaseni.vratStav(Stavy.materska), aktualniPrihlaseni.vratStav(Stavy.osvc),
        aktualniPrihlaseni.vratStav(Stavy.zamestnanecAttavena), aktualniPrihlaseni.vratStav(St
        aktualniPrihlaseni.vratStav(Stavy.vek_50), aktualniPrihl Prihlaseni PrihlasForm.aktualniPrihlaseni ce)

   prihlaseniTableAdapter.Update(mujDataSet);
}
```

Tyto metody používají k přístupu k lokální databázi DataAdaptery mající za úkol provádět operace nad zvolenými tabulkami. V prvním případě, tedy při volání metody ulozeniUdajuDoDatabaze(), dáváme lokální databázi příkaz Insert, jež jí tlumočí DataAdapter tabulky Prihlaseni. Tato třída pro práci s databází byla spolu s jinými vygenerována v okně **Data Source Configuration Wizard**, jakmile jsme vytvořili lokální databázi. Kdo chce vědět více, ať se podívá do třídy PrihlasovaniDataSet.

```
private void updateLokalniDatabaze()
{
    PrihlasovaniDataSetTableAdapters.PrihlaseniTableAdapter prihlaseniTableAdapter = new Prihlasov
    PrihlasovaniDataSet mujDataSet = new PrihlasovaniDataSet();
    prihlaseniTableAdapter.Fill(mujDataSet.Prihlaseni);


    DataRow aktualniRow = mujDataSet.Prihlaseni.Rows.Find((object)vytvorJmenoS(false));
    aktualniRow[3] = Convert.ToDateTime(aktualniPrihlaseni.CasOdhlaseni);



   prihlaseniTableAdapter.Update(mujDataSet.Prihlaseni);

}
```

8.3.4.5 Vytvoření tříd s formulářem pro práci s lokální databází

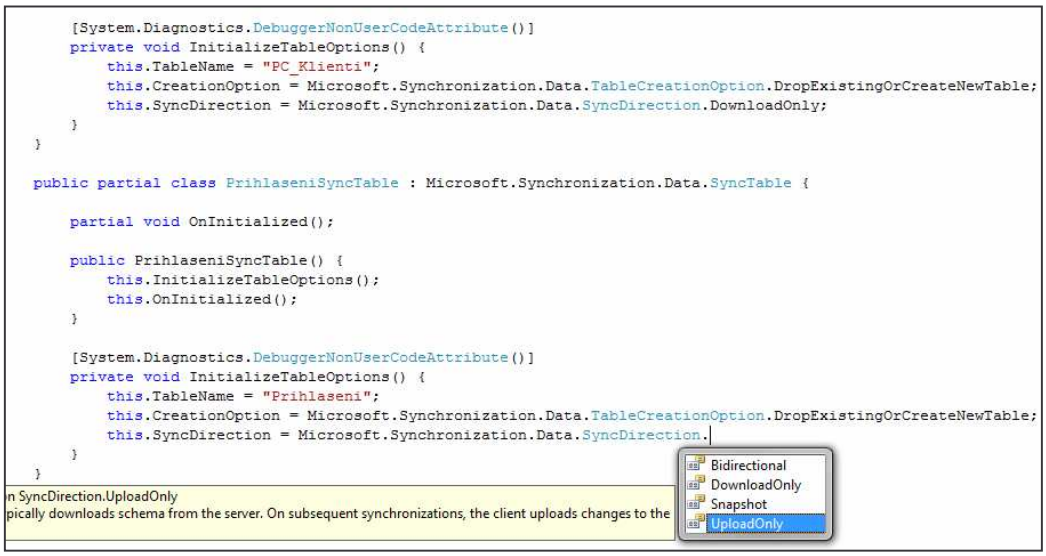
Nyní vytvoříme třídy pro editaci dat v lokální databázi. Do projektu přidáme dvě nové položky přes **Project ⇒ Add New Items… ⇒ Windows Form ⇒ WindowsForm**. A pomocí tlačítka Shift+Alt+D zobrazíme Data Sources. Pravým tlačítkem nad každou tabulkou v Data Sources můžeme zvolit styl zobrazení jejích sloupců. Pokud zvolíme DataGridView, budeme data na formuláři mít na objektu Windows Formu právě v objektu DataGridView. Pokud Details, vytvoří se formulář, jehož objekty budou záviset na typech sloupců v databázi. Pokud na každý z nově vložených WinFormů přetáhneme tabulky z Data Sources, přesvědčíme se o tom. Práce nám zabrala pár vteřin. Pokud bychom však stejně jako v první fázi tvorby programu každou položku přetahovali z ToolBoxu, zabrala by nám práce několik hodin. Zvlášť, když bychom ještě vytvářeli spojení mezi databází a formulářem, a psali metody pro ukládání a mazaní položek z databáze. Obě nově vytvořená okna můžeme začlenit do správcovské aplikace třeba tak, že ve třídě SpravcePrihlasovaniForm vytvoříme dvě tlačítka, a pomocí jejich posluchačů budeme volat instance těchto WinForm.

8.3.4.6 Nastavení typu replikace

A blížíme se k závěru. Po celou dobu jsem se nezmínil o souboru LocalDataCache. Tento soubor obsahuje důležité třídy, např. LocalDataCacheSyncAgent, a také metodu OnInitialized() sloužící k jeho inicializaci. V souboru LocalDataCache.Designer.cs najdeme další třídy pro synchronizaci.

Na příklad LocalDataCacheClientSyncProvider, tedy dodavatel pro klientskou databázi, i jeho protějšek pro serverovou databázi. Dále synchronizační tabulky ke každé z našich tabulek a metody, které synchronizaci provádějí obsahující SQL příkazy.

Nyní najdeme následující část kódu (viz. obrázek níže) týkající se našich tabulek a zvolíme typ replikace. U každé SyncTable upravujeme metodu InitializeTableOptions(), přesněji inicializaci její vlastnosti SyncDirection. Můžeme ji nastavit na jednu z následujících hodnot ovlivňujících způsob replikace. Bidirectional (oboustranná), Snapshot (snímková), DownloadOnly (pouze stažení) nebo UploadOnly (pouze nahrání) na centrální server.



Poslední operací, kterou provedeme, je zavolání metody Synchronize() na objekt třídy LocalDataCacheSyncAgent. Do formulářů, které jsme vytvořili výše (pro editaci a správu lokální databáze), vložíme tlačítka, a do kódu pro jejich posluchače vložíme kód, který volá metodu synchronizačního agenta Synchronize().

Jen malé upozornění! K synchronizaci dojde, ale nemusí dojít k aktualizaci Data Source u formulářových objektů. (Není vidět žádná změna.) To můžeme napravit zavoláním metody Fill() Data-Adaperu pro dotyčnou tabulku. Viz kód níže. LocalDataCache-SyncAgent obsahuje vlastnosti, které nás mohou informovat o průběhu synchronizace.

```
private void toolStripButtonSycnchronizace_Click(object sender, EventArgs e)
{
    LocalDataCache1SyncAgent syncAgent = new LocalDataCache1SyncAgent();
    Microsoft.Synchronization.Data.SyncStatistics syncStats = syncAgent.Synchronize();
    this.prihlaseniTableAdapter.Fill(this.prihlasovaniDataSet.Prihlaseni);

}
```

8.4 Tvorba aplikace závěr

8.4.1 Nedostatky aplikace

Aplikace je tedy zcela funkční. Co se týče samotné efektivity kódu, není v první fázi tvorby aplikace na vysoké úrovni. (Ukládání nastavení, hesla atd.) Soubory s nastavením bychom měli vložit do Resources aplikace a okna s nastavením by měla být děděná z rodičovských oken. Dále připojení k lokální a hlavně centrální databázi jsou v aplikaci tzv. „natvrdo“, což není nejvhodnější řešení. Nevyužíváme jazyka LINQ a nových východ např. při deklaraci vlastností. Spouštět přihlašovací mód bychom mohli jako WindowsServices. Neměli bychom ani zapomínat na ošetření výjimek, ke kterým může dojít při synchronizaci.

Oprava těchto nedostatků je již plně v rukou čtenáře. Berte ji jako další úkol pro procvičení zvýšení efektivity kódu programování v C#.

8.4.2 Spuštění aplikace

Ve třídě Program, kde najdeme vstupní bod celé aplikace, nalezneme i podmínku kontrolující parametr, s jakým byla aplikace spuštěna. Díky tomu má aplikace tzv. dva módy. Správcovský a přihlašovací. Správcovský se spouští bez parametru a uživatelský mód s parametrem „p“.

Správce po spuštění aplikace ve správcovském módu může odeslat informace o přihlášeních na centrální server, a i jiným způsobem spravovat aplikaci.

Abychom zajistili při startu spuštění této aplikace, nastavíme ho v plánovači úloh systému Windows. Aplikace se tedy spustí a poběží tak dlouho, dokud se počítač nevypne nebo uživatel neodhlásí. Za běhu systému je možné aplikaci v přihlašovacím módu zastavit pouze ukončením procesu Prihlasovani.exe ve Správci úloh Windows.

Pro správnou funkci programu je nutností mít v adresáři se spouštěcím souborem Prihlasovani.exe i xml soubory obsahující informace o nastavení, a to soubory serverFTP.xml a nastaveniPrihlaseni.xml.

8.4.3 Srovnání

Srovnáme- li obě fáze vývoje programu, dojdeme k jasnému závěru. Ve druhé fázi za pomoci nástrojů Visual Studia docílíme stejných, jistě i lepších výsledků, a to v kvalitě zdrojového kódu i jeho efektivitě.

Přitom tvorba pomocí nástrojů Visual Studia nebyla nikterak složitá ani pracná. Psát kód replikace by bylo velice zdlouhavé a nepříliš zábavné.

Proto pokud je to možné, využití nástrojů doporučuji. Vždyť v případě potřeby lze vygenerovaný kód ručně upravit a přizpůsobit našim potřebám.

9 Závěr

9.1.1 Několik slov závěrem

V této práci jsme se seznámili s problematikou replikace dat a práce s daty vůbec. To za využití Sync a. NET Frameworku a programovacího prostředí Visual Studio 2008. Celá práce byla proložena množstvím příkladů, jejichž zdrojové kódy můžete najít na přiloženém CD. V poslední části jsme si přiblížili tvorbu aplikace rozdělenou do dvou fází. V první fázi vývoje aplikace jsme psali kód ručně a ve druhé jsme použili nástrojů Visual Studia 2008.

Při srovnání obou metod dojdeme k závěru, že nástroje, které Visual Studio 2008 nabízí, nám tvorbu aplikace výrazně zjednoduší. Tyto nástroje generují kód na základě našich požadavků a umožňují nám tak nechat nezáživnou práci na nich jak při tvorbě grafického uživatelského rozhrání, tak při problémech složitějších, jako je například synchronizace databází.

Samozřejmě není možné tyto nástroje využívat, aniž bychom jim alespoň trochu porozuměli. Protože vždy budeme muset generovaný kód upravit a přizpůsobit našim požadavkům. Proto jsme důležité teoretické poznatky podrobně rozpracovali v prvních částech práce.

Jsem rád, že jsem Vás mohl problematikou replikace dat provést, a snad Vám i umožnit nakouknout „pod pokličku“ tvorby aplikace využívající datových zdrojů, jako jsou XML soubory a databáze, neboť bez těchto znalostí se téměř není možné v dnešní době při tvorbě aplikací obejít.

9.1.2 Reference na použitou literaturu

[1] SHARP, John. *Microsoft Visual C# 2005 : Krok za krokem*. [s.l.] : Cpress, 2006. 528 s.

[2] PRICE, Jason. *C# programování databází*. [s.l.] : Grada Publishing, 2005. 624 s.

[3] Microsoft Corporation. *Microsoft Sync Framework Developer Center*. 2008.
www: http://msdn2.microsoft.com/cs-cz/sync/default(en-us).aspx

[4] JAVŮREK, Jan. *Replikace na MS SQL Serveru 7.0* [online]. [2002] [cit. 2008-03-04]. Dostupný z WWW: <http://nb.vse.cz/~zelenyj/it380/eseje/xjavj03/xjavj03.htm>.

[5] ČINČURA, Jiří. *MS Sync Framework | Vyvojář.cz* [online]. 2007 [cit. 2008-03-15]. Dostupný z WWW: <http://www.vyvojar.cz/Series/6-ms-sync-framework.aspx>.

[6] GRMELA, Zdeněk. *Asynchronní XML schémata*. [s.l.], 2008. 80 s. Vedoucí diplomové práce Doc. Ing. Jan Janeček, CSc.

9.1.3 Výčet příloh na CD

Na přiloženém CD najdete:

1. Složku Přihlašovaní (Obsahuje projekt Visual Studia s aplikací Přihlašování)

2. Složku Příklady (obsahující zdrojové kódy příkladů, použitých v práci)

3. Složku Text (obsahující text této práce ve formátu pdf)