VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

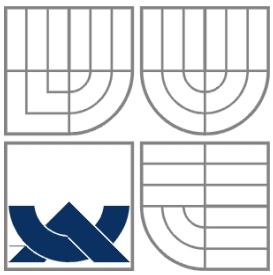
VYUŽITÍ PDA PRO DISTRIBUCI INFORMACÍ V RÁMCI UZAVŘENÝCH SÍTÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

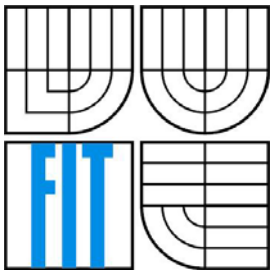
AUTOR PRÁCE
AUTHOR

MARTIN MASLAŇÁK

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYUŽITÍ PDA PRO DISTRIBUCI INFORMACÍ V RÁMCI UZAVŘENÝCH SÍTÍ

PDA FOR INFORMATION DISTRIBUTION IN CLOSED NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN MASLAŇÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Dr. Ing. PETR HANÁČEK

BRNO 2007

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací klient – server aplikace. Klientská část aplikace je tvořena pomoci Compact .NET Frameworku a běží na mobilním zařízení (PDA) . Serverová část je psána v .NET Frameworku a běží na stolním počítači. První část pojednává o PDA zařízení, jeho využití a rozebírá možnosti komunikace s SQL Servery. V teoretické části se zmiňuji o výhodách, které programování pomoci .NET platformy nabízí a o samotné platformě. V části o návrhu aplikace se specifikuje její struktura a také popis architektur server – klient, aby bylo pochopitelné, proč jsem zvolil daný postup. Konec diplomové práce je věnován implementaci aplikace a popisu problému s ní souvisejících.

Klíčová slova

.NET Framework, SQL, Microsoft SQL Server, PDA, WI-FI, ADO.NET, XML, C#, webové služby, Visual Studio 2005.

Abstract

This thesis deals with creating client – server application. Client part of the application is created with a help of the Compact .NET framework and it is running on the mobile facility (PDA). Server part is written on .NET framework and it is running on the desktop computer. In the first part I characterize PDA facility, it's using and also I discuss communication between SQL Servers and PDA facilities. Next part describes .NET platform and advantages, which this platform provides. Also I tried to show differences between client – server architectures, because of understanding in my working. The last part of this work deals with implementation of the client – server application.

.

Keywords

.NET Framework, SQL, Microsoft SQL Server, PDA, WI-FI, ADO.NET, XML, C#, web services, Visual Studio 2005.

Citace

Martin Maslaňák: Využití PDA pro distribuci informací v rámci uzavřených sítí. Brno, 2007, diplomová práce, FIT VUT v Brně.

Využití PDA pro distribuci informací v rámci uzavřených sítí

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Doc. Dr. Ing. Petra Hanáčka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Martin Maslaňák
22.5.2007

Poděkování

Rád bych poděkoval svému vedoucímu Doc. Dr. Ing. Petru Hanáčkovi za podnětné návrhy a příjemné vedení diplomové práce.

Obsah

1 Úvod

Programování vícevrstvých aplikací je rozsáhlou oblastí, ve které se lze setkat s řadou různých technologií, aplikačních serverů , nástrojů a postupů.

Úkolem mé diplomové práce je prostudování tvorby klient – server aplikace pro malé mobilní zařízení (PDA) a jeho implementace za pomoci platformy .NET. Nejhojněji využívaná je právě vícevrstvá architektura klient – serveru. Nejdříve je nutné se seznámit se zařízením PDA, aby bylo jasné, proč se vůbec těmto počítačům věnuje taková pozornost. Stavebním kamenem pro implementování takovéto aplikace je seznámení se se samotnou platformou .NET a všemi prvky, které lze efektivně využít k realizaci tohoto projektu a které tato platforma nabízí (ADO.NET, Compact Framewok, PDA emulátor). Pro zařízení PDA existují schopné programovací prostředky, na druhou stranu, tato zařízení nejsou tak výkonná jako stolní počítače. Proto bylo důležité si při návrhu aplikace vytvořit takový model, aby aplikace těchto prostředků rozumně využívala. Jedním ze základních problémů je připojení PDA zařízení k databázi. Proto se v teoretické části dokumentace zabývám možnostmi připojení a práce zařízení PDA s databázovými servery. PDA nabízí řadu možností, jak se vzdáleně připojit k databázi a kvůli omezené kapacitě tohoto zařízení byla použita pouze databáze na straně serveru.

Aby byla odzkoušena funkčnost samotné klient – server aplikace, bude realizována jako projekt pro malá muzea. Server poběží na stolním počítači a klient na PDA. Komunikace bude probíhat přes wireless LAN (dále WI-FI) neboli přes bezdrátovou síť. Server se bude starat o zasílání příslušných vyžádaných dat do PDA, o správu databáze a o její aktualizaci. Ta bude probíhat v administrační části serveru. V reálu bude uživatel na PDA zadávat kód příslušného produktu a na zařízení PDA se podle zadaného kódu nahraje seznam multimediálních (mp3, video) a textových souborů o daném produktu.

Diplomová práce nenavazuje na žádný předchozí projekt.

2 PDA a současný stav aplikací

O zařízení PDA jsem se začal zajímat až v poslední době. Nejdříve jsem si myslel, že je to jen výstřelek moderní doby, který jak rychle se na trhu objevil, tak rychle i zmizí. Po seznámení se s programováním pomoci platformy .NET jsem zjistil, že i pro tato malá zařízení se dají napsat „velké“ aplikace.

2.1 PDA a jeho využití

PDA

PDA neboli Personal Digital Assistant je u nás známý jako „kapesní počítač“. Většinou jsou to zařízení bez klávesnice (viz. **Obrázek 2-1**), která spadají do kategorie palm size PC. U těchto typů se klávesnice nahrazuje rozpoznáváním psaného písma na dotykovém displeji nebo používáním virtuální klávesnice, kdy se píše pomoci stlačení příslušného obrázku písmena na klávesnici zobrazené na displeji přístroje.

Obrázek 2-1: Typičtí zástupci „kapesních“ počítačů

Tak jak se zvyšuje poptávka po těchto malých počítačích, zvyšuje se také kvalita programů a podpora od společností zabývajících se vývojem programovacích nástrojů pro PDA (Sun, Microsoft atd.). A samozřejmě se zdokonalují samotná zařízení. Na trhu je řada výrobců, kteří se předhánějí v dokonalosti svých zařízení, takže koncový uživatel si už má opravdu z čeho vybírat. V současné době jsou dva používané operační systémy, prvním z nich je platforma Pocket PC s operačním systémem Windows Mobile nebo také Windows CE (dále WinCE). Nejnovější verze je 5.0, ale už od verze 4.2 je podporován .NET Framewok 2.0, což je důležité pro vývoj mé aplikace, protože využívám

programovacích prostředků .NET Framewok 2.0. Druhá platforma je odvozena od operačního systému Palm OS a nazývá se Palm (aktuální verze je Garnet 5.4).[6]

Využití

V poslední době se přenosné počítače třídy Smartphone a Pocket PC těší větší oblibě. Je to především z důvodů rozvoje internetu a mobilních komunikací. Protože je v této době nutnost reagovat rychle na různé podněty (odpověď na e-maily, obchodní transakce, sledování cen v supermarketech atd.), není již PDA jen hračkou, ale plnohodnotným počítačem. Hlavní a nejlépe propracované jsou organizační funkce. Protože se objevuje mnoho aktivit, ať už soukromých nebo pracovních, které se navzájem kryjí, bylo PDA vymyšleno jako interaktivní organizátor času s možností synchronizace se stolním počítačem. S rozvojem technologií se také rozvíjí možnosti těchto zařízení. Většina zařízení je vybavena WI-FI technologií nebo GSM, takže připojení k internetu z tohoto zařízení není ničím neobvyklým. S tím souvisí možnost stahování ebook (internetové knihy) z internetu, prohlížení emailů a používání oblíbených komunikátorů (ICQ, Messenger, Skype). I když není monitor PDA nijak velký (u levnějších přístrojů bývá rozlišení 320x240pixelů, dražších 640x480 a 800x480), jde na něm celkem obstojně prohlížet webové stránky.

Další důležitou vlastností je možnost synchronizace mobilního zařízení se stolním počítačem, obvykle přes USB rozhraní. Ale může probíhat také synchronizace přes bluetooth, WI-FI nebo infračervený port. Pro synchronizaci zařízení s WinCE se využívá program ActiveSync, dodávaný společností Microsoft (viz. **Synchronizace**). Pro většinu nových PDA není problém dokoupit další paměťovou kartu a užívat je pro přenos a ukládání dat (fotografie, videa). I když se nabízí možnost využití jako přehrávače zvukových souborů (mp3 přehrávač), nejsou levnější zařízení úplně vhodná a to především z důvodu mono výstupu na sluchátka. Samozřejmostí je i čtení a psaní textů ve známých formátech. Zařízení vybavené operačním systémem od firmy Microsoft obsahují standardní nástroje sady Office pro práci s textem (Excel, Word).

Jedním z hlavních důvodů, pro pořízení těchto zařízení, je bezesporu využití jako navigátoru. Existuje široká škála GPS modulů, které se dají dokoupit. Od modulů bluetooth, přes moduly připojené kabelem až po zásuvné do slotu PDA. Abychom měli plnohodnotný přístroj, který bude konkurovat jednoúčelovým GPS navigátorům, je potřeba si dokoupit paměťovou kartu (kapacita PDA nedostačuje pro uložení dostatečného množství map) a samozřejmě kvalitní software. Existují i programy, které lze sehnat zdarma , ale jejich použití je omezené (postačují většinou pouze pro navigaci v autě).

V předchozí části jsem popisoval, jaké jsou možnosti použití mobilních zařízení z pohledu „normálního“ uživatele. Ale nesmíme zapomenout, že původně byla tato zařízení spíše doménou obchodníků. V kombinaci s integrovaným telefonem a připojením k internetu se jedná o silný nástroj pro zlepšení prestiže firmy. Už několik let se také pořádají firemní semináře, které učí obchodníky, jak tato zařízení efektivně využívat. PDA přístroje vybavené čtečkou čárových kódů, novodobě se

nahrazuje RFID (Radio Frequency Identification, identifikace na rádiové frekvenci) čtečkou, mají široké uplatnění v komerční oblasti. Ve skladech se zbožím se jich využívá k aktualizaci databáze, inventarizaci majetku nebo zjištění údajů o daném produktu. RFID je ještě poměrně mladá technologie. Vysílače signálů, tzv. RFID tagy jsou docela nákladné. Ale je to logicky další krok, který bude následovat po čárových kódech. Dosahy signálu jsou podle frekvence tagů od 0.2 – 10 metrů a podle napájení až 100m (baterka je tištěna na etiketu). Toho se dá využít při sledování pohybu zásilek po skladu, nebo zavazadel na letišti.

Jak je vidno, zařízení mají široké uplatnění jak v komerční oblasti, tak jako prostředky pro zábavu nebo usnadnění života. Do budoucna možná budeme využívat tato zařízení tak, jako dnes mobilní telefony.

2.2 Současný stav aplikací

Existuje celá řada aplikací, které se zabývají podobnou tematikou. Abych zjistil jak se vůbec aplikace typu klient – server, kdy klient je na mobilním zařízení nebo PDA, vytváří, bylo potřeba prostudovat najít a prostudovat specifikace již existujících aplikací. Dalším podnětem bylo také to, jestli mají takovéto aplikace nějaké širší využití.

Aplikace použitá v nemocnici

Když jsem se začínal zajímat o aplikace, které by mohly být užitečné v reálném životě, narazil jsem na práci pro *University of Washington Neonatal Intensive Care Unit (NICU)*. Jedná se o aplikaci, která umožňuje doktorům zadávání diagnóz pacientů pomoci PDA zařízení, tisk výstupních zpráv, prohlížení karet pacientů, ale také plánovač úloh, pro každého pracovníka nemocnice atd. Tato aplikace byla sice vyvíjena ještě před rozmachem PDA (leden 2001), jaký je v této době, ale jde o dobře zvládnuté objektivní hodnocení jak programovacích prostředků, tak hardwaru, který použít. Hlavní úkol, který si autoři dali bylo snížit papírování v nemocnicích, ale také vyřešit problémy při předávání směn.

Rozhodování o tom, který operační systém použít na PDA, probíhalo mezi Palm OS a WinCE. Jako vítěz vyšel Palm OS. Důvodem nebyla nedostačující funkčnost WinCE, ale v době vývoje aplikace byla v nemocnicích, u kterých přicházelo v úvahu, že by systém používaly, početněji zastoupena zařízení s Palm OS systémem (75%). Další, také docela podstatnou výhodou je, že program pro synchronizaci se stolním počítačem (HotSync), dodávaný se zařízeními Palm OS, je kompatibilní jak s Windows, tak s Macintosh platformou. Klientská aplikace běžící na PDA je programována v jazyce Visual Basic a je používána lokální databáze na mobilním zařízení (Pendragon Forms).

Na počítači běží aplikace, využívající Microsoft Access databázi. Pro předávání dat z databáze na PDA do databáze na stolním počítači je využívána replikace databáze. Jako prostředek na ochranu

citlivých dat je autentizace, jak pro připojení do databáze na klientovi, tak pro připojení databáze na straně serveru. [15]

Myšlenku, že bych využíval databázi na straně klienta, jsem zavrhl už v rané fázi návrhu aplikace, protože by byla docela náročná na kapacitu paměti. Také s využitím wireless LAN připojení jsem od začátku počítal, proto jsem se musel porozhlédnout po dalších možnostech a projektech, které bych mohl použít pro inspiraci.

Dvojjazyčný překladač

Další z aplikací, které jsou již vytvořeny a také používány, je překladač Anglických vět do Japonštiny a naopak. Jde o program, který by měl sloužit cizincům při návštěvách destinací, ve kterých se mluví těmito jazyky. Program je vyvíjen japonskými programátory a má nahradit stávající překladač *Vermobil system* (Wahlster, 2000), kde je klientem mobilní telefon, do kterého uživatel řekne větu, kterou chce přeložit a dostane odpověď od serveru. Problémem byla telefonní kvalita přijímaného signálu. Klient, běžící na PDA se systémem WinCE, přijímá přes mikrofon od uživatele větu v jednom z jazyků. Tuto větu pošle přes wireless LAN serveru, kde běží aplikace na rozpoznávání řeči, která se pokusí danou větu rozpoznat a odešle ji zpět klientovi. K rozpoznání slouží algoritmus, který vytvoří binární strom ze slov věty podle gramatických pravidel daného jazyka. Poté se hledá odpovídající sekvence slov v souboru vět, který obsahuje asi 100000 vět pro každý z jazyků. Poté co je slovní spojení rozpoznáno a přeloženo, zašle se zpět klientovi a přehraje na reproduktoru. Rozpoznávání slov mělo úspěšnost asi 97,5% pro japonské mluvící a 92% pro anglicky mluvící. Úspěšnost překladu byla potom 90% ku 88% ve prospěch překladu z Angličtiny do Japonštiny. [16]

Tento projekt jsem zkoumal ještě v době, kdy jsem neměl jasno, jestli používat i odpojený model aplikace (stand-alone), ale uvědomil jsem si, že to není třeba. Na rozdíl od tohoto řešení je moje klientská aplikace bez připojení k serveru docela nepoužitelná, protože by sloužila pouze jako prohlížeč souborů v jednom adresáři.

Aplikace využívané ve statistice

Jedná se o aplikaci používanou pro sběr dat do kurzů vysokoškolské statistiky na Universitě v Nitre. Tento jednoduchý program, běžící na mobilním zařízení, je psán v Compact .NET Framework a jako úložný prostor je použita MS SQL 2005 mobile edition databázový server. Jde pouze o ulehčení sběru dat v terénu. Předejde se tak zbytečnému přepisování získaných údajů z papírů do počítače. Na serveru, tj. stolním počítači běží program využívající MS SQL 2005, takže student po sběru dat využije možnost replikace dat z databáze klienta do databáze serveru. [17]

2.3 Režimy práce a synchronizace

Mobilní aplikace může pracovat ve dvou režimech. Prvním z nich je režim online, kdy se předpokládá trvalé připojení k síti. Typickým příkladem je sklad se zbožím. Zákazník přijde na přepážku, tam se zjistí, jestli má zboží již zaplacené. Údaje se odešlou skladníkovi na PDA zařízení. Ten pomocí čtečky čárových kódů zjistí, jestli je zboží na skladě a pokud ano, putuje k zákazníkovi.

Druhým ze zmiňovaných režimů je režim offline, kdy se počítá s občasným připojením do sítě. To může probíhat pomoci datového přenosu GSM nebo se mobilní zařízení připojí do sítě (obchodník se vrátí ze služební cesty) a proběhne synchronizace dat.[1]

Synchronizace

Synchronizací dat se rozumí proces, kdy se data, která jsou nová na počítači nebo na PDA, překopírují na zařízení tak, aby byla na obou stejná. V podstatě to znamená, že při první synchronizaci si vybereme co chceme synchronizovat (např. Outlook express, kontakty, schůzky), na počítači se vytvoří osobní adresář uživatele, do kterého se budou ukládat údaje, které se budou po připojení PDA zařízení synchronizovat. Takže uživatel si na počítači přidá kontakt a po připojení PDA se zjistí, jestli jsou kontakty na počítači novější než na PDA nebo naopak. Toto zjišťuje synchronizační program, který, pokud najde na jednom ze zařízení novější verzi, ji přehraje na druhé.

Programy pro synchronizaci se využívají v podstatě dva: jsou to HotSync a ActiveSync. První z nich se používá na PDA zařízeních se systémem Palm OS a druhý na zařízeních se systémem WinCE. Jelikož používám zařízení s WinCE, budu se poněkud podrobněji zabývat pouze programem ActiveSync.

ActiveSync je program, který je umístěný jak na počítači, tak na mobilním zařízení. Stavebním kamenem je práce s PIM daty. PIM daty se rozumí databáze integrovaných aplikací kalendáře, úkolů atd. Při synchronizaci těchto aplikací musí být na obou zařízeních části aplikace, které umí s těmito daty pracovat. Standardem se stal Microsoft Outlook. Problém by mohl nastat při instalaci aplikací, které přímo Outlook nepodporuje. Pokud takováto aplikace obsahuje obě části, jak na počítači, tak na PDA a je vyřešen programově způsob synchronizace, pak to stačí synchronizačnímu nástroji pouze sdělit. Pokud není synchronizace podporována, pak si synchronizační nástroj ukládá na Pocket PC zálohu. Existuje více režimů připojení mobilních zařízení:

- *Standard partnership* – vytvoří trvalou vazbu mezi počítačem a mobilním zařízením, takže je možno synchronizovat soubory aplikací (kontakty, kalendář, úkoly)
- *Guest partnership* – vazba pouze pro přenos souborů a instalaci aplikací. Při dalším připojení stejného zařízení, se bude zařízení tvářit, jako by bylo připojeno poprvé.

Program umožňuje připojení zařízení přes USB, Bluetooth, WI-FI atd. Pro mě je velice důležitou vlastností to, že simuluje připojení Pocket PC emulátoru jako normálního PDA. Bez této vlastnosti, by nebylo možno odzkoušet, jestli aplikace psaná pro PDA pracuje správně. Proces synchronizace je vyvoláván mobilním zařízením a to hlavně z důvodu jednoduchosti. U ActiveSync je trošku problém, že se spouští po startu systému a toto nastavení nelze nikde v programu změnit, takže i když připojíme PDA k počítači jen kvůli nabíjení, je navázána komunikace. Tato maličkost je řešitelná dodatečnými programy. Docela podstatnou výhodou ActiveSync oproti HotSync je přesměrování internetu z hostitelského počítače (samozřejmě pokud na něm běží) do mobilního zařízení.

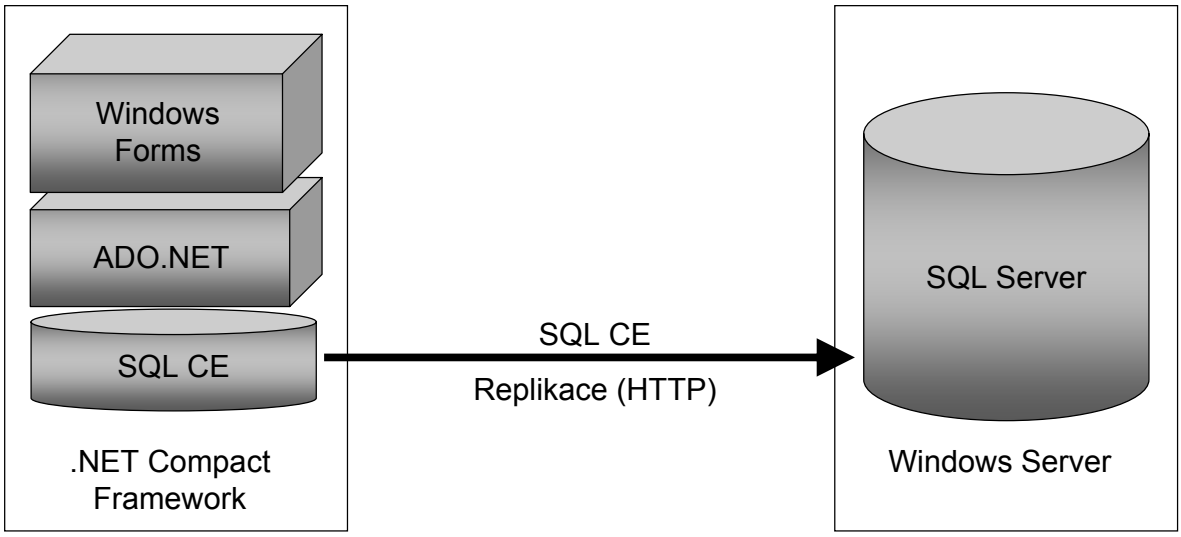
2.4 PDA jako úložiště dat

Jedna z možností, jak se na kapesním počítači dostat k datům uloženým v databázi, je využití databáze přímo na tomto zařízení. Není to příliš obvyklé řešení, a to především z důvodu malé kapacity těchto zařízení. Avšak pro data v textové podobě nebo nemožnost připojit se k databázi jiným způsobem, vyvinula firma Microsoft SQL Server 2005 Mobile Edition (CE). Jedná se asi o dvou megabytový soubor, který, na rozdíl od „velké“ verze SQL Serveru, nepodporuje pohledy, uložené procedury, spouštění triggerů, ani procedurální jazyk T-SQL.

Toto není první server, který je možno na PDA provozovat. Už přístroje vybavené Windows CE od řady 4.1 umožňují provozovat SQL databázi přímo na tomto malém zařízení. Je to produkt MS SQL Server 2000 pro PDA a smartphone. Výhodnější je provozovat databázi na serveru a přistupovat k ní z klientské aplikace (tenký klient, chytrý klient), protože se tak snižují nároky na výkon mobilního zařízení.

SQL CE replikace

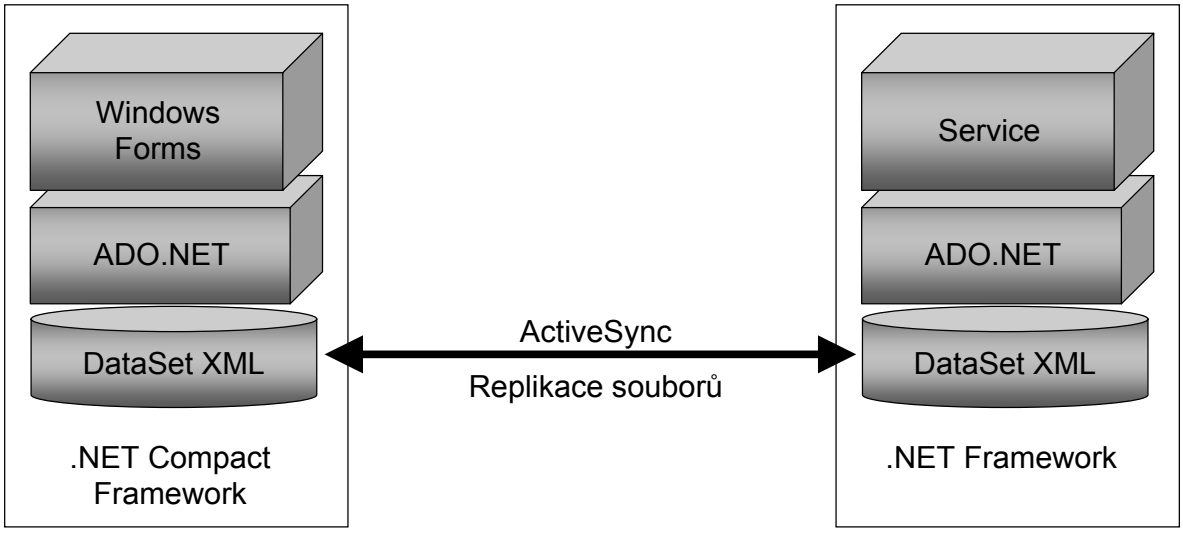
Tento způsob se využívá hlavně tehdy, pokud více uživatelů pořizuje paralelně data v odpojeném offline režimu. Uživatel se připojí k serveru, replikují se dané tabulky podle výběru a server pošle údaje pořízené ostatními uživateli, samozřejmě jen pokud je takto replikace nastavena. Správa replikací se provádí pomocí Enterprise Manageru, který se musí nainstalovat při instalaci Microsoft SQL Serveru. Výhodou je využití databáze, která je na mobilním zařízení a je vždy přístupná, a také možnost používat dotazovací jazyk SQL (Structural Query Language). Synchronizace je realizována pomocí HTTP protokolu. Nevýhodou je nutnost konfigurace serveru a také využívání omezené paměti PDA.



Obrázek 2-2: SQL CE Replikace

Replikace XML souborů pomoci ActiveSync

Jde o poměrně častý způsob sjednocení dat. Například práce více zaměstnanců na jednom dokumentu nebo synchronizace údajů mezi aplikacemi Outlook a Pocket Outlook. Výhodou je možnost práce offline a nevýhodou je připojení pouze jednoho zařízení k jednomu počítači. Jediné co se musí ošetřit je uložení datasetu do XML a naopak. Poté se již jen synchronizují soubory XML, kde musí být jednoznačně dáno, který je dominantní.



Obrázek 2-3: Replikace souborů pomoci ActiveSync

2.5 Vzdálený přístup k databázi

Jde o řešení, které budu používat při implementaci diplomového projektu. Aplikace nebude využívat offline mód, proto by bylo velice obtížné použít některý z výše jmenovaných postupů. Také požadavky na kapacitu PDA budou už tak docela velké, protože se budou ukládat soubory, které si bude chtít klient prohlížet a aplikace je spíše typu jednorázového použití, takže by ani nemělo smysl uvažovat o nějaké lokální databázi na straně klienta.

2.5.1 Posílání dotazu v URL

Tento způsob funguje jen pro MS SQL Server 2000. Budeme – li využívat pro přístup k serveru nějaký prohlížeč HTML stránek, může být dotaz SQL součástí URL (Uniform Resource Locator). Aby byly údaje vráceny ve formátu XML je potřeba dotaz napsat v tomto formátu:

```
SELECT * FROM nazev_tabulky FOR XML AUTO
```

Poslední část dotazu „FOR XML AUTO“ znamená, že výstup dotazu bude tabulka ve formátu XML (jeden dlouhý řetězec).

Ještě před samotným dotazováním je potřeba nakonfigurovat Internet Information Server (IIS) pro podporu virtuálního adresáře. To se provádí následujícím způsobem:

- Na disku se vytvoří adresář, např. *C:\inetpub\wwwroot\HTTP\Virt_adresar*
- Pomoci nástroje *IIS Virtual Direktory Management for SQL server* se přiřadí vytvořenému adresáři virtuální adresář, např. *virtual_adresar*
- Povolí se služby, které se budou na adresář směrovat (*Allow URL Queries, Allow template Queries*, atd.)
- Vytvoří se adresář pro šablony, kvůli efektivnějšímu zadávání SQL dotazů, např. *C:\inetpub\wwwroot\HTTP\Virt_adresar\templates*
- Nastaví se práva příslušnému virtuálnímu adresáři

Dotaz, který vrátí tabulku v XML formátu bude potom:

```
http://nazev_serveru/virtualni_adresar?sql=SELECT+*+nazev_tabulky+FOR+XML+AUTO&root=nazev_korene
```

Toto je ovšem krajně nepraktické, pokud je potřeba získat pouze určitou hodnotu z tabulky. Protože výstup takovéhoto dotazu je XML dokument, pro který se musí vytvořit XSL šablona, ve které se určí formát výstupu do HTML stránky klienta nebo musí programátor vytvořit vhodnou aplikační logiku. Dalším způsobem získání je za pomoci uložené procedury, která se volá v URL. Jednoduchá uložená procedura, která vyhledává záznam podle zadaného parametru, by mohla vypadat takto:

```
CREATE PROCEDURE ParamVypis @Parametr VARCHAR(10)
AS
  SELECT * FROM nazev_tabulky
  WHERE Parametr=@Parametr
  FOR XML AUTO
GO
```

A šablona Procedura.xml

```
<ROOT xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
  <sql:header>
    <sql:param name='Parametr'>Podminka</sql:param>
  </sql:header>
  <sql:query >
    exec ParamVypis @Parametr
  </sql:query>
</ROOT>
```

Volání procedury potom vypadá

```
http://nazev_serveru/virtualni_adresar/template/Procedura.xml?Parametr=nejaky_parametr
```

Aby bylo možno tento způsob používat je zapotřebí nakonfigurovat IIS pro podporu dotazování na MS SQL server. V *Data Source* daného virtuálního adresáře se zvolí instance SQL serveru, na kterou budou dotazy směřovány. Nakonec se musí nastavit název pracovní databáze. Tento způsob se využívá především tehdy, kdy se má výstup dotazu posílat do HTML stránky nebo stačí vrátit jednu hodnotu.

2.5.2 Komunikace s databází pomoci webových služeb

Webové služby

Webová služba je softwarová aplikace, která komunikuje s ostatními aplikacemi prostřednictvím zpráv zapsaných v jazyce XML a přenášených protokoly. Zjednodušeně je webová služba třída obsahující metody, kterou může používat ve svém programu kdokoliv na internetu.

Praktický příklad by mohl vypadat takto: Kdybychom chtěli v aplikaci přistupovat k nějakým konkrétním informacím z vnějšího zdroje, např. aktuálními kurzy měn. Vyhledáme na internetu odkaz na webovou službu, která tyto informace poskytuje. Odkaz vložíme do naší aplikace, která se k webové službě připojí. Po zjištění v jakém formátu má odeslat požadavek, sestaví dotaz a požadavek odešle webové službě. Ta požadavek zpracuje a navrátí aplikaci informace, se kterými můžeme v aplikaci dále pracovat. Celá webová služba se řídí standardy.

Pro komunikaci mezi klientem a serverem se používá SOAP (Simple Object Data Access Protocol). Komunikace probíhá přes protokol HTTP a je uskutečňována výměnou dat XML ve speciálním formátu. O vyměňování dat se stará zprostředkovatel SOAP (pro Microsoft je to SOAP Toolkit), který se skládá z klientské a serverové části. Každá část se stará o zpracování dat na své straně.

Při požadavku na webovou službu se vrací soubor s příponou WSDL (Web Service Description Language), který popisuje danou webovou službu (popis komunikace, názvy procedur, typy parametrů, vazby protokolů).

Pro vyhledání služby se běžně používá registr služeb, kam poskytovatel služby vloží popis a její adresu. Nejčastější implementací je UDDI (Universal Description, Discovery and Integration). UDDI je otevřený obchodní registr založený na XML. Nabízí seznam služeb a kontaktů, v kterých je možno vyhledávat. [14]

HTTP/SOAP Endpoints

Jedná se o novou vlastnost MS SQL Serveru 2005, prostřednictvím které lze využívat webové služby namapované na objekty serveru. Na rozdíl od SQL Serveru 2000 již není potřeba využívat IIS, díky implementaci *http listener* přímo do jádra operačního systému Windows Server 2003 (určen k provozu a poskytování zprostředkovatelských služeb na webu). V první řadě je potřeba určit, který databázový objekt se má zpřístupnit webovým službám. Může to být uložená procedura nebo uživatelsky definovaná funkce. Není možné zpřístupnit tabulku nebo pohled na tabulku. Postup jak využít připojení přes http je následující:

- Vytvoří se uložená proceduru (CREATE PROCEDURE)
- V databázi se definuje a nastartuje http endpoint (CREATE ENDPOINT), který má parametry

- STATE – určuje, jestli je endpoint nastartovaný
- AS HTTP – pro komunikaci se využívá protokol http
- For SOAP – definuje webovou metodu, kterou jsou zaslané požadavky

- Ověříme funkčnost zasláním dotazu, který vrátí popis služby ve WSDL (zapisuje se v XML formátu)

```
https://nazev_serveru/nazev_endpoint?wsdl
```

Endpointů se využívá, pokud se nepracuje s velkými binárními objekty, protože konverze těchto objektů do SOAP zpráv by byla zbytečně náročná. Pro zvýšení bezpečnosti je dobré dodržovat následující pravidla

- Přístupová práva povolit malému počtu uživatelů
- Při komunikaci využívat bezpečný protokol SSL (Secure Sockets Layer)
- SQL Server by měl být umístěn za bránou firewall
- Důkladná autentizace [7]

2.6 Vzdálený přístup pomoci ADO.NET

Jde o rozsáhlou knihovnu tříd, která umožňuje práci s databázemi a datovými soubory. Protože jde o součást .NET Framework, poskytuje ADO.NET stejné prostředky, jaké obsahuje samotný .NET (podpora různých jazyků, automatické uvolňování paměti, OOP). Při realizaci ADO.NET se vycházelo z předpokladu, že databáze není na stejném zařízení jako aplikace, takže přes ADO.NET je spojení s databází navázáno pouze po dobu zpracování dotazu. Výhodou je, že se může připojit hodně uživatelů v relativně krátkém časovém okamžiku, nevýhodou je udržení konzistence dat v paměti a v databázovém systému. V ADO .NET se hlavní důraz klade na tzv. „odpojený model“. Nejdříve se pošle dotaz na databázi, pomocí něj se naplní objekt *DataSet* a s tím se potom pracuje jako s normální databází. Po provedení změn v *DataSet* se provede zpětná synchronizace.

Zprostředkovatelé přístupu k databázi

Microsoft při návrhu ADO.NET předpokládal použití s MS SQL Serverem, proto pouze při této kombinaci může programátor využívat speciální datové typy. Jedním z nich je zprostředkovatel pro MS SQL SqlClient (System.Data.SqlClient), pro ostatní databáze je to OleDb. SqlClient je kompletně napsán v řízeném kódu a při připojení k databázi se snaží využívat co nejmenší počet vrstev, takže by měl být ve většině případů rychlejší než OleDb. Podstatnou výhodou OleDb je, že jsou to vlastně třídy a

metody těchto tříd pro práci s objekty, kde jsou uložena data. Nejde pouze o databázi v pravém slova smyslu, ale také o soubory nebo o XML struktury. Další z možností je připojení pomoci ODBC ovladače, který je již značně zastaralý a nevýkonný.

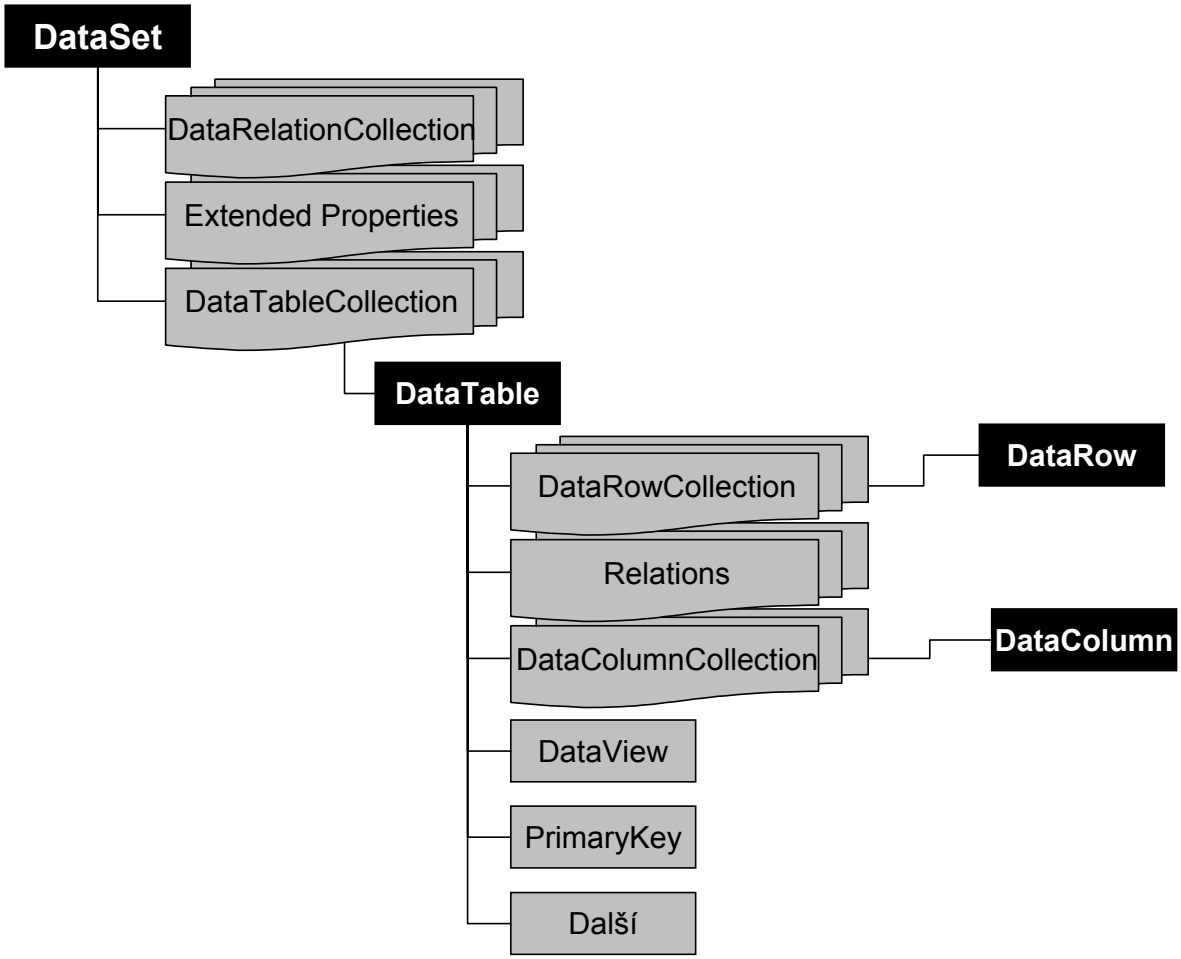
Architektura ADO.NET

Existuje několik základních jmenných prostorů, potřebných pro práci s databázemi. Zajímavé jsou především tyto dva

- System.Date.OleDb – třídy zprostředkovatele OleDb (jiné servery než MS SQL Server)
- System.Date.SqlClient – třídy zprostředkovatele MS SQL Server

Výhoda použití OleDb je hlavně v tom, že je to obecný jmenný prostor obsahující třídy pro práci s jakýmkoliv z databázových serverů (takže i MS SQL) nebo datových úložišť . Pokud není jisté, jak budou uložena data, předpokládá se využití tříd tohoto jmenného prostoru.

Třídy v knihovně ADO.NET lze rozdělit na dvě množiny. První tvoří třídy zaměřené na správu dat (Data Related Components) a jsou zcela nezávislé na použité databázi. Tyto třídy reprezentují umístění dat z databáze v paměti (viz.Obrázek 2-1).



Obrázek 2-4: Struktura databázových dat v paměti

Jak je patrno z obrázku, stavebním kamenem objektu uloženého v paměti, reprezentujícího databázi je DataSet. Představuje databázi ve formátu XML, a jako s databází lze s tímto objektem pracovat. Takže není problém programově přistoupit k jakékoliv tabulce uložené v DataSetu, definovat primární klíč tabulky nebo vztahy mezi jednotlivými tabulkami.

Druhou tvoří třídy závislé na použité databázi (Data Provider Components). Tyto třídy umožňují propojení mezi třídami pro správu dat a databázovými systémy a nejdůležitější jsou

- SqlConnection – navázání spojení s databází
- SqlCommand – provádění SQL dotazů
- SqlDataAdapter – datový adapter, slouží pro přenos dat z databáze do DataSetu
- SqlDataReader – reprezentuje nejjednodušší přístup k datům v databázi. Existují určité nevýhody – pracuje pouze s připojenou databází a v jednom okamžiku může pracovat pouze s jedním řádkem v databázi. Výhodou je, že pro práci nepotřebuje DataSet.

Ke každé z těchto tříd existuje samozřejmě ekvivalent pro práci s jinou databází než Microsoft SQL.

Databáze

Pomoci rozhraní ADO.NET se dá připojit k programu libovolný databázový systém, který má vlastní *OleDb* ovladače. Seznam systémů, které se dají běžně použít je

- MS SQL Server – předpokládá se práce s tímto serverem
- Oracle – konkurenční server
- MySQL – open – source určený pro multiplatformní použití
- Microsoft Access – určený pro tvorbu jednouživatelských databází
- Další

XML

Je využito při většině interaktivních operací. Data v XML se mohou načítat do DataSetu nebo naopak data z DataSetu se mohou ukládat v XML formátu. Pomoci schémat dokumentu XML může program automaticky vygenerovat zdrojový kód v příslušném programovacím jazyce, který vytvoří tabulky podle schématu. Pokud XML soubor neobsahuje schéma, ale pouze data, objekt si strukturu *DataSet* vytvoří sám podle uspořádání těchto dat.[10]

3 Klient - Server

Za posledních 20 let se stala architektura klient – server důležitou součástí při řešení celé řady aplikací. S rozvojem informačních technologií souvisí také množství dat, které je potřeba na počítačích ukládat. Nejlepší možností by byla tato architektura, tzn. vznik serverů, na kterých by byla uložena veškerá data potřebná pro běh aplikace. Klient potom obsahuje aplikaci, která k těmto datům přistupuje a využívá je.

Jednu z těch jednodušších aplikací si lze představit jako seznam kontaktů. Máme například soubor, ve kterém jsou uloženy jména a telefonní čísla v určitém formátu (Jméno příjmení, telefonní číslo). Aplikace bude vědět, jak jsou data v souboru uložena a podle toho si je ze souboru načte nebo je naopak do souboru uloží. Pokud nahradíme soubor tabulkou, těchto tabulek umožníme vytvořit více a doplníme je o určitou logiku (spojování tabulek, pohledy, triggery, uložené procedury atd.) dostaneme databázový server.

3.1 Zpracování dat

Existují dvě obecné možnosti přístupu k načítání a zpracování dat z databáze klientem.

Načítání veškerých dat

Jde o zastaralejší přístup. Strana serveru je využívána pouze jako úložiště dat, veškeré zpracování je realizováno prostředky klienta. Tento způsob se nepoužívá hlavně kvůli velkým nárokům na přenos dat. I kdyby chtěl klient jenom jeden záznam musí se přenést veškerá data z tabulky, ze kterých si klient vybere jenom to co potřebuje. Nároky na klienta byly velké, to je v této době nevyhovující. Protože je čím dál více aplikací běžících na přenosných mobilních zařízeních a jiných přístrojích, kde jsou omezené možnosti výkonu.

Načtení příslušných záznamů

Tento přístup předpokládá, že je možno specifikovat dotaz tak, aby mohla být vrácena pouze data, která klient potřebuje. Databázový server není pouhým úložištěm dat, ale aktivním článkem, který podle požadavku vybere množinu dat, případně ji předzpracuje (záleží na logice serveru) a předá klientovi k dalšímu zpracování. Jasnými výhodami tohoto řešení je ušetření prostředků pro přenos dat a menší zatížení klienta. Toto je také postup, který jsem se rozhodl použít v aplikaci, kterou budu psát. [12]

3.2 Rozdělení architektur klient – server

Monolitické aplikace

Nejstarší aplikace bývaly monolitické (jednolité). To znamená, že byly vytvořeny jako jediný a nedělitelný celek. Provozovat se dají na samostatných počítačích, ale i na sdílených počítačích vybavených terminály. V prostředí sítě se dají využívat jako host/ terminál.

Architektura klient – server

Dalším vývojovým stádiem byla architektura klient/server. Na serveru jsou uložena data, klient slouží k zprostředkování těchto dat uživateli. Stále se ještě počítá s velkým zatížením klienta. Tento model již počítá se síťovým prostředím. Obsahuje – li server nějakou aplikační logiku (např. vrácení specifických údajů podle požadavků klienta), je klient oproštěn od procházení všech dat a je využíván efektivněji.

Vícevrstvá architektura klient – server

Vyznačuje se rozdělením aplikace na tři základní části

- Databázový server – Databáze a nad ní potřebné databázové operace
- Aplikační server – interpretuje data, připojuje se k databázi a vrací potřebná data klientovy
- Klient – prezentace výsledků, sběr dotazů

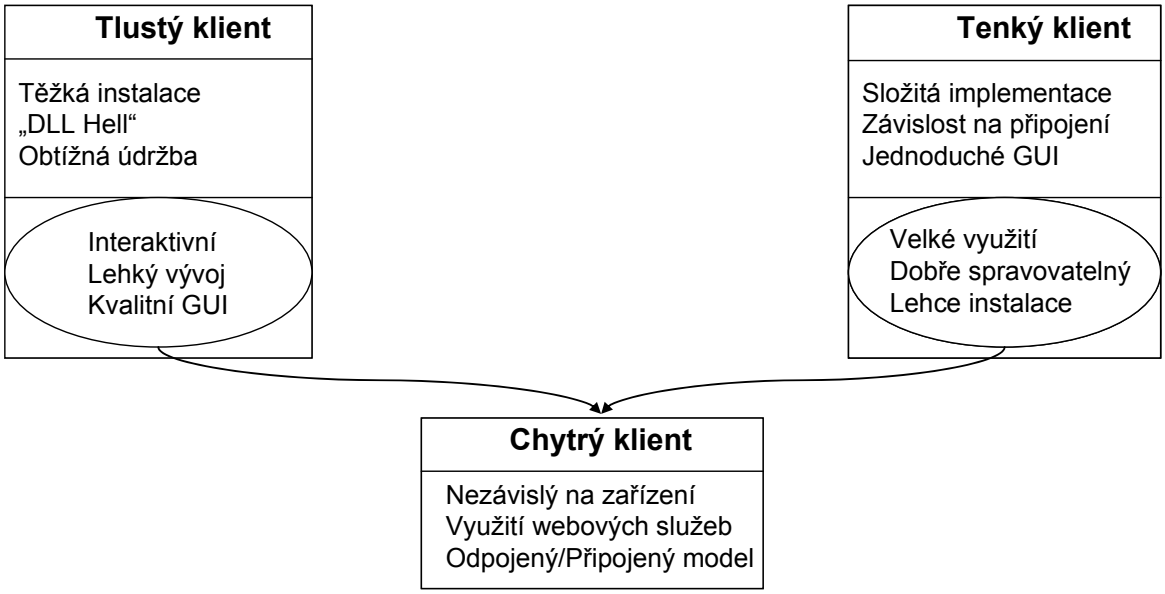
Každá z těchto částí se může ještě skládat z více částí, proto vícevrstvá architektura. Toto je v dnešní době nejpoužívanější postup, i při tvorbě webových aplikací se využívá vícevrstvý model. Jako klient slouží WWW prohlížeč. Jeho hlavní předností je, že pro práci s více různými aplikacemi stačí jediný klient, který pouze realizuje univerzální prezentační funkce. O to, co je specifické pro danou aplikaci, se stará aplikační server. Aplikačním serverem je webová aplikace a jako databázový server slouží některý z dostupných databázových systémů. Poslední vrstvou starající se spíše o zpřístupnění informací je webový server daného zprostředkovatele.

3.3 Rozdělení klientů

Klienti se dělí podle způsobu užití a nároků na server nebo na klientské zařízení. Donedávna existovalo rozdělení do dvou skupin, tenký a tlustý klient. S příchodem platformy .NET vznikl pojem chytrý klient (smart klient). Rozdíly jsou

- Tenký klient – na straně klienta se vykonává minimum aplikační logiky a většina je vykonávána na straně serveru. Klade větší nároky na server a komunikaci. Většinou webový prohlížeč.
- Tlustý klient – většina aplikační logiky je na straně klienta. Klade větší nároky na klienta a má většinou nižší objem přenesených dat než tenký klient.
- Chytrý klient – nabízí komfort klasických stolních aplikací s využitím rozšířených standardů webu (web services, připojený/odpojený model, snadné aktualizace).

Následující obrázek popisuje, jak je chytrý klient provázán s tenkým a tlustým klientem.



Obrázek 3-1: Cíl modelu chytrých klientů

4 Platforma .NET

Jelikož bude aplikace psána v prostředí .NET (dot Net), bude o ní pojednávat tato část. Je to platforma vyvinutá firmou Microsoft. Cílem této platformy je zjednodušit tvorbu programových aplikací a současně zpřehlednit jejich zdrojové kódy. V neposlední řadě pak zjednodušit správu softwarových instalací na konkrétním počítači. Jde o poměrně novou technologii, která se snažila vzít to nejlepší z ostatních technologií. Využívá řadu programovacích jazyků, ale nejoblíbenější pro aplikace je C#, který je syntaxí podobný jazyku JAVA. Microsoft .NET se stává, díky internetovým standardům, platformou webových XML služeb pro zprostředkování vzájemné spolupráce aplikací, služeb a zařízení.

4.1 Architektura .NET

Platforma .NET Framewok stojí jako nadstavba nad operačním systémem. Základem koncepce .NET je *Common Language Runtime (*dále *CLR)*, což je prováděcí prostředí pro všechny .NET aplikace. Kód, který je spuštěn v rámci *CLR*, se nazývá *managed code* (řízený kód), což znamená, že aplikace vyvinutá v libovolném programovacím jazyce či prostředí, se musí řídit podle specifikací určených v prostředí .NET. Jsou vlastně jen dvě, *Common Language Specification (CLS)*, určující povahu vývojového jazyka, a *Common Type Systém (CTS)*, určující předávání hodnot, typy atd. Takové aplikace jsou plně pod správou *CLR*, které odpovídá za jejich správný běh. Kromě toho je ale možné spustit i takzvaný *unmanaged code* (neřízený kód), což je kód, který se neřídí těmito specifikacemi, jako například COM komponenty. Pokud programátor píše aplikaci v neřízeném kódu, nemůže využívat výhod CLR.

Činnost CLR lze popsat poměrně krátce v několika bodech

- Správa kódu, který běží v rámci CLR, kontrola typové správnosti, správa paměti, „garbage collector“, zachytávání výjimek.
- Poskytuje přístup k systémovým službám v rámci Windows API nebo přístup ke COM službám.
- Poskytuje služby propojení aplikací vyvinutých v různých programovacích jazycích.

Jednou z podstatných funkcí, které přináší .NET je *Garbage Collecting*. Jde o automatické uvolňování paměti, tj. objektů z paměti poté, co na ně zaniknou všechny reference. Pro programátora to znamená, že nemůže pracovat s adresami objektů, není možno použít ukazatele. Velkou výhodou Garbage Collectoru je skutečnost, že nemůže dojít k porušení ochrany paměti, nelze se například programově odkazovat na položku, která už v paměti není. Nevýhodou je běh Garbage Collectoru ve

dvou vláknech (jedno se stará o monitorování referencí a druhé o uvolňování prostředků), takže větší zatížení procesoru. Další podstatnou nevýhodou je nedeterministické chování, tzn. za standardní situace není nikdo schopen říct, kdy bude objekt odstraněn z paměti.

Velmi důležitou částí .NET Framework jsou podporované jazyky. .NET Framewok je jazykově nezávislý, pro libovolnou úlohu lze de facto použít jakýkoliv z podporovaných jazyků. Pro psaní webových aplikací jsou nejlepší C# a Visual Basic.NET. Výsledek těchto dvou jazyků bude stejný, pokud jde o funkčnost i výkonnost. Visual Basic.NET připomíná skriptovací jazyky. C# je elegantní, moderní, sevřený, ale obtížněji čitelný než Visual Basic .NET.

Obrázek 4-1: .NET Framewok

- ADO.NET – knihovna pro práci s daty s možností jejich XML reprezentace (viz. **Vzdálený přístup pomoci ADO.NET)**
- Windows Forms – knihovna pro vývoj uživatelského rozhraní pro aplikace spouštěné na stolním počítači
- ASP.NET – knihovna pro vývoj uživatelského rozhraní pro webové aplikace.

4.2 Integrace vývojových prostředí

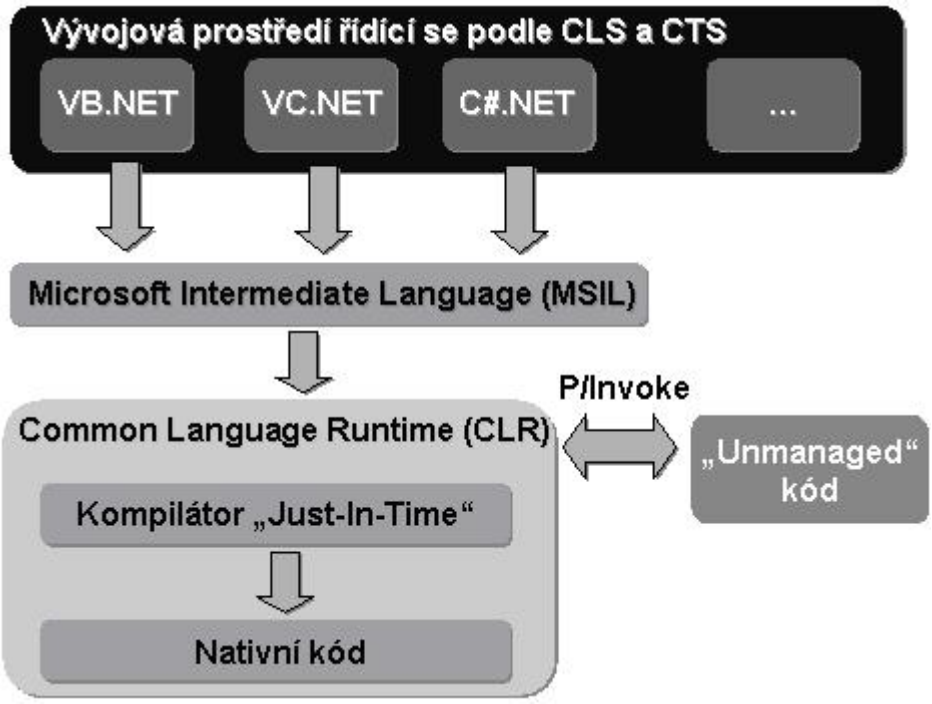
Program, pro který chceme využívat prostředky prostředí .NET, je otevřen v podstatě libovolnému vývojovému nástroji nebo programovacímu jazyku, který dokáže svůj kód kompilovat do *Microsoft Intermediate Language* (MSIL).

Microsoft v tomto ohledu nabízí své Visual Studio .NET, které je možné pokládat za špičku v oboru vývojových nástrojů a které je s prostředím .NET zcela integrováno. Kromě toho mohou i jiní programátoři zůstat věrní svým nástrojům, pokud dodavatel jejich vývojového prostředí vytvořil kompilátor do MSIL kódu. Primárně jde o snahu společnosti Microsoft zvýšit přenositelnost kódu na různý hardware (PDA, smartphone, tablet PC, 64-bitové procesory), na kterém běží Microsoft Windows. I když je MSIL podobné assembleru, neexistuje v praxi procesor, který by ho uměl vykonávat. Proto se musí pomocí *Just-in-Time (JIT)* kompilátoru přeložit do strojového jazyka. Existují tři druhy JIT kompilace:

- Kompilace v době instalace - toto řešení však není pravá *Just-in-time* kompilace, protože se neprovádí při spouštění programu, ale již při jeho instalaci. Výhodou je, že se *MSIL* kód překládá pouze jednou a pak při každém spouštění se spouští strojový kód. Nevýhoda je v tom, že pokud se nainstalovaný exe nebo dll soubor zkopíruje na jiný počítač, není možné aplikaci spustit. Je tedy nutná opětovná instalace.

- Skutečný *JIT* překlad - .NET aplikace je spuštěna a *MSIL* kód je najednou přeložen do strojového kódu. Velkou výhodou je, že tento kód je přímo optimalizovaný na konkrétní stroj, na kterém byl spuštěn a může proto naplno využít jeho potenciál. Nevýhodou je však přítomnost obou verzí programu v paměti - jak strojový, tak *MSIL* kód. Ten je však odstraněn hned po dokončení kompilace. Druhá nevýhoda spočívá ve výkonnostní režii *JIT* kompilace, která je potřeba při spouštění aplikace. Jakmile je však kód přeložen do strojového kódu, je prováděn stejnou rychlostí jako jakýkoli jiný strojový kód.

- Ekonomický *JIT* překlad - jde o *skutečný JIT překlad*, ale při překladu *MSIL* do strojového kódu jsou vypnuty všechny optimalizace jak pro zmenšení, tak pro zrychlení programu. Druhou jeho vlastností je inteligentní překlad programu, tzn. nepřeloží se celý program najednou, ale jenom jeho prováděná část. Volání funkcí, které zatím nejsou přeloženy, se nahradí tzv. stubem (krátký kód), který se navenek tváří jako plnohodnotná funkce, avšak při jejím zavolání dojde k přeložení další části programu a jejímu vykonání.



Obrázek 4-2: Integrace vývojových prostředí **[2]**

V praxi to vypadá následovně, program napsaný pro .NET o jehož provádění se stará *CLR* je na výstupu kompilátoru kompilován do *MSIL*. A podle *JIT* kompilátoru kompilován do strojového jazyka a prováděn. [3]

4.3 Programování v .NET

Jak se vlastně programuje v .NET Framework? Jde o plně objektově orientovaný programovací nástroj. Už výše jsem uvedl, že vychází i z Javy, a to proto, že Microsoft musel reagovat na vzrůstající oblibu Javy. To se do jisté míry povedlo, problém pro Microsoft vznikl možná v tom, že reakce přišla docela pozdě. Velké aplikace jsou psány v Javě (informační systémy bank), další aplikace, které musí být svižné se píší v C (aplikace pro ovládání letadel) a pochybuji, že by se v nejbližší době chystal je někdo do prostředí .NET přepisovat. Každopádně kdokoliv začne psát aplikaci využívající všeho co mu .NET přináší bude spokojen. Vedle plně objektového programování se klade velký důraz na bezpečnost kódu. Neexistují ukazatele, problémové části kódu jsou řešeny přes výjimky. Je kladen velký důraz na typovou přesnost, a to z důvodu předcházení problémům, které by mohly nastat a jejichž hledání by zabralo hodně času.

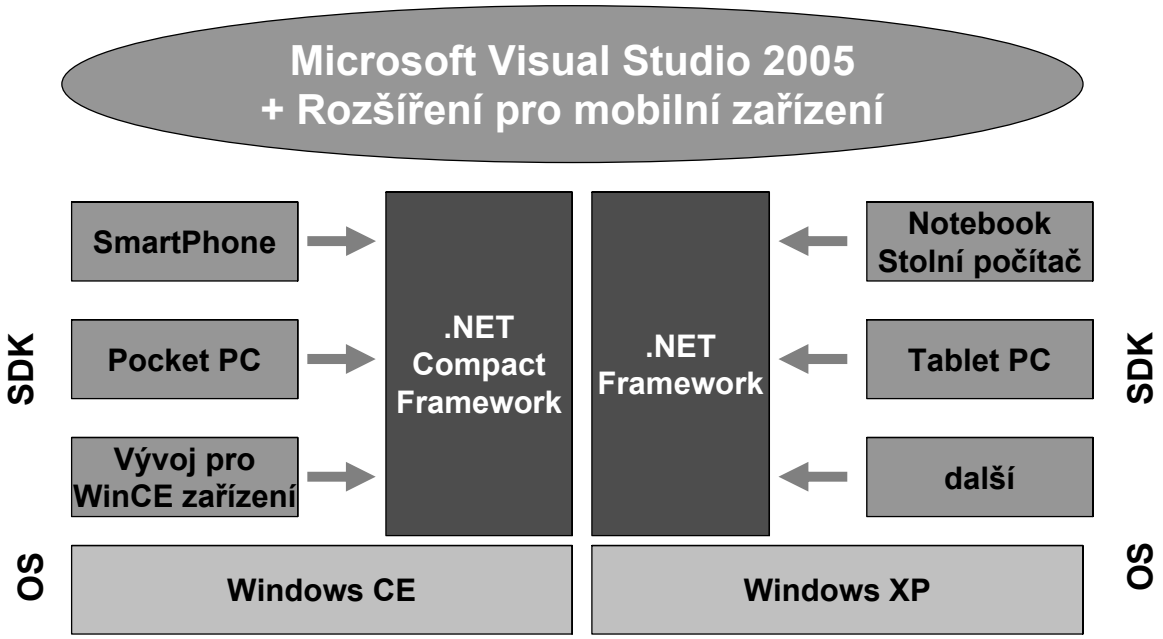
Compact .NET Framework

Jedná se o část prostředí .NET Framework. Jako programovací jazyk může být použit buď Visual Basic .NET nebo C#. Na rozdíl od psaní standardního programu v prostředí .NET není možné tyto dva jazyky kombinovat v jedné aplikace pro mobilní zařízení.

Vývojové prostředí Microsoft Visual Studio .NET doplněné o balík *Smart Device Extension* zastřešuje vývoj aplikací pro stolní počítače i pro mobilní zařízení. Při respektování rozdílů mezi .NET Framework a Compact .NET Framework (CF .NET) můžeme vyvíjet aplikace pro různé druhy zařízení. Protože je CF .NET ořezanou verzí obsahuje asi jenom čtvrtinu metod a funkcí, které obsahuje .NET. Pokud nemáme možnost odzkoušet si aplikaci na skutečném PDA zařízení, můžeme použít emulátor Pocket PC 2003, který je obsažen v MS Visual Studiu 2005.

4.4 Microsoft Visual Studio 2005

Visual Studio 2005 je určeno pro programování klasických stolních, serverových, webových (ASP.NET) i mobilních aplikací běžících na platformách Windows. Z programovacích jazyků jsou k dispozici Visual C++ (nativní i řízené), Visual C#, Visual Basic a Visual J#. Produktová řada Visual Studia 2005 zahrnuje: VS Standard Edition, VS Professional Edition, VS Tools For Office, VS Team System a minimalistickou edici VS Express (který je na jeden rok zdarma). Součástí Visual Studia je základní verze MS SQL Server 2005 Express.



Obrázek 4-3: Přehled nástrojů a možností pro vývoj aplikací

Jak je vidět na obrázku, Visual Studio 2005 obsahuje nástroje pro vývoj nepřeberného množství zařízení. Pokud bych měl srovnat práci v tomto „editoru“ s ostatními, se kterými jsem měl možnost pracovat (JDeveloper, C++ Builder, Eclipse, MinGW a další) jde zatím o nejkomplexnější a nejpohodlnější nástroj pro vývoj aplikací. V kombinaci s velkou podporou firmy Microsoft, sice obsáhlou, ale přehledně zvládnutou dokumentací (*MSDN*), a různými drobnostmi, které psaní kódu zpříjemní (intellisense, možnost rozdělení definice třídy do více souborů a oddělení definice komponent formulářů od jejich funkčnosti) se jedná o špičku v kategorii vývojová prostředí.[5]

4.5 Porovnání J2EE a .NET

Přece jenom existuje nějaká souvislost mezi .NET a Java Enterprise Edition. Rozhodl jsem se proto, že uvedu menší srovnání výkonnosti. Na tento test jsem narazil při procházel internetu, ještě v době kdy jsem se rozhodoval, jestli programovat v J2EE nebo .NET. Testy, které jsou použity jsou příklady ze skutečných aplikací, nejde jen o matematické výpočty. Autorem testu je pan Kouba Tomáš. [18]

Test výkonnosti Javy a C# oproti C

Testovaný program	Java (s)	C# (s)	C (s)	Java vs C (%)	C# vs C (%)	Java vs C# (%)
Ackerman	0,25	0,71	0,40	160,00	56,34	284,00
Array access	0,30	0,24	0,04	13,33	16,67	80,00
Count Lines/Words/Chars	0,21	0,13	0,02	9,52	15,38	61,90
Echo client/server	7,55		9,85	130,46		
Exception	1,55	2,75	0,05	3,23	1,82	177,42
Fibonacci numbers	0,26	0,16	0,51	196,15	318,75	61,54
Hash Access	2,52	0,90	0,77	30,56	85,56	35,71
Hello world	0,21	0,10	0,02	9,52	20,00	47,62
List Operation	8,32	0,25	0,10	1,20	40,00	3,00
Matrix Multiplication	7,85	14,24	2,36	30,06	16,57	181,40
Method Calls	8,35	4,24	9,78	117,13	230,66	50,78
Nested Loops	5,54	5,16	2,33	42,06	45,16	93,14
Object Instantiation	7,67	1,47	19,04	248,24	1295,24	19,17
Producer/Consumer Threads	6,47	9,89	8,51	131,53	86,05	152,86
Random Number Generator	7,97	6,16	1,22	15,31	19,81	77,29
Reverse a File	3,52	1,04	0,12	3,41	11,54	29,55
Sieve of Erathostenes	8,27	14,70	4,27	51,63	29,05	177,75
Spell Checker	0,78		0,14	17,95		
Statistical Moments	0,37	0,15	0,06	16,22	40,00	40,54
String Concatenation	0,82	0,29	0,15	18,29	51,72	35,37
Sum a Column of Integers	0,30	0,11	0,06	20,00	54,55	36,67
Word Freaquency Count	0,54		0,06	11,11		
Průměr				**19,15**	**51,72**	**86,62**

Tabulka u každého jazyka uvádí, jak dlouho probíhalo provádění daného programu (v sekundách). Poslední tři sloupce potom symbolizují procentuální rychlost oproti ostatním jazykům (čím větší, tím je provádění programu v tomto jazyce rychlejší). Nejde vlastně ani tak o jazyk, jako o platformu. Takže Java symbolizuje J2EE a C# v podstatě .NET. Rozdíl mezi objektově orientovanými jazyky (OOJ) a strukturálním C jazykem je patrný. To je dáno náročnějším překladem OOJ do nativního kódu. I když je průměr dost zavádějící termín, uvedl jsem ho spíše pro orientaci. Jde vidět, že je .NET o něco rychlejší než Java, ale rozdíl není určitě nijak markantní.

5 Návrh intranetové aplikace

Cílem mé diplomové práce je klient – server aplikace, která umožní uživateli po zadání příslušných údajů do formuláře na přístroji PDA, zobrazit multimediální (mp3, video) soubor nebo textový soubor o příslušném produktu. Aplikace využívá třívrstvé architektury klient/server. Funkčnost budu testovat na příkladu malého muzea.

5.1 Popis aplikace

Pro lepší pochopení dané aplikace uvedu příklad z praxe. Předpokladem je, že zařízení PDA je již vybaveno příslušným softwarem (tzn. obsahuje klientskou aplikaci). Pro širší využití aplikace se bude klient připojovat k serveru až po vstupu do zmíněného muzea (knihovna, výstava). Pro připojení k serveru bude k dispozici formulář, do kterého se zadá jméno serveru (případně IP adresa). Návštěvník přijde k exponátu a bude mít zájem dozvědět se více o dané položce. Do formuláře na přístroji PDA zadá unikátní řetězec, který bude charakterizovat daný exponát. Proběhne kontrola databáze, aby se zjistilo, které soubory jsou dostupné o daném exponátu. Pokud bude pro daný exponát existovat textový dokument s doplňujícími informacemi, dále fotografie a multimediální soubor mp3, nabídnou se uživateli tyto soubory. Podle toho kterou možnost zvolí, stáhne se daný soubor z databáze a zobrazí se uživateli na přístroji PDA v aplikaci, která je pro typ daného souboru určena.

5.2 Server

Bude realizován v prostředí .NET 2.0. Půjde o WinForms aplikaci, která se bude starat o připojení klientů. Bude možno na serveru sledovat, které zařízení (IP adresa) a kolik jich je k serveru připojeno. Aplikace bude více vláknová

- Hlavní vlákno – bude se starat o aplikaci samotnou, vykreslování informací o připojených uživatelích, protokolování, možnost odpojení uživatelů.
- Vlákno pro zjištění připojení – toto vlákno bude naslouchat, jestli se připojuje některý z klientů k serveru
- Uživatelská vlákna – budou se starat o jednotlivé uživatele, obsluhovat smyčku zpráv

Po spuštění aplikace serveru se spustí i vlákno pro zjištění připojení. Pokud se uživatel připojí (nebo odpojí), nastartuje se pro něj nové vlákno a aplikace serveru překreslí aktuální připojené uživatele.

V uživatelském vlákně bude probíhat obsluha smyčky zpráv. Klient bude serveru zasílat zprávy a podle číselného označení zprávy bude server vykonávat určitou činnost. Zprávy budou následující

- Požadavek na zaslání dat o příslušném exponátu
- Požadavek na zaslání souboru
- Odpojení uživatele
- Čekání na další zprávu

Komunikace mezi serverem a klientem bude probíhat přes WI-FI (wireless LAN), proto počítač na kterém bude nainstalován server musí být vybaven WI-FI kartou. Pro účely testování bude stačit emulátor, který je součástí MS Visual Studia 2005. Finální verze bude však odzkoušena přímo se zařízením PDA s nainstalovaným klientem, server poběží na stolním počítači.

Administrační část

Bude obsažena v serverové části aplikace. Administrátor bude mít možnost přidávat do databáze položky, upravovat je a rušit. V databázi budou uloženy pouze cesty k příslušným souborům. Samotné soubory budou uloženy na disku v adresářích. Pro každý typ souborů bude speciální adresář, tzn. textové soubory budou v adresáři TEXT, videa v adresáři VIDEO atd.

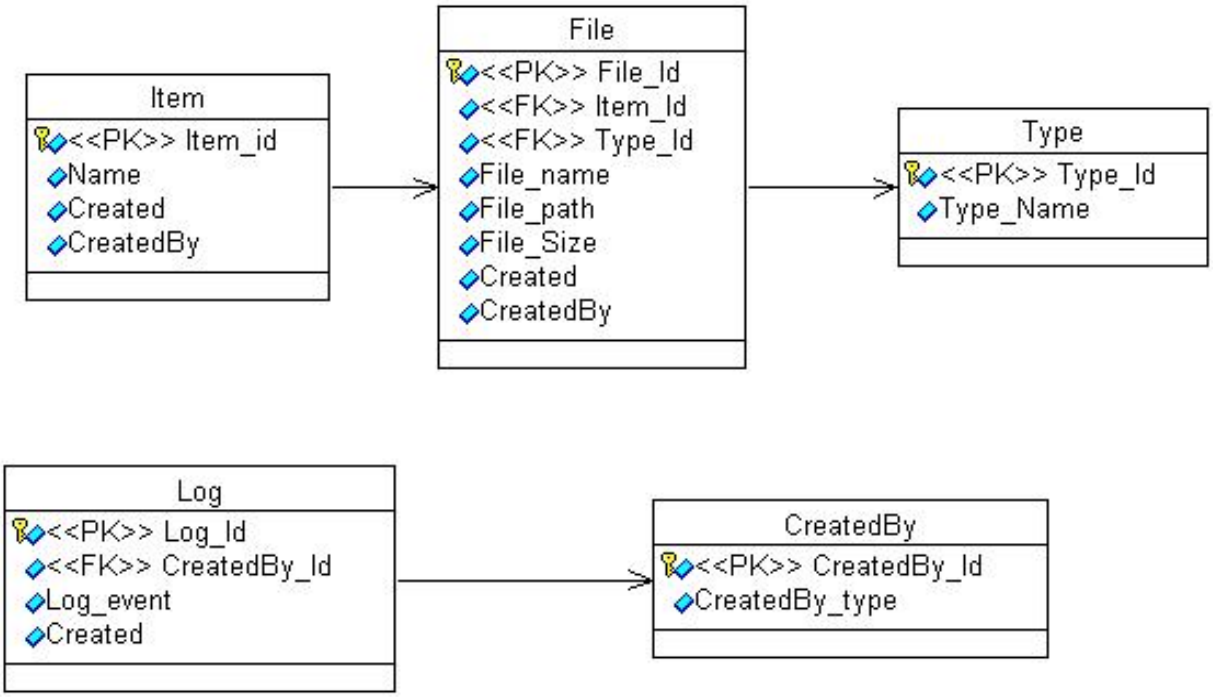
Protokolování

O protokolování se bude starat serverová část aplikace. Pro protokolování bude vytvořena v databázi speciální tabulka, kde se budou ukládat právě probíhající zprávy a výjimky, pokud k nim dojde. Pro lepší přehlednost, bude záloha ukládána také do textového souboru na disku. Tento soubor bude mít následující formát: datum čas Název události. Typ, tzn. jaká je to událost (vyvolaná serverem, výjimka, vyvolaná klientem). Příklad některých událostí

```
07.01.2007 10:15:22 Client 1 connected from ip: ip_adresa. Client
07.01.2007 10:15:24 Client 1 received list of exponat files fileNum 10. Server
```

Databáze

Jako server pro ukládání dat bude využit Microsoft SQL Server 2005. Pro připojení k databázi a pro práci s daty využívám ADO.NET. Jak bude vypadat databáze ilustruje následující obrázek



Obrázek 5-1: Diagram tříd

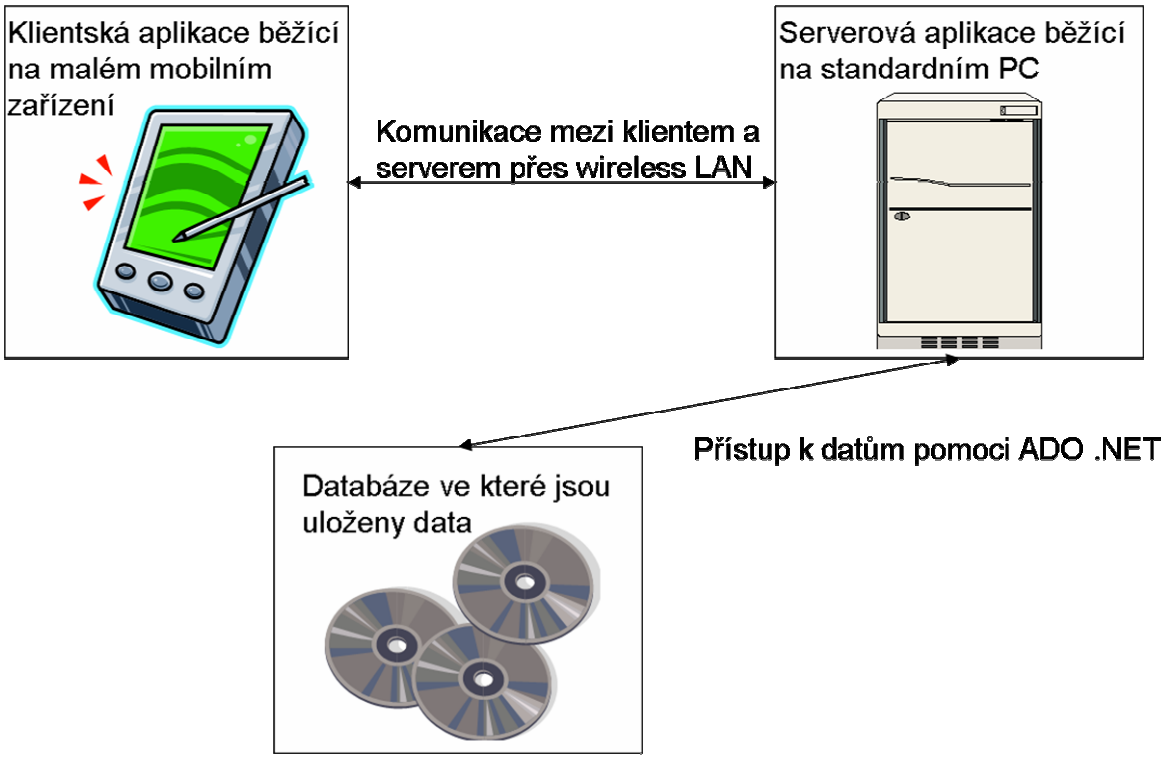
5.3 Klient

Klient bude na zařízení PDA, k realizaci budu využívat Compact .NET Frammewok 2.0. Tento Framewok je možno nainstalovat na zařízení s WinCE 4.2 a vyšší. Nejdříve bude aplikace odzkoušena na emulátoru, který je součástí MS Visual Studia 2005. Klient bude typu tlustý klient. Klientská aplikace bude obsahovat jednoduchý formulář, do kterého bude klient zadávat příslušná data (číslo exponátu). Po připojení k databázi a zjištění, které soubory jsou pro tento exponát dostupné se zobrazí formulář se záložkami (*text, picture, video, sound*). V každé záložce budou zobrazeny názvy dostupných souborů daného typu. Uživatel bude informován, pokud pro danou kategorii nebude existovat žádný soubor. Pokud si bude chtít prohlédnout soubor z jedné z kategorií, klikne na daný soubor. Klient pošle zprávu serveru a vyžádá si příslušný soubor, který se po stáhnutí do mobilního zařízení automaticky spustí.

Nebude – li zrovna uživatel využívat připojení k serveru (tzn. nebude žádat o poslání seznamu souborů nebo o soubor samotný), bude jeho vlákno uspáno. Probuzeno bude opět při odeslání jedné ze zpráv znamenající potřebu určité činnosti serveru. Při odhlášení od serveru nebo při odpojení (ať už ze strany serveru, nebo kvůli nedostupnému připojení), se implicitně všechny soubory uložené na zařízení PDA odstraní.

6 Implementace

V této části se pokusím nastínit, jak jsem implementoval server – klient aplikaci v praxi. Při řešení se vyskytla také celá řada problému nebo situací o kterých bych se rád zmínil. Při implementaci a návrhu některých částí aplikace jsem vycházel z již dříve napsaných aplikací (viz. **Současný stav aplikací**). Tato aplikace je třívrstvá aplikace, tak jak interpretuje obrázek.



Obrázek 6-1: Třívrstvá architektura klient server

6.1 Komunikace

Při realizaci aplikace jsem se nejdříve zaměřil na komunikaci mezi klientem a serverem, tzn. pokusil jsem se z klienta připojit, odeslat si nějaký testovací řetězec a zjistit, jak taková komunikace vůbec probíhá. Hlavním důvodem bylo vybrat správné programovací prostředky, které .NET nabízí. S protokolem, který budu využívat, nebyly problémy, ale musel jsem se rozhodnout, jestli používat třídy TCPClient a TCPServer nebo obecnější socket (viz. **Socket**).

6.1.1 Protokoly

Komunikační protokoly jsou vlastně určitým souborem pravidel, potřebných pro přenos, příjímání a odesílání dat mezi dvěma počítači. Jde o jistou potřebu standardizovat výměnu dat mezi různými

hardwarovými platformami a operačními systémy. Rozhodování, který z protokolů použít nebylo až tak složité. V úvahu připadaly dva TCP/IP a UDP.

Paket

Jde o základní jednotku informačního přenosu ve všech moderních počítačových sítích. Skládá se ze tří základních prvků.

Hlavička	Datová oblast	Trailer

Hlavička obsahuje adresu potřebnou pro doručení paketu. Datová oblast jsou informace, které se odesílají a trailer nejčastěji obsahuje potvrzení bezpečnosti přenosu.

TCP/IP

Je dnes nejpoužívanějším a nejnovějším síťovým protokolem. Jedná se o soustavu protokolů, kde nejdůležitější jsou dva TCP (*Transmission Control Protocol*) a IP (*Internet Protocol*). IP protokol pracuje na síťové vrstvě modelu ISO/OSI a je protokolem spojově neorientovaným. Přijímá pakety, přidává do nich hlavičku a odesílá data na příslušné adresy, ale nekontroluje, zda data dorazila v pořádku.

Pro kontrolu správnosti doručení dat, byl vytvořen protokol TCP. Využívá služeb IP protokolu. Pokud se zjistí, že data nedošla v pořádku, znova je pomocí IP protokolu odešle. Na straně příjemce přeskupováním paketů zajišťuje správnost pořadí.

UDP

Jde o tzv. nespolehlivý protokol ze sady protokolů internetu. Tento protokol nezaručuje, zda data dorazí ve správném pořadí, jestli se nějaký paket neztratí nebo se doručí několikrát. Po odeslání dat si UDP vrstva u odesilatele neudržuje žádný stav, proto jde o rychlejší protokol než TCP/IP. Jediné, co UDP protokol přidává k odesílaným datům je kontrolní součet a schopnost roztřiďovat pakety mezi více aplikací běžících na stejném počítači. UDP protokol je vhodný pro přenos dat, kde je kladen velký důraz na včasnost doručení. Používá se například pro streamovaná média, přenos hlasu, videokonference nebo online hry.

Protože moje aplikace je založena na přenosu souborů, řetězců a XML soborů, potřebuji aby byla data dodávána ve správném pořadí. Proto jsem jako komunikační protokol vybral TCP/IP, který vyhovuje potřebám mé aplikace více než UDP.

6.1.2 Socket

Tuto část bych chtěl věnovat vysvětlení rozdílu mezi synchronními a asynchronními operacemi třídy *socket* jmenného prostoru *Systém.Net*. Nejdříve by bylo vhodné popsat, co to *socket* vůbec je. Jde o datové spojení mezi dvěma koncovými body. V případě použití TCP protokolu je koncovým bodem určitý port na počítači. Socket nabízí práci s více protokoly než jen TCP.

Synchronní mód

V tomto režimu se operace na socketu provádí ve stejném vlákně, ve kterém byly volány a způsobí zablokování vlákna do doby, než jsou provedeny. Pokud vlákno, ve kterém je operace volána obsahuje uživatelské rozhraní nebo jiné procesy vyžadující rychlou reakci programu, může to způsobit problémy. Proto se většinou synchronní operace volají ve vlastním vlákně.

Asynchronní mód

V tomto režimu si operace, kterou socket provádí zavolá přes asynchronního delegáta provádění funkce v jiném vláknu, než byla volána. O ukončení operace se může program dozvědět například pomoci zpětného volání (*callback*). Metody pro asynchronní mód jsou rozděleny na dvě části, první je vyvolání operace (např. *BeginConnect*) a druhá část je přečtení výsledku po jejím dokončení (*EndConnect*). Ještě je potřeba signalizovat programu, že je operace dokončena. To lze udělat pomoci funkce Set identifikátoru typu ManualResetEvent. [19]

6.2 Klientská aplikace

Při realizaci klientské části diplomového projektu jsem prostudoval celé množství aplikací, které se zabývají více či méně podobnou tématikou. Pokusil jsem se napsat aplikaci tak, aby byla jednoduchá na ovládání a zároveň obsahovala vše, co by se dalo od tohoto typu aplikace očekávat.

6.2.1 Aplikační část

Klientská aplikace je psána v Compact .NET Framework. Jedná se o synchronního klienta, ale všechny operace pro komunikaci (připojení, posílání dat a přijímání dat) jsou asynchronní. Synchronní klient, na rozdíl od asynchronního klienta, po vyslání požadavku na data přijme data a potom pracuje dále.

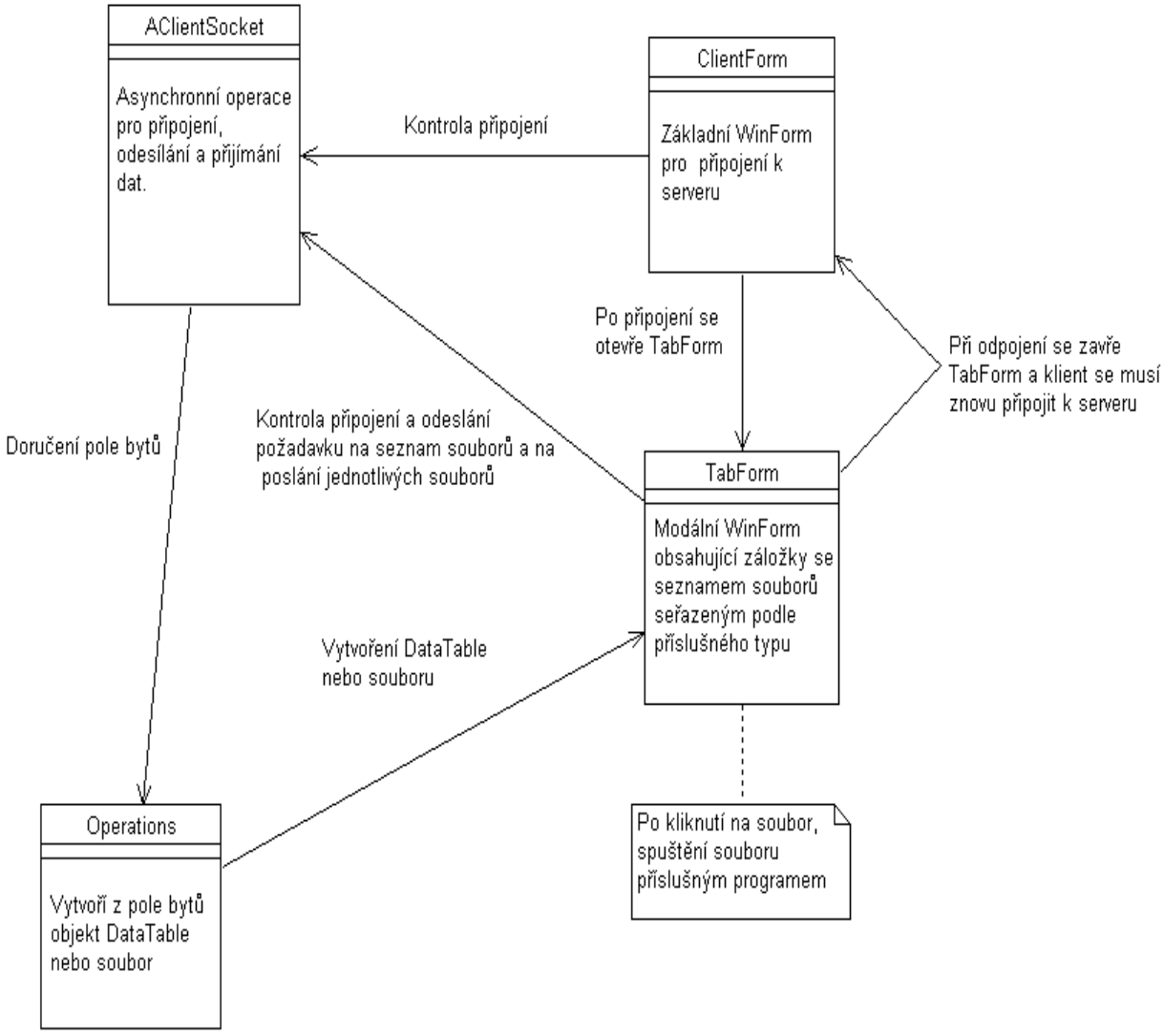
Asynchronní klient nečeká až přijme data, ale rovnou pokračuje v provádění programu. Jednou z nejdůležitějších částí je komunikace klienta se serverem a vůbec zjištění dostupnosti připojení. Při implementaci komunikace jsem se setkal s „podivným“ chováním emulátoru.

Popis aplikace

Základem aplikace je *WinForm*, jde o přihlašovací dialog klienta na server, ve kterém se po připojení spouští další formulář jako modální okno. Tento modální formulář obsahuje vlastní funkčnost aplikace, tj. odesílání čísel exponátů, zobrazování souborů a další (viz dále).

Na rozdíl od serveru jsem neviděl potřebu obsluhu klienta řešit pomoci vláken, protože se vlastně jedná o docela jednoduchou aplikaci, která slouží pouze k zpřístupnění souborů z databáze přes wireless LAN. Složitost této aplikace je v tom, že je psána v Compact .NET Framework a testována z velké části na PDA emulátoru, který se nechová vždy tak, jak by se dalo očekávat.

Klient se k serveru připojuje pomoci IP adresy serveru, tzn. musí mu být předem známa.



Obrázek 6-2: Diagram činnosti klienta

Testování připojení

Aby se emulátor choval alespoň z části jako standardní PDA připojené k počítači, je potřeba nastavit režim přemostění (*cradle*) po spuštění emulátoru. Poté použijeme program ActiveSync, který po spuštění automaticky detekuje standardní zařízení PDA. Bez tohoto postupu jsem nebyl schopen se z klientské aplikace k serveru vůbec připojit. Když jsem vyřešil tuto „drobnost“, objevila se další: tentokrát se při volání metody *socket.connect* připojilo zařízení i k serveru, když nebyl spuštěn. I z tohoto důvodu jsem testoval připojení k serveru jinak, než jsem původně zamýšlel. Prvotní návrh počítal s časovačem (*Timer*), který by běžel v hlavním vlákně a testoval dostupnost připojení v určitém časovém intervalu odesíláním prázdného řetězce serveru. Toto řešení by bylo náročné na baterii zařízení a když jsem vzal v potaz podivné chování emulátoru, rozhodl jsem se postupovat jinak. Nakonec testování připojení probíhá při každém požadavku klienta na připojení k serveru. Takže pokud uživatel zadá číslo položky a chce od serveru získat soubory k této položce, tak před odesláním čísla položky odešlu prázdný řetězec serveru a v případě nedostupného připojení zachytím výjimky, které tato metoda vrací. Podle příslušné výjimky potom jednoduše poznám co se stalo.

Zasílání zpráv

Jelikož server obsahuje smyčku, ve které je stavový automat, který zjišťuje jaká data se mají posílat zpět klientovi, je potřeba si vytvořit identifikující řetězec tak, aby obsahoval všechno potřebné. Vytvořil jsem si následující řetězec, který klient posílá na server

```
messageId:exponatNumber:fileId:Time
```

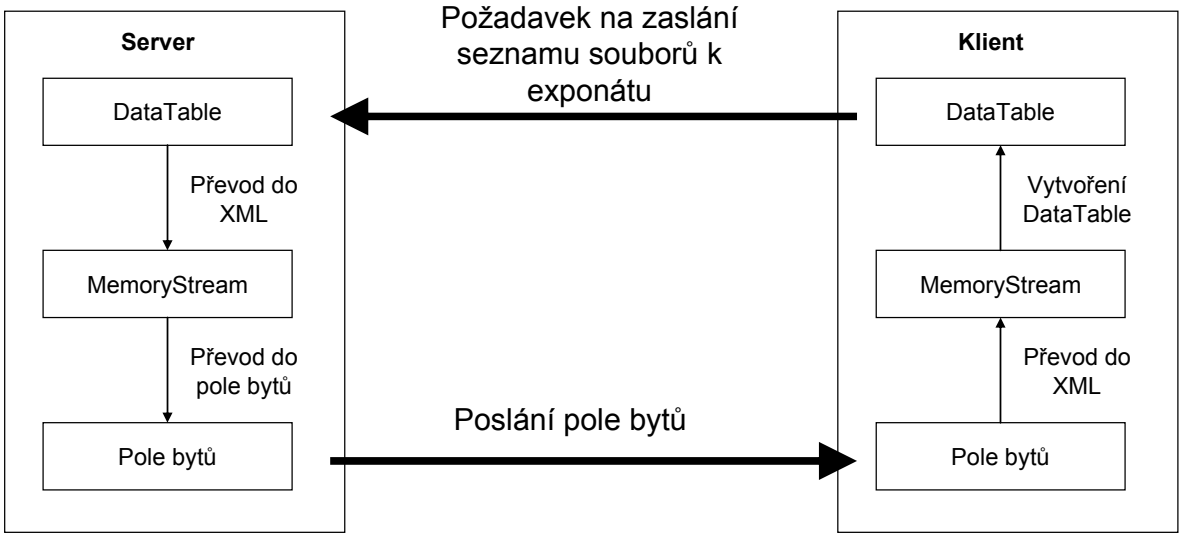
- MessageId – identifikátor zprávy, tj. jestli se klient připojil, vyžádal si všechny soubory k danému exponátu, odpojil se nebo chce poslat soubor
- ExponatNumber – je číslo položky, od které chce klient stáhnout seznam souborů
- FileId – číslo souboru, který se má poslat klientovi
- Time – aktuální čas poslání zprávy od klienta

Pokud se nestahuje soubor nebo není potřeba seznamu souborů, ale jenom se na server zasílá zpráva, která kontroluje dostupnost připojení, jsou hodnoty nepotřebných částí řetězce nastaveny na určitou implicitní hodnotu (-1).

Zaslání seznamu souborů

Pro přístup k databázi využívám prostředků ADO.NET. Ten obsahuje komponentu *DataTable*, která je využita pro přenos seznamu souborů k určitému exponátu. Nejdříve jsem se pokoušel o *serializaci*

DataTable do binární podoby, pomoci třídy *BinaryFormaterr*. Narazil jsem ovšem na problém při *deserializaci* na zařízení PDA, protože Compact .NET Framework nepodporuje třídu *BinaryFormatter*.



Obrázek 6-3: Odeslání DataTable ze serveru klientovi

V úvahu tedy přišla možnost vytvoření *DataTable* z XML souboru. To znamená, že klient pošle požadavek na soubory k nějakému exponátu a na straně serveru se do *DataTable* uloží z databáze tabulka, která je výsledkem příslušného dotazu. Výsledná *DataTable* se převede do XML, poté XML do pole bytů a odešle se zpět klientovi. Na straně klienta se pole bytů převede zpět do XML formátu a vytvoří se z něj *DataTable*, kterou dále užívám v aplikaci.

Přijímání souboru

Seznam souborů patřících k danému exponátu je zobrazen v komponentě *ListView*. U každého souboru se zobrazí název a také velikost souboru. Na této komponentě lze zachytávat událost, která je volána po kliknutí na soubor, takže pokud klient na soubor klikne, soubor se stáhne a spustí v programu, který je s daným souborem asociován. Přijímání souboru probíhá stejně jako přijímání DataTable, tzn. soubor se po částech přijímá a ukládá do objektu *MemoryStream*. Protože PDA má omezenou kapacitu, ukládá se soubor po stáhnutí určitého množství bytů do dočasného souboru. Poté co se stáhne celý soubor je dočasný soubor přejmenován. Nastane – li situace, že se soubor nestáhne celý (chyba při připojení nebo uživatel sám ukončí stahování souboru), vymaže se dočasný soubor z disku.

Uložení souborů

Ukládání souborů na straně klienta probíhá tak, že po požadavku na stáhnutí souboru zjistím cestu, odkud se aplikace spouští. V adresáři, kde bude uložen spustitelný soubor aplikace, vytvořím adresář *Temp* do kterého se budou ukládat všechny soubory, které si bude uživatel stahovat. Po ukončení

programu se uživateli zobrazí seznam souborů v adresáři *Temp*, u každého bude zaškrtávací tlačítko (checkBox). V tomto seznamu souborů budou všechna tlačítka zaškrtnuta, aplikace implicitně maže všechny stažené soubory. Ty soubory, které si uživatel chce na zařízení nechat zůstanou v adresáři *Temp*.

Lokalizace

Protože je možné, že tento typ aplikace budou využívat zahraniční uživatelé, je aplikace lokalizovaná, tj. přizpůsobena prostředí uživatele. Lokalizací uživatelského rozhraní umožníme aplikaci, aby komunikovala s uživatelem v jeho mateřském jazyce, ale nejen to. Jde o celkové nastavení aplikace tak, aby se čísla, data nebo měna zobrazovala ve formátu typickém pro kulturu, která je zvolena. Aby mohla být aplikace vícejazyčná, je potřeba vytvořit pro každý jazyk soubor zdrojů, který má společné jméno, ale liší se příponou charakterizující kulturu. Uvažujme pouze dva jazyky, češtinu a angličtinu (Britskou angličtinu). Soubory pro tyto jazyky mohou vypadat například takto

```
Messages.cs-CZ.resx
Messages.en-GB.resx
```

Soubory jsou ve formátu XML a v každém ze souborů jsou stejné klíče a k nim příslušné řetězce. Klíče v příslušných souborech vypadají pro českou verzi takto

```
<data name="FinalForm_Text" xml:space="preserve">
     <value>Odstraňování souborů</value>
</data>
```

a pro anglickou takto

```
<data name="FinalForm_Text" xml:space="preserve">
     <value>Files deleting</value>
</data>
```

Chceme – li lokalizaci použít v projektu, využijeme třídy *ResourceManager*, které při vytváření instance zadáme jako argument název souborů, využívaných pro lokalizaci. V tomto případě *Název_aplikace.Messages*. Řetězce pro daný jazyk získáme voláním funkce *GetString*, které musíme jako argument předat klíč hlášky nebo lokalizovaného zdroje a kulturu která se má pro lokalizaci využít. Pokud vytvoříme v aplikaci jenom tyto dva soubory, nebude lokalizace fungovat. Pro správnou funkčnost musíme vytvořit soubor se stejným názvem jako předchozí dva, ale bez specifikace kultury (tzn. *Messages.resx*). Toto je obecný soubor, který se použije pokud nebude nalezen soubor s příslušnou kulturou.

Při vytvoření aplikace (*build*) se v adresáři, kde je soubor spouštějící aplikaci vytvoří adresář pro každý ze souborů. Pro správný běh je nutno, aby tyto adresáře byly vždy u ve stejném adresáři jako aplikace.

6.2.2 Popis tříd

Klient neobsahuje nějaké velké množství tříd, protože ke správné funkčnosti postačili následující.

ClientForm.cs – hlavní formulář, ve kterém uživatel zadává IP adresu serveru. Pokud se mu podaří se k serveru připojit, otevře se modálně formulář *TabForm*. Pokud se uživateli nepodaří připojit, zobrazí se *DialogBox* s případnou chybovou hláškou.
AClientSocket.cs – obsahuje asynchronní operace pro připojení, posílání dat a příjem. Při volání vlastnosti *Connected* odešle klient na server prázdný řetězec, pokud se odeslání nepodaří zahodí příslušnou výjimku, která je obsloužena na vyšší úrovni.
TabForm.cs –formulář obsahující záložky, ve kterých se zobrazují příslušné soubory stejného typu. Při nedostupnosti připojení se tento formulář zavře a uživateli se opět zobrazí základní připojovací formulář.
Operations.cs – statická třída, která obsahuje metody, kdy z pole bytů vytvoří příslušný objekt (*DataTable, FileStream*).
ProgressBarForm.cs –formulář, který zobrazuje průběh stahování souboru ze serveru. Po dokončení stahování se před uzavřením formuláře uloží soubor na zařízení. Při předčasném zavření formuláře se ukončí stahování souboru.
FinalForm.cs –formulář zobrazující se při zavírání aplikace. Obsahuje *ListView*, kde jako položky jsou soubory, které si uživatel stáhl za běhu aplikace. Nabídne jejich smazání nebo ponechání v adresáři *TEMP*.

6.3 Serverová aplikace

Serverová část aplikace se skládá ze dvou částí, první je obsluha klienta a druhá část je administrační, pro práci s daty v databázi. Při návrhu serveru, jsem se rozhodoval, jestli udělat nějakou autentizaci (ověřování), jak pro přihlášení klienta, tak ověření přístupu při práci s databází. Myslím, že registrace klientů a jejich následné přihlašování by bylo zbytečné a nepraktické.

6.3.1 Úvod

Aplikace je psána v jazyce C# (*sharp*) a využívá prostředky prostředí .NET. Na rozdíl od klienta jsou operace pro komunikaci synchronní. Je to z toho důvodu, že aplikace je více vláknová a obsluha každého klienta probíhá v jiném vlákně, takže není třeba aby server asynchronně zpracovával požadavky od klientů. Situace je popsána na obrázku (viz. Obrázek 6-5), při startu serveru se v hlavním vlákně spustí automaticky jedno vlákno, které se stará o poslouchání, zda se chce nějaký

klient připojit. Pokud se tak stane a klient se připojí, tak se v tomto vlákně spustí další, které už se stará o samotného klienta.

Delegáty

Protože v další části budu používat termín delegáty, bylo by vhodné nastínit, co to delegát vůbec je. Jedná se o bezpečné ukazatele na funkce, které jsou ale na rozdíl od ukazatelů v C++ objektově orientované a typově bezpečné. V .NET se delegáty používají hojně, ať už jde o zachytávání událostí, kdy delegát je ukazatel na obsluhu události nebo o asynchronní metody, kdy se delegát používá právě jako ukazatel na asynchronní metodu.

Problémy při práci s vlákny

Při práci s vlákny je třeba dodržovat určitá pravidla, aby nedošlo k nestandardnímu chování aplikace. Je třeba dát si pozor na tzv. uváznutí (*deadlock*), ten nastane například pokud jedno vlákno čeká na probuzení jiným vláknem, které je už ukončeno nebo také uspáno. Tento problém mě při práci příliš netížil, protože v aplikaci nenastane situace, kdy by jedno vlákno čekalo na jiné.

Další úskalí při psaní aplikace pomoci vláken je přístup ke sdíleným proměnným. Pokud více vláken přistupuje ke stejnému zdroji (stejné proměnné), například jedno vlákno zapisuje hodnotu a v zápětí by mělo jiné vlákno hodnotu číst, je potřeba zajistit, aby vše bylo synchronizované. To znamená, aby se hodnota nečetla dříve než se zapsala. V prostředí .NET se toto řeší uzamčením části kódu tak, aby s ním ve správném okamžiku mohlo pracovat pouze jedno vlákno.

```
//argument object je objekt, který má být uzamknut
lock (object)
{
    //kód, který se má provádět nebo zpracovávat
}
```

Toto používám při přidávání klientů a pro následné načítání do respektive z statického pole klientů.

Problémem by také mohlo být přistupování ke komponentám v jiném vlákně, než ve kterém byly vytvořeny. Pro příklad, pokud si vytvoříme *ListView* (komponenta pro zobrazení seznamu položek) v jednom vlákně a chceme v jiném vlákně do komponenty vložit nějakou položku, musíme tak provést přes delegáta. Takže by to mohlo vypadat takto

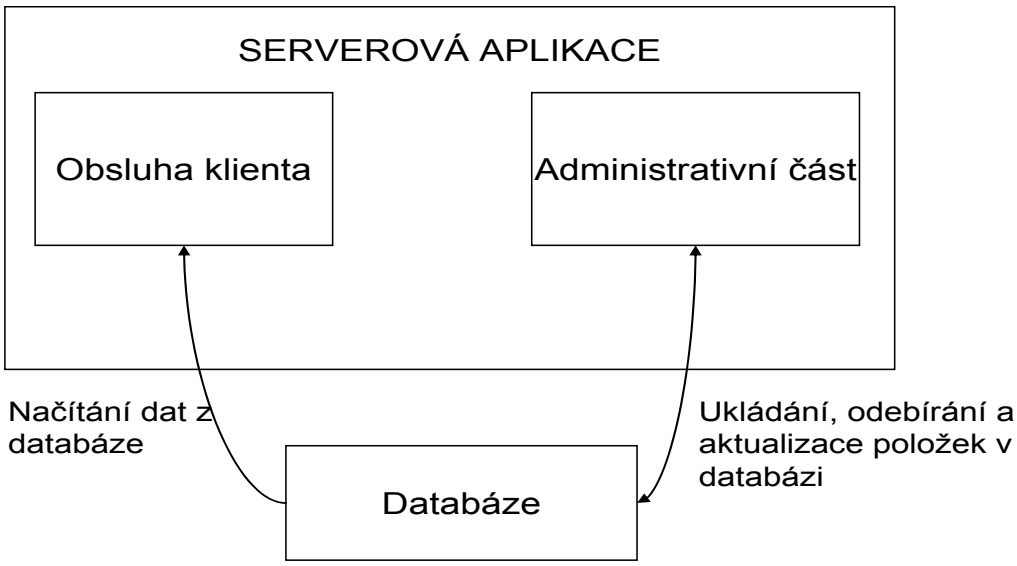
```
public void methodName(array_of_arguments)
{
   //pokud je ke komponentě přistupováno z jiného vlákna
```

```
    if (listView.InvokeRequired)
    {
        //vytvoříme si delegáta s určitým jménem a řekneme mu kterou funkci má volat
        delegateName dn = new delegateName(methodName);
        //zavoláme delegáta pro určitou komponentu a určíme co budou argumenty funkce
        listView.Invoke(dn, new object[] {array_of_arguments};
    }
    else
    {
        //provede se kód, jako by se ke komponentě přistupovalo z vlákna ve kterém byla vytvořena
    }
}
```

Prostředí .NET se stará o to, aby aplikace byla bezpečná (*thread-safe*), takže pokud se objeví „možnost“ takovéhoto přístupu, je na to programátor upozorněn.

6.3.2 Popis aplikace

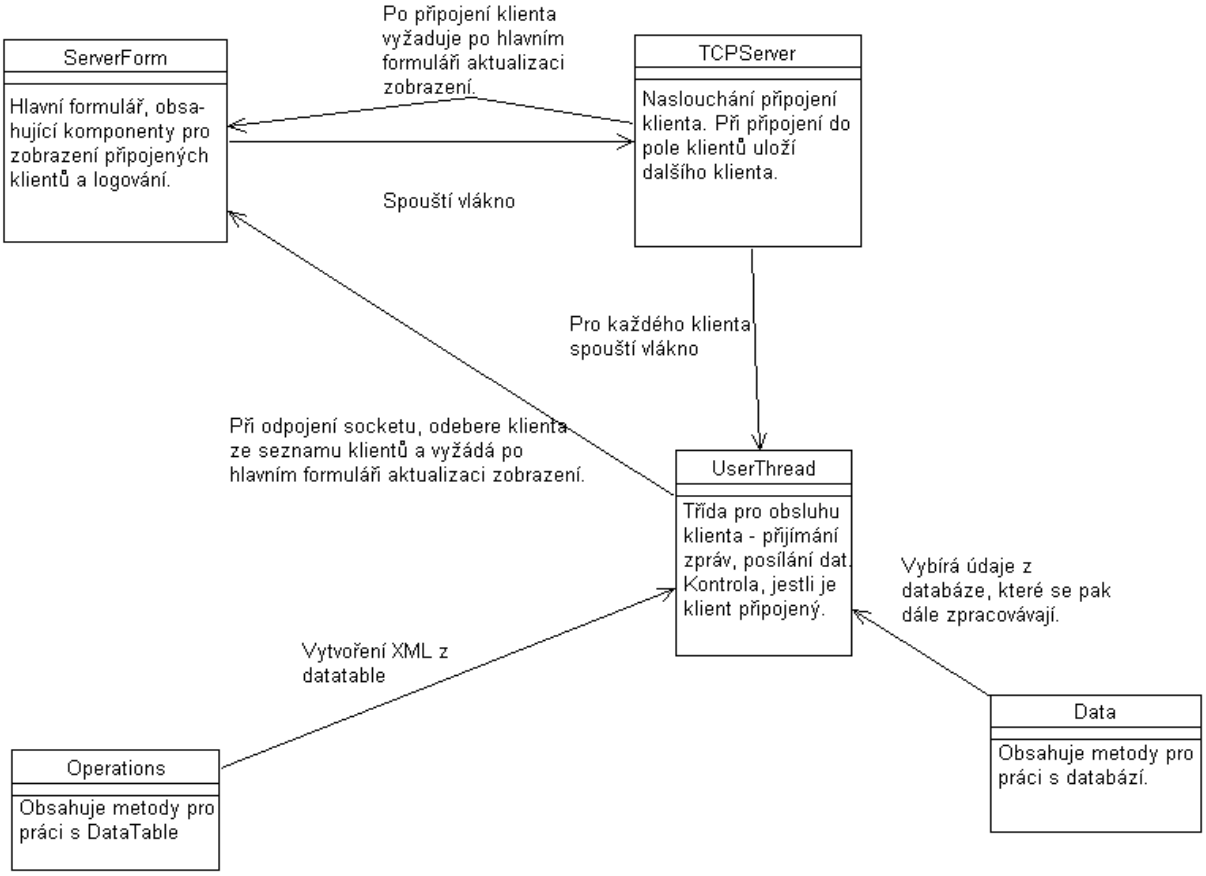
Základem aplikace je jako v případě klienta *WinForm*, hlavní formulář, ve kterém je v horní části zobrazen seznam připojených klientů a v dolní části probíhá informování o tom, co se na serveru děje. Přes menu se může uživatel dostat do administrační části a upravovat data v databázi. Z jakých částí se server skládá a co která část dělá ilustruje obrázek.



Obrázek 6-4: Spolupráce mezi administrativní částí a obsluhou klienta

Obsluha klienta

Jak jsem již dříve psal, obsluha každého klienta probíhá ve vlastním vlákně. Jakmile se klient připojí k serveru, je mu spuštěno vlákno. Je také přidán do seznamu připojených klientů a je aktualizován seznam připojených klientů na úvodní obrazovce. Jak asi spolupracují jednotlivé třídy je ukázáno na obrázku



Obrázek 6-5: Obsluha klienta

Původně se vyskytl problém, že se klienti neaktualizovali tak, jak měli, tzn. když se klient připojil, objevil se sice v seznamu, ale pouze jako prázdný řádek. To mohlo být způsobeno tím, že vlákno ve kterém se vkládal klient do seznamu, vložilo údaj později než proběhla aktualizace zobrazení. Problém byl vyřešen uzamknutím příslušného kódu a uspáním vlákna na krátký časový okamžik (10ms).

I v této části jsem chtěl původně pro zjištění, jestli je klient připojen, vytvořit časovač, který by se pokoušel odeslat prázdný řetězec klientovi a zachytával výjimky, ale nebylo to nutné, protože na rozdíl od klienta není problém sledovat v serverové části klientský *socket*. V klientském vlákně běží smyčka, obsluhující potřeby klienta, dokud je *socket* připojen (*socket.Connected*). V této smyčce přijímám data od klienta v určitém formátu (viz.**Zasílání zpráv**) a podle *MessageId* zjišťuji, co klient chce.

Pokud chce klient poslat seznam souborů postupuje se viz. **Zaslání seznamu souborů**, pro převod *DataTable* je použita možnost zápisu *DataTable* do XML. Při poslání souboru se postupuje trochu jinak, nepoužívá se metoda *socket.send*, jako u posílání ostatních dat, ale nová metoda *socket.sendFile*, která má jako argument cestu k souboru. Při požadavku na soubor, si ze zprávy, kterou klient zašle na server zjistím číslo souboru, který chce klient poslat a z databáze dostanu cestu, kde je soubor uložen. Potom už není problém soubor klientovi odeslat.

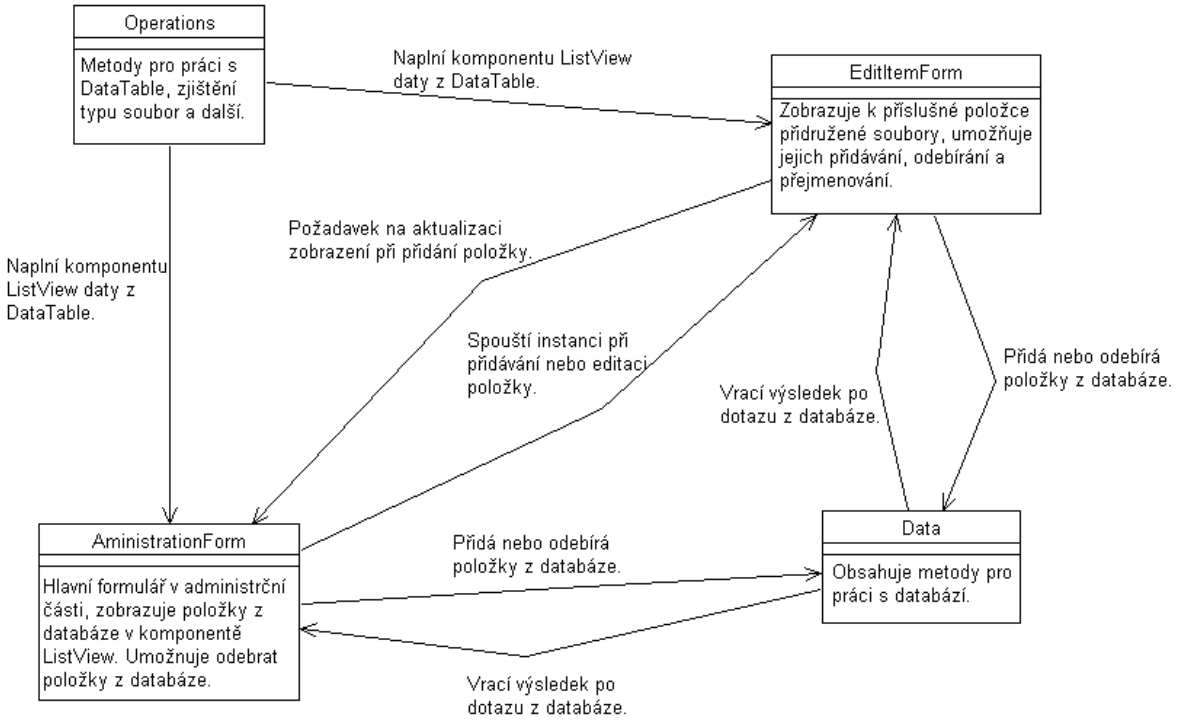
Pokud se klient odpojí ze serveru, je nejdříve odstraněn ze seznamu klientů a posléze je zastaveno vlákno, ve kterém běžel.

Administrační část

Administrační část obsahuje také hlavní formulář, ve kterém se zobrazují všechny položky z databáze. Jako komponenta pro zobrazování se využívá *ListView*, kdy reaguji na stisk pravého tlačítka nad jednotlivými položkami. Uživatel si tedy může vybrat co chce s danou položkou dělat. Může ji:

- Editovat – spustí se další formulář se soubory patřícími k dané položce
- Odebrat – odstraní položku z databáze a také všechny odpovídající záznamy v tabulce souborů
- Přejmenovat – správce může přejmenovat položku pomoci zobrazeného *TextBoxu*.

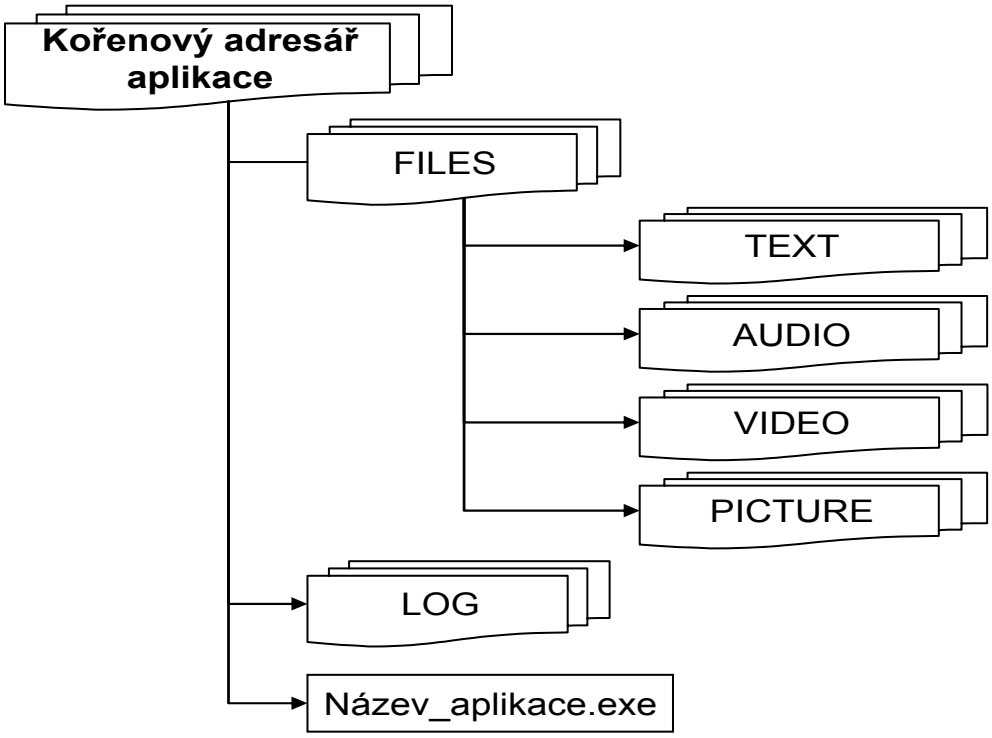
Jak funguje administrační část ilustruje následující obrázek



Obrázek 6-6: Administrativní část

Dovoluje správci serveru editovat, přidávat a mazat položky z databáze. I když ADO .NET umožňuje odpojený model, tzn. uložím si celou databázi do objektu *DataSet* a potom s ní pracuji, jako s normální databází, využil jsem spíše standardního přístupu. To hlavně z důvodu, že nehrozí přehlcení databáze dotazy, protože tabulky v databázi nejsou obsáhle a obsahují pouze textové hodnoty. I při dotazování velkého množství klientů, by neměl být problém tyto dotazy zpracovávat v přijatelném čase.

Pro seznam položek v databázi využívám komponentu *ListView* (dále seznam), stejně jako pro všechny seznamy. Po otevření formuláře se provede dotaz do databáze, který uloží příslušnou tabulku do *DataTable* a z této tabulky se záznamy vloží do seznamu. Ukládání přímo do seznamu jsem nepovažoval za příliš povedené, protože metodu, která vrací dotaz v podobě *DataTable* využívám prakticky v celé aplikaci. Uživatel může přidávat jak položky, tak ke každé položce soubory. Při přidávání položek se vkládají data pouze do tabulky v databázi, kdežto u vkládání souborů se ukládají data jak do tabulky, tak na disk. Pro vkládání souborů je na událost stisku tlačítka otevřen dialog pro výběr souboru, správce vybere soubor někde na disku (vyměnitelné media, diskety) a přidá soubor k položce. Aby bylo dodrženo nějaké rozumné uspořádání souborů, zjišťuji si při vkládání souborů, ze kterého adresáře byla aplikace spuštěna. Struktura adresářů bude nejlépe patrna z obrázku



Obrázek 6-7: Struktura adresářů

Pokud při ukládání neexistuje adresář *FILES*, vytvoří se stejně jako ostatní adresáře uvnitř. Soubory jsou ukládány se stejným jménem, jaké mají databázi. Jestliže soubor v příslušném adresáři neexistuje, zkopíruje se z aktuálního místa do příslušného adresáře podle svého typu. Existuje – li

více položek, které mají společný soubor a správce odstraní soubor pouze u jedné položky, pak se nesmaže soubor z disku. Pokud je záznam v tabulce jediný, maže se soubor i z disku.

Pro snadnější odstraňování položek z databáze, je v seznamu zobrazeno u každého řádku zaškrtávací tlačítko (*checkbox*), takže se může odstraňovat více položek najednou.

Protokolování

Protokolování je záznam událostí, které se na serveru dějí nebo udály. Tyto události se ukládají do souboru, který je podobně jako soubory uložené v databázi uložen v adresáři *LOG* (viz. Obrázek 6-7), který je umístěn v kořenovém adresáři aplikace. Formát souboru je

```
Datum.log (např. 19.6.2007.log)
```

Aby nebyl soubor po nějaké době příliš velký, vždy při zápisu zprávy do souboru se najde soubor, který byl vytvořen nejpozději. Jestli bude tento poslední soubor starší než měsíc, vytvoří se nový soubor s aktuálním datem. Zaznamenávat se budou události:

- Připojení klienta
- Odpojení klienta
- Požadavek na zaslání seznamu souborů
- Požadavek na zaslání souboru
- Odeslání seznamu souborů klientovi
- Odeslání souboru klientovi
- Výjimky

I když se ve zprávách od klienta posílá i datum, je pouze orientační, protože protokolování probíhá na straně serveru, proto se bude ukládat aktuální datum, kdy server požadavek vyřídí. Hlavním důvodem protokolování je zaznamenávání výjimek, které by při případném pádu aplikace mohli pomoci při vyřešení problému. Všechny výjimky, které zachytávám mají stejnou textovou podobu

```
Aktuální_datum název_funkce - název_třídy - typ_výjimky: text_výjimky
```

Proto nebude problém zjistit, ve které funkci se stala chyba a případné nesrovnalosti rychle vyřešit.

Určité zobrazování zpráv bude probíhá i v samotné aplikaci, aby měl případný správce přehled, jestli aplikace funguje správně a co se zrovna děje.

6.3.3 Popis tříd

ServerForm.cs – hlavní formulář, ve kterém si spustíme vlákno pro poslech klientů. Tento formulář obsahuje komponenty pro zobrazení připojených klientů a pro zobrazení zpráv. Obsahuje také metody pro aktualizaci zobrazení klientů, které se volají z jiného vlákna.
TCPServer.cs – třída pro poslech klientů. Při připojení klienta nebo odpojení klienta aktualizuje obsah seznamu připojených klientů.
UserThread.cs – spouští vlákno pro každého klienta. Obsahuje metody pro posílání a přijímání dat a také smyčku zpráv, ve které zjišťuje co klient požaduje.
AdministrationForm.cs – hlavní formulář pro zobrazení existujících položek v databázi a jejich odebírání nebo editaci.
EditItemForm.cs – formulář, který se zobrazuje, pokud chceme u nějaké položky v databázi aktualizovat příslušné soubory, tzn. přidávat, odebírat nebo přejmenovávat.
Data.cs – obsahuje veškeré metody pro práci s databází. Všechny metody jsou statické, proto není třeba vytvářet instanci této třídy.
Operations.cs – podobně jako u klienta obsahuje operace pro ulehčení a zpřehlednění práce, jako například převedení *DataTable* do XML a následně XML do pole bytů. Nebo naplnění *ListView* komponenty daty z *DataTable*.
Log.cs – zaznamenávání informací do předem vytvořeného souboru. Pokud je soubor starší než 30 dnů, vytvoří v předem daném adresáři nový soubor se jménem odpovídajícím aktuálnímu datu.

6.4 Databáze

Jako databázový server využívám MS SQL 2005. Vytvořil jsem si testovací databázi, která obsahuje všechny potřebné tabulky. Původní návrh předpokládal ukládání informací o tom co se na serveru děje (protokolování) do tabulky v databázi. Jenomže toto řešení je docela nepraktické, protože by se musela tabulka promazávat, aby nebyla příliš velká. Proto je protokolování řešeno jiným způsobem (viz. **Protokolování**). Velké problémy při připojení nebyly, MS SQL je přece jen server, který je součástí instalace Visual Studia 2005. Při připojování si musíme dát pozor na to, abychom se připojovali na správný název serveru. Při vytvoření serveru jsem měl tendenci se připojit na server se jménem *jméno_serveru*, přičemž správné jméno je *jméno_serveru\sqlexpress*. Na MS SQL server se přistupuje ze serverové aplikace, *autentizace* připojení k databázi je nastavena na *windows authentication*, proto se k databázi připojí jakýkoliv uživatel operačního systému, pokud má plná práva.

6.5 Problémy při řešení

Některé problémy , které se vyskytly při řešení jsem popsal u příslušných částí, některé další se budu snažit zmínit v této části.

Klient

Problémem, u kterého jsem se nejvíce zastavil při tvorbě klientské aplikace, bylo nalezení asociovaných ikon s daným programem. Chtěl jsem si seznam souborů v určité záložce zobrazit jako ikony, pod kterýma by byl příslušný název souboru. Pokud by tedy uživatel stáhl soubory, které se mají spustit v programu pocket Word, měla by se u souboru zobrazit ikona, která je s tímto programem asociována. Problém nebyl při nalezení programu, který je s daným souborem asociován, to jsem měl už vyřešeno pomoci procházení registrů na pocket PC. Ikony pro příslušný program jsou uloženy v souboru typu *dll*. Pro každý program je v tomto souboru uloženo několik ikon. V registrech je klíč uložen jako *Název_souboru, číslo* , kde číslo symbolizuje číslo ikony. Ale nepodařilo se mi danou ikonu ze souboru extrahovat, v knihovnách Win32 na to existují určité metody (*SHGetFileInfo, ExtractIconEx*), které podle názvu souboru zjistí asociovanou ikonu. Zde se ovšem vyskytl problém, že tyto funkce jdou použít v .NET Frammewok, ale v Compact .NET Frammewok nejsou uloženy ve stejných knihovnách nebo neexistují vůbec. Po několika dnech jsem pokus o vyřešení tohoto problému vzdal a nakonec nezobrazuji ikony, ale názvy souborů. Na funkčnosti aplikace to nic nemění.

Další záležitostí, kterou jsem musel vyřešit, je spouštění souboru v příslušném programu. Po zkušenostech s ikonami jsem myslel, že by to mohlo být hodně složité, ale nakonec tomu tak nebylo.

```
//vytvoříme si nový proces
ProcessStartInfo psi = new ProcessStartInfo();
psi.UseShellExecute = true;
//nastavíme jméno souboru, který chceme spustit v asociovaném programu
psi.FileName = "nazev_souboru.pripona";
//spustíme proces
System.Diagnostics.Process.Start(psi);
```

Daný proces spustí soubor v programu, který je s ním asociován.

Jednou z posledních věcí bylo dilema, jestli při ukončení klientské aplikace smazat všechny soubory, které si klient za dobu běhu aplikace stáhl nebo mu je nechat. Rozhodl jsem se o kompromis, jelikož se soubory ukládají u klienta do adresáře, ve kterém se spouští aplikace, není problém aby si uživatel tyto soubory našel. Ukládání probíhá v relativně normálním formátu, tzn.

ukládá se název souboru a ne číslo, jak jsem měl původně v plánu. Při stáhnutí souboru se uloží do *DataTable* název souboru a jeho cesta a při ukončení aplikace má uživateli možnost vybrat, které soubory chce na PDA nechat a které ne.

Serverová aplikace

Tady bych zmínil spíše jen nedostatek Compact .NET Frammewok. Původně jsem chtěl seznamy souborů zakódované do *DataTable serializovat* do pole bytů. Na to je v .NET Framewok třída *BinaryFormatter*, která ale v Compact .NET Framewok chybí, proto se musel převod *DataTable* řešit jiným způsobem (viz. **Zaslání seznamu souborů**).

Při zobrazování položek v administrativní části by se mohlo dost dobře stát, že by počet položek které se mají zobrazit v komponentě *ListView,* byl příliš velký. Protože jsem nenašel možnost specifikovat, kolik položek se má maximálně zobrazovat, tzn. neexistuje možnost záložek v *ListView*, snažil jsem se tento problém vyřešit jinak Zjistím si počet položek, které se budou zobrazovat, pokud jich bude více než sto, budu jich načítat pouze sto a zobrazovat v *ListView*. Podle počtu položek se zobrazí pod komponentou čísla (komponenta *label*), která budou reagovat na klik myši. Těchto čísel bude podle počtu položek, maximálně však 10. Jestli bude načteno více než 1000 položek, bude se v jedné záložce zobrazovat *počet_položek / 10* záznamů.

Databáze

Databázi jsem si vytvořil standardním způsobem, tzn. vytvořil jsem si tabulku, určil co bude primární klíč a jak se bude zvyšovat (většinou +1) a po vytvoření všech tabulek jsem mezi nimi definoval vztahy. Nejvíce mě asi zajímal vztah mezi tabulkou *tabItem* a *tabFiles*, který byl definován, jak je uvedeno na obrázku

Obrázek 6-8: Vztah mezi *tabFile* a *tabItem* (obrázek z Visual Studia)

Pokud definujeme vztahy v MS SQL, můžeme tak učinit přímo ve Visual Studiu. Vytvořil jsem si vztah mezi těmito tabulkami a nastavil *delete On Cascade*, tzn. pokud se vymaže záznam z jedné tabulky s primárním klíčem *primary_key*, smažou se všechny záznamy z druhé tabulky, kde je *foreign_key* stejný jako *primary_key* první tabulky. Tento způsob jsem potřeboval požít, abych při

vymazání položky smazal všechny záznamy v tabulce souborů, které se k dané položce vztahovaly. Řešením pak bylo vyhledat všechny záznamy v tabulce souborů, které se k dané položce vztahují a smazat je postupně. Původně se mi to zdálo nepraktické, ale protože může jeden soubor patřit více položkám, poskytuje mi to alespoň místo pro kontrolu, jestli soubor, který chci odstranit z databáze, je rovněž posledním záznamem daného souboru a má se odstranit i z disku.

6.6 Testování

Testování mělo tři fáze, první byla odladit a odzkoušet aplikaci na stolním počítači. Nabízela se možnost testovat aplikaci v emulátoru, který obsahuje Visual Studio. Chování emulovaného PDA by mělo být stejné jako při připojení standardního přístroje, jenomže tomu tak nebylo. Hlavním problémem při spuštění aplikace v emulátoru je její rychlost. I když byla celá aplikace tvořena v Compact Frameworku, po sestavení je možno ji spustit i na stolním počítači. Výhodou je, že při spuštění se také otevře konzole, ve které se zobrazují hlášky posílané z programu na standardní výstup.

Poté co byla správnost aplikace ověřena na stolním počítači, začalo testování v emulátoru. Protože aplikace na stolním počítači pracovala správně bylo zarážející, že při odzkoušení na emulátoru se objevily určité problémy. Hlavní byly při volání aktualizace komponenty typu *control* v jiném vlákně. Samozřejmě toto je problém, protože se ke komponentám z jiných vláken nedá přistupovat bez volání metody *Invoke* pro danou komponentu (viz. Problémy při práci s vlákny), ale je s podivem, že při spuštění aplikace na stolním počítači tento problém nenastane.

Po opravení jsem měl možnosti odzkoušet aplikaci na skutečném PDA, ale bez možnosti připojení přes Wireless LAN. PDA bylo připojeno k počítači přes USB a simulaci připojení zajišťoval program ActiveSync. Ani toto testování neproběhlo úplně v pořádku, protože při volání metody *socketu* pro připojení k serveru, se klient připojil i k serveru, který by ani existovat nemohl (např. IP adresa 1.0.0.0). Proto při testování připojení musí klientovi nazpět dorazit zpráva o úspěšnosti připojení.

Poslední fází testování bylo ověření funkčnosti na přístroji vybaveném WI-FI kartou pro připojení k bezdrátové síti. Pro navázání připojení jsem si na stolním počítači nainstaloval WI-FI kartu, tento počítač sloužil jako server. Serveru i PDA zařízení jsem nastavil pevné IP adresy, protože jinak připojení nefungovalo. Při připojení přes bezdrátovou síť se zařízení chovalo stejně jako při propojení přes USB. Jediný rozdíl byl v tom, že na přístroji s WI-FI kartou byly WinCE 5.0 a druhý byl typu Pocket PC. Aplikace proto vypadala na každém zařízení úplně jinak, proto jsem udělal dvě klientské aplikace, jednu pro Pocket PC a druhou pro zařízení s WinCE 5.0.

Protože jsem neměl k dispozici více přístrojů PDA, simuloval jsem připojení několika klientů, připojením jednoho z PDA a několikerým připojením ze stolního počítače počítače.

7 Závěr

Podle zadání diplomového projektu jsem měl prostudovat tvorbu klient – server aplikací, možnosti připojení z malého mobilního zařízení k databázovým systémům a v poslední části implementaci klient – server aplikace pro malá muzea. Zadání bylo splněno v celém rozsahu.

V této práci jsem implementoval dvě aplikace. První aplikací je server, což je *WinForm* aplikace, která se dělí na dvě části. Jedna z části se stará o obsluhu klienta, vyřizování jeho požadavků, připojování. Druhá, administrační část, umožňuje práci s předem známou databází. Celá serverová aplikace pracuje s více vláknovým modelem. Takže před samotnou implementací bylo potřeba si navrhnout tento model tak, aby server příliš nezatěžoval počítač na kterém běží, ale také aby vyřizování požadavků probíhalo v krátkém čase. Toto se myslím povedlo dostatečně, protože celý projekt má být implementován pro malé muzea, takže jsem nemusel předpokládat obrovský počet simultánně připojených klientů. Pro každého z klientů běží vlastní vlákno, proto žádný z připojených klientů nebrzdí v práci jiného. Jelikož všechny data v podobě souborů, která se posílají klientům, jsou uložena v adresáři na lokálním disku a v databázi jsou pouze reference na tato data, není dotazování uživatelů ve stejném okamžiku problémem. Serverová část aplikace také obsahuje docela přehledné protokolování do souboru, takže při případných problémech lze z tohoto souboru zjistit, kde chyba nastala a rychle ji opravit.

Klientská aplikace je psána pro kapesní počítač, s využitím Compact .NET Framework. Klient je synchronní s asynchronními operacemi pro prácí se *socketem*. Pokud uvažujeme reálné užití aplikace v muzeu, musí být aplikace lokalizovaná. To hlavně z toho důvodu, že tato zařízení navštěvují i cizinci. Proto je klientská aplikace ve dvou jazycích v češtině a v angličtině. Uživatel má možnost si vybrat jazyk ještě před připojením serveru, ale také v průběhu práce aplikace. Klientská aplikace se k serveru připojuje po zadání IP adresy serveru. Klient si stáhne k položce, jejíž číslo zadá do formuláře, seznam souborů. Pokud si chce stáhnout nějaký soubor, zobrazí se formulář s průběhem stahování. Při předčasném ukončení formuláře uživatelem se soubor přestane ze serveru stahovat a jeho část uložená na disku PDA se odstraní. Po úspěšném stáhnutí se soubor spustí v programu, který je s tímto typem asociován, pokud nějaký existuje.

Rozšířením projektu by mohlo být využití čteček čárových kódů. Číslo exponátu by se nezadávalo do formuláře ručně, ale po přečtení čárového kódu exponátu by se stáhl automaticky seznam souborů. Při takové implementaci klienta by bylo potřeba udělat na serveru patřičné změny. Ty by spočívaly v možnosti tisknutí takovýchto štítků ke každému exponátu. Toto rozšíření by nemělo být až tak náročné, protože ke čtečkám jsou dodávány knihovny od výrobce, které obstarávají příslušnou logiku. Problémem je, že každý výrobce má vlastní knihovny, proto každá ze čteček funguje trochu jinak a programátor musí s tímto počítat. Potom se kód přečtený čtečkou v aplikaci zpracovává jako standardní řetězec.

Literatura

[1] Lacko, L.: Vývoj aplikácií pre mobilné zariadenia, 2003

[2] Šejda, J.: J2EE, .NET a vývoj rozsáhlých systémů 2, 11.února 2003. Dokument dostupný na URL http://interval.cz/clanky/j2ee-net-a-vyvoj-rozsahlych-systemu-2 (květen 2006).

[3] Najman, M.: Architektura .NET Framework, 8.březen 2002. Dokument dostupný na URL http://interval.cz/clanky/architektura-net-Frammewok/ (prosinec 2006).

[4] Janecek, A.: Programming Interactive Real-Time Games over WLAN for Pocket PCs with J2ME and .NET CF, 4.října 2005.

[5] Caha, V.: První dojmy z Microsoft Visual Studio 2005, 6.prosince 2006. Dokument dostupný na URL http://connect.zive.cz/?q=node/164 (prosinec 2006).

[6] Hájek, P.: Proč PDA?, 2.ledna 2004. Dokument dostupný na URL http://www.svetpda.cz/svetpda/svetpda.nsf/v/5EC8BC808CA14E39C1256AF600528082 (prosinec 2006).

[7] Papaj, K.: Databáze pod Windows V. – SQL Server 2005 a Web Services, 24 listopad 2006. Dokument dostupný na URL http://connect.zive.cz/?q=node/485 (prosinec 2006).

[8] Jakel, M.: Jak na MS SQL – zajímavé možnosti XML, 6.srpna 2002. Dokument dostupný na URL http://interval.cz/clanky/jak-na-ms-sql-zajimave-moznosti-xml/ (prosinec 2006)

[9] Plecháč, V.: ADO.NET kurz. Dokument dostupný na URL http://msdn.microsoft.cz/dotnetKurzy/adonet/LRNViewer.htm (prosinec 2006)

[10] Kupka, J.: ADO.NET – Úvod do databází v prostředí .NET, 2.května 2005.

[11] Kocan, M.: Databázové systémy neustrnuly, kráčejí směrem ke gridu, 24.listopadu 2003. Dokument dostupný na URL http://www.computerworld.cz/cw.nsf/id/B667F2BA515CCF3DC1256E9400332CF7?OpenDocument&cast=4 (leden 2007).

[12] Skřivánek, F.: Klient-server je když …, 22.března 2004. Dokument dostupný na URL http://www.dbsvet.cz/view.php?cisloclanku=2004032201 (leden 20007).

[13] Whatis.com: Rich client, 3.leden 2006. Dokument dostupný na URL http://searchsmb.techtarget.com/sDefinition/0,,sid44_gci1155694,00.html (leden 2007).

[14] www.symbio.cz: Co Vám přináší webové služby?, 4.červenec 2006. Dokument dostupný na URL http://www.symbio.cz/clanky/co-vam-prinasi-webove-sluzby.html (prosinec 2006).

[15] Taraczy – Hornoch, P., Salina, S., Carroll, A.: Development of a Personal Digital Assistant (PDA) Based Client/Server NICU Patient Data and Charting Systém, 2001. Dokument dostupný na URL http://faculty.washington.edu/pth/NICU-PDA.PDF (květen 2007).

[16] Yamabana, K., Osada, S., Okumura, A.: A Speech Translation System with Mobile Wireless Clients, 2002. Dokument dostupný na URL http://acl.ldc.upenn.edu/P/P03/P03-2023.pdf (květen 2007).

[17] Munk, M. RNDr.: IKT Vo vysokoškolských kurzoch štatistiky, 31.5.2006, s.283-286. Dokument dostupný na URL http://uninfos.ukf.sk/documents/zbornik_uninfos2006.pdf (květen 2007).

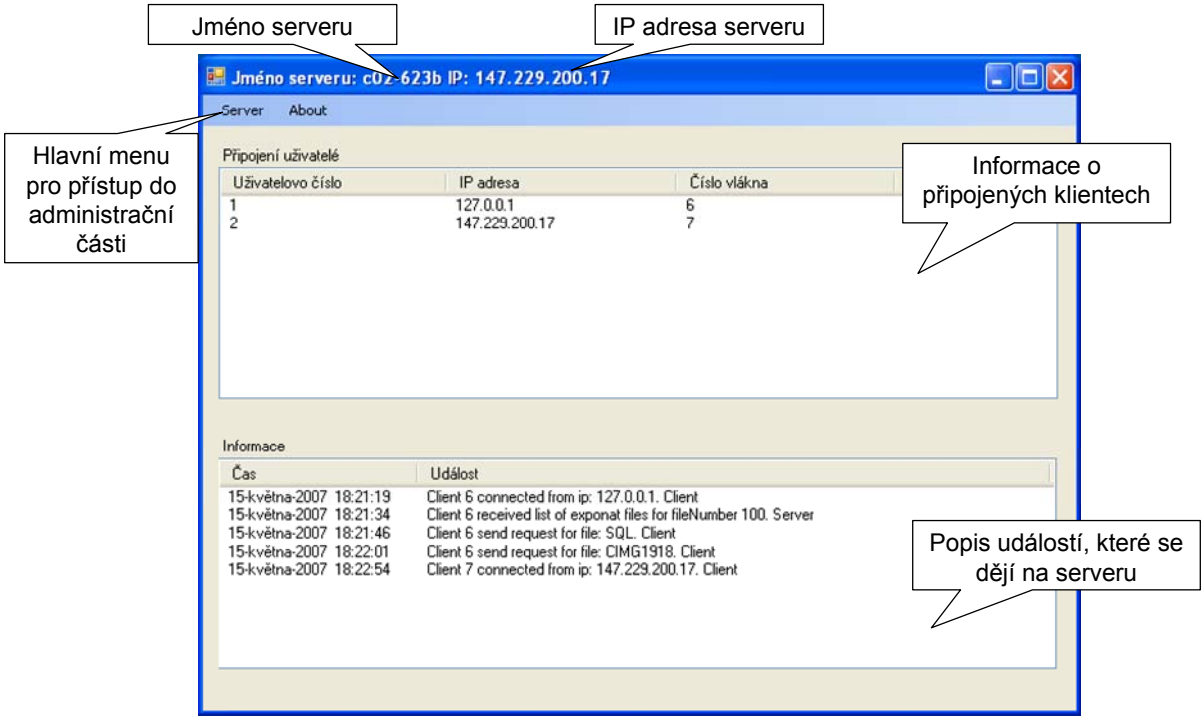
[18] Kouba, T. : Porovnání rychlosti Javy na MS Windows a Linuxu a srovnání s C# a C, 17.9.2003. Dokument dostupný na URL http://www.neo.cz/~tomas/java.net/2003-10.html#170000 (květen 2007).

[19] Hurt, O.: Podpora distribuovaných aplikací na platformě .NET, 15.12.2003. Dokument dostupný na URL http://nb.vse.cz/~zelenyj/it380/eseje/xhuro01/distnet.htm#obsah (květen 2007).

Seznam příloh

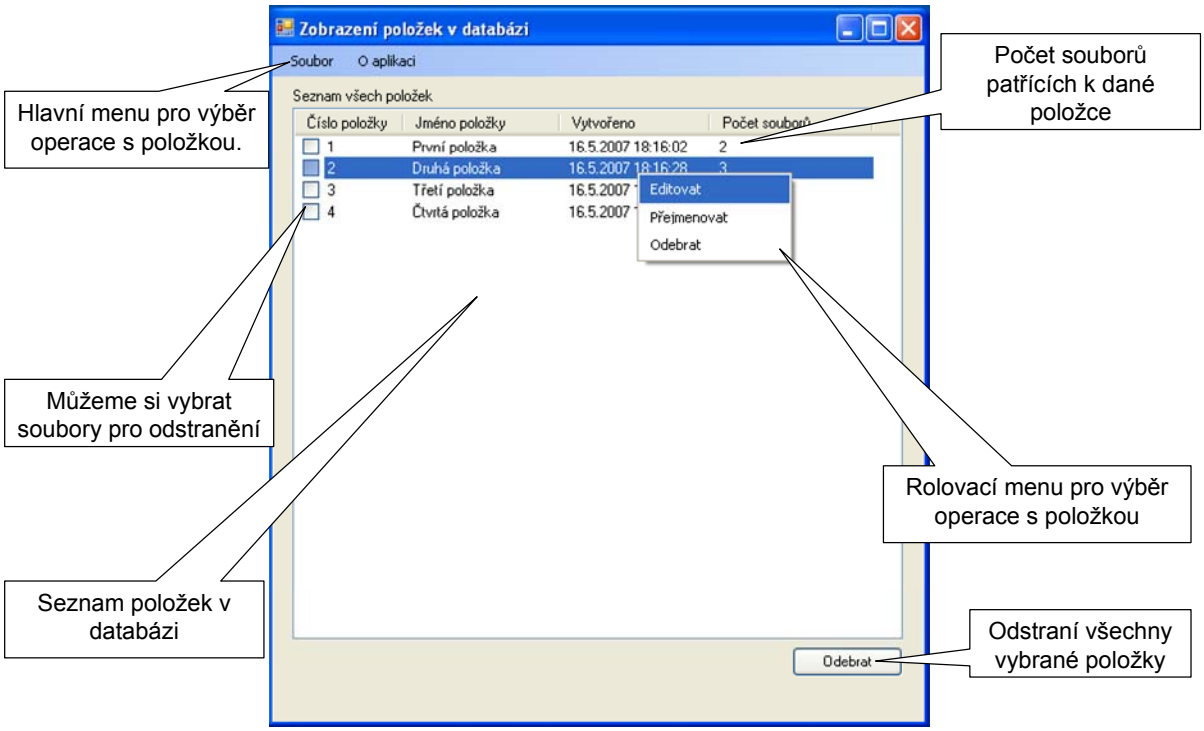
Příloha 1. Manuál k serveru

Hlavní formulář aplikace, ve kterém se v horní části zobrazují připojení klienti. Spodní část je vyhrazena pro zobrazování událostí, které se na serveru odehrávají. Přes položku v menu *Server* je možno se dostat do administrační části.



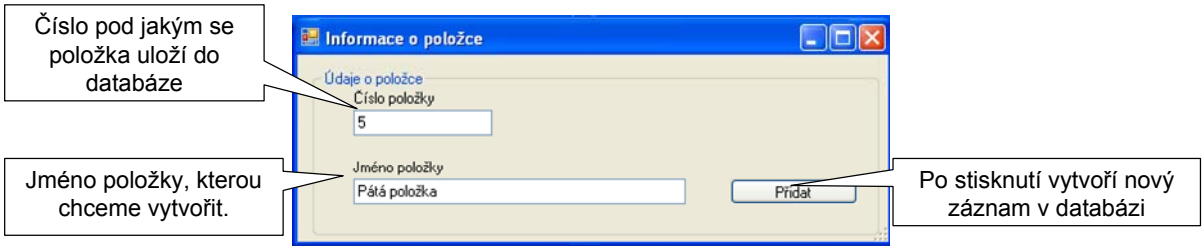
Obrázek 1: Úvodní formulář

Pokud se chceme dostat do administrační části, klikneme v menu *Server* na položku administrační část. Otevře se okno, ve které jsou zobrazeny všechny položky v databázi. Aplikace nabízí jejich editaci, přejmenování nebo odstranění. Pro výběr jedné z těchto možností můžeme použít buď standardní menu nebo rolovací menu, které se zobrazí po kliknutí pravým tlačítkem myši na některou z položek. Chceme – li odebrat více položek najednou, označíme si položky, které chceme odebrat a po stisknutí tlačítka odebrat se odstraní všechny vybrané položky i soubory k nim patřící. Pokud některý ze souborů patří ještě k jiné, existující položce, bude odstraněn záznam pouze v databázi. V případě, že soubor k žádné položce nepatří, bude odstraněn i z disku.



Obrázek 2: Administrační formulář

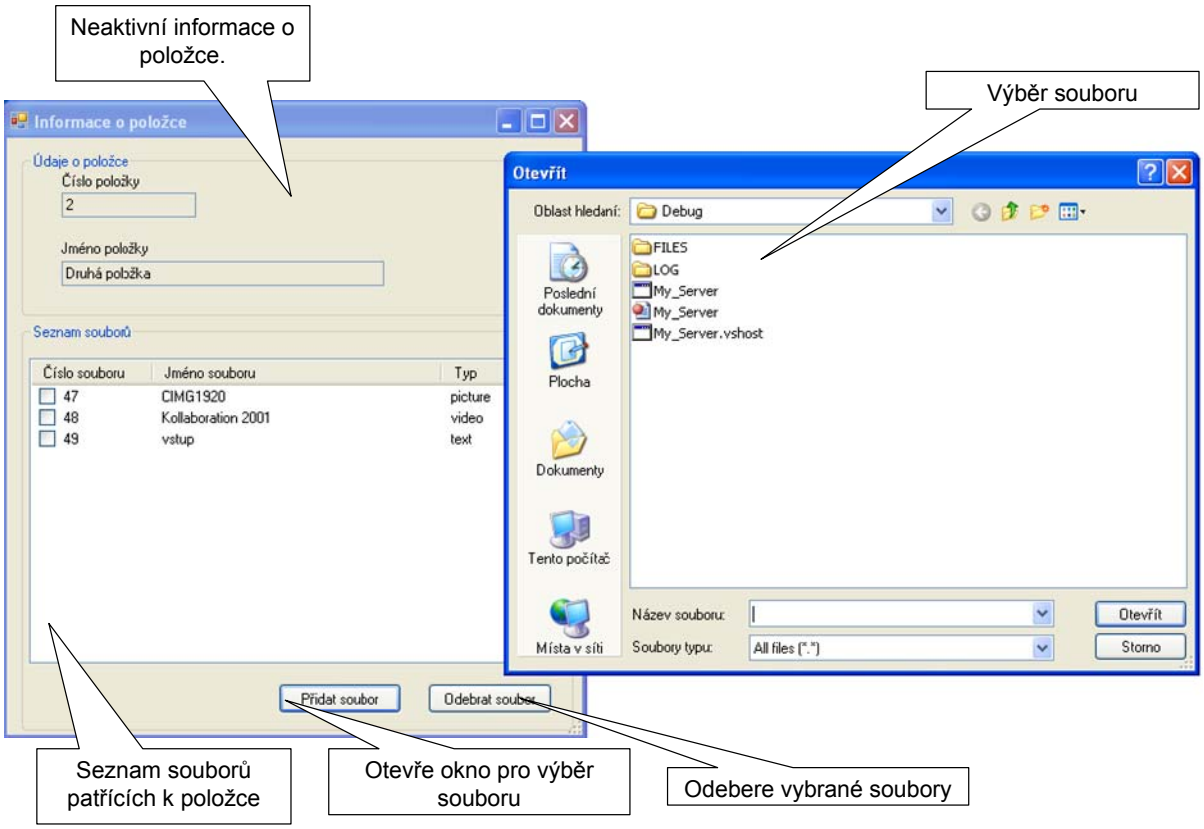
Vybereme – li položku *Přidat* v menu *Soubor* zobrazí se poslední formulář pro přidání položky. Do tohoto formuláře zadáme číslo a jméno položky, kterou chceme vytvořit. Obě textová pole musí být vyplněny a do textového pole *číslo položky* musí být zadáno číslo, jinak program zobrazí chybou hlášku.



Obrázek 3: Přidání položky

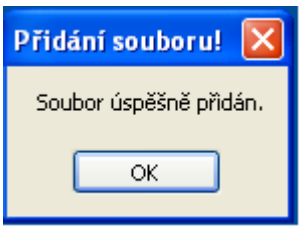
V administračním formuláři máme možnost editovat danou položku, pokud této možnosti využijeme, zobrazí se formulář pro editaci, který má podobný vzhled jako formulář pro přidání položky, ale obsahuje seznam souborů patřících k dané položce. I v tomto seznamu je možnost zobrazení rolovacího menu po stisknutí pravého tlačítka myši na příslušném řádku. Pro zobrazení okna pro výběr souboru je potřeba stisknout tlačítko *Přidat soubor*. Pro vymazání více souborů najednou musíme označit soubory, které chceme odstranit. Toto učiníme pomoci zaškrtávacího tlačítka

zobrazeného u každého ze souborů. Pro odstranění označených souborů slouží tlačítko odstranit. Soubory se odstraní z databáze a pokud je neobsahuje jiná položka, odstraní se také z disku.



Obrázek 4: Přidání souboru k položce

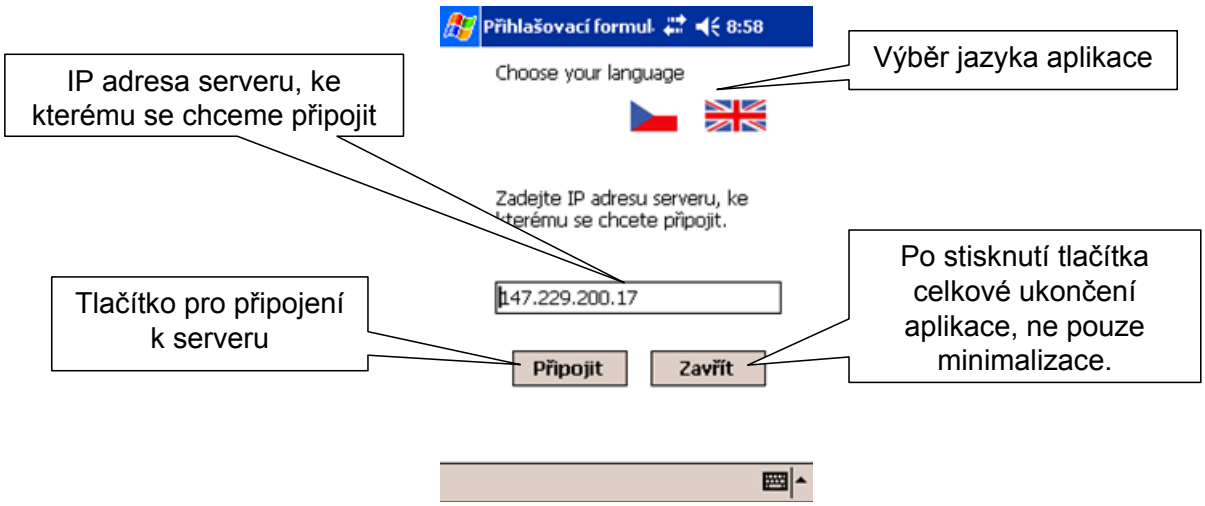
Pro lepší orientaci v administračním programu se zobrazují hlášky, které oznamují případné chyby nebo zprávy. Například při přidání souboru se zobrazí tato zpráva.



Obrázek 5: Zpráva

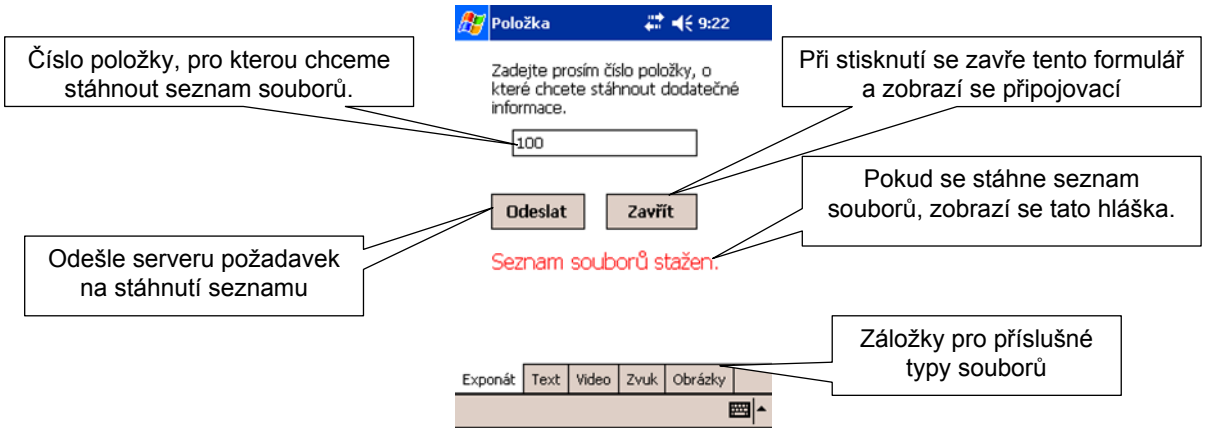
Příloha 2. Manuál ke klientovi

Po spuštění klienta příslušným *exe* souborem se zobrazí úvodní formulář pro připojení k serveru. Do textového pole se zadává IP adresa serveru, ke kterému se chceme připojit. Je možnost zobrazení aplikace ve dvou jazycích, pro změnu jazyka jsou umístěny v horní části formuláře vlajky. Po kliknutí na některou z vlajek se změní jazyk aplikace. Jazyk lze měnit i za běhu aplikace.



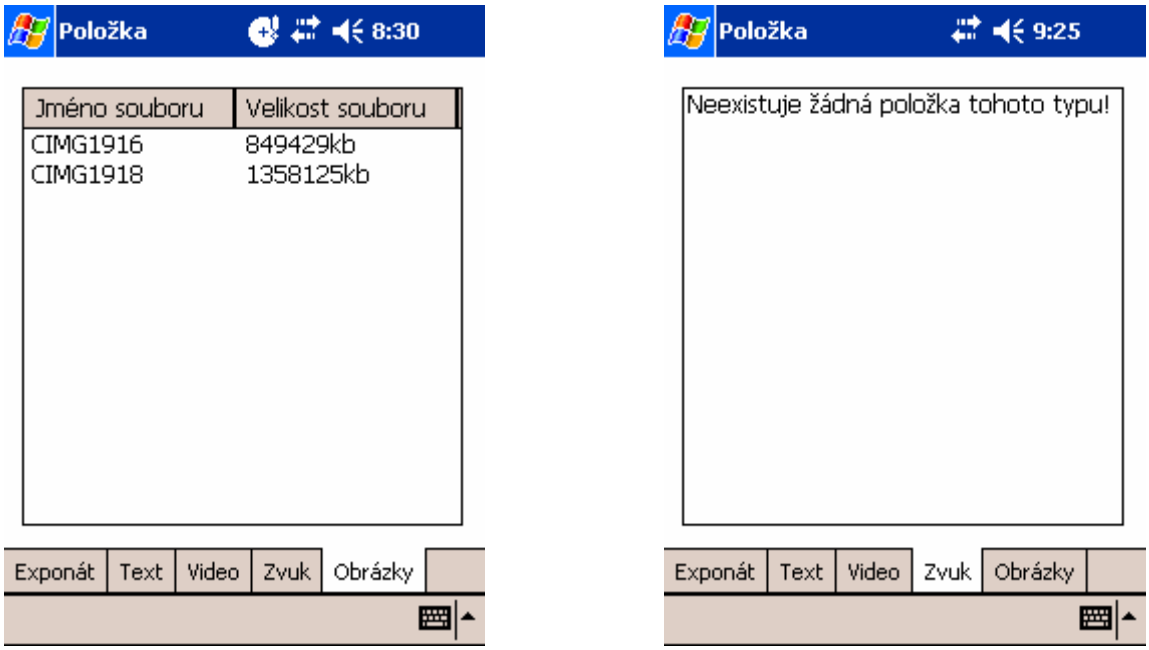
Obrázek 1: Úvodní formulář pro připojení k serveru

Po připojení se zobrazí okno obsahující záložky pro příslušný typ souborů. Do textového pole v první záložce zadáváme číslo položky, pro kterou chceme od serveru poslat seznam souborů. Při úspěšném stáhnutí seznamu se zobrazí zpráva. Při stisknutí tlačítka *Zavřít* se zavře aktuální formulář a zobrazí se opět připojovací.



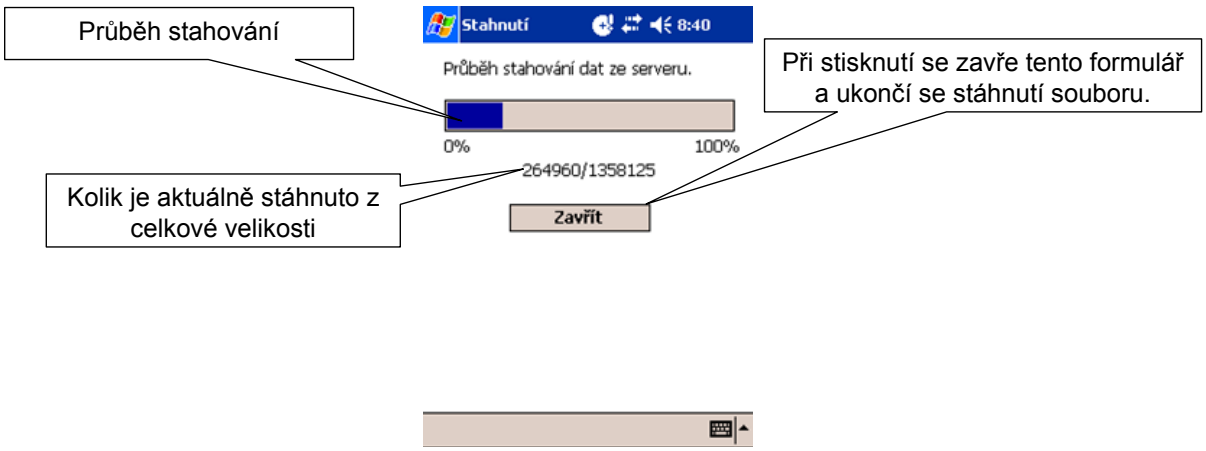
Obrázek 2: Odeslání čísla položky serveru

Po kliknutí na záložku reprezentující typ souboru, se zobrazí seznam souborů příslušného typu. Pokud neexistuje žádný soubor daného typu zobrazí se o tom informace.



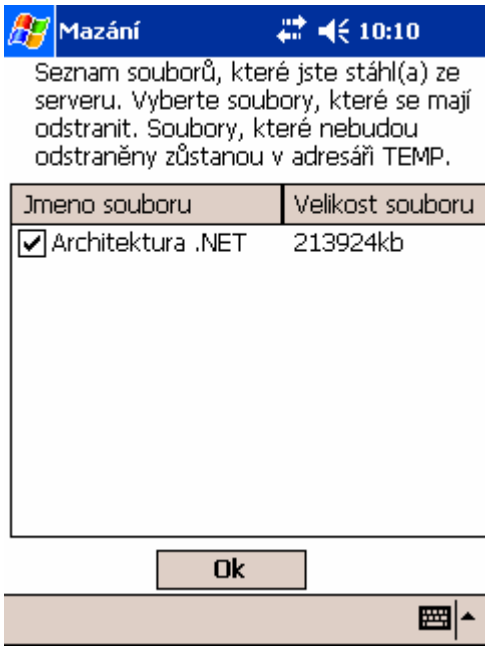
Obrázek 3: Možnosti zobrazení položek daného typu

Po dvojitém kliku na soubor, se začne soubor stahovat ze serveru, průběh stahování je zobrazen v následujícím formuláři. Stáhnutí souboru se ukončí stiskem tlačítka *zavřít*



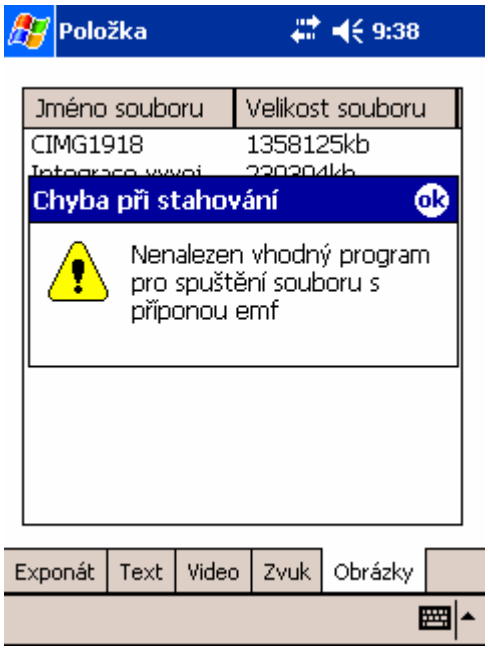
Obrázek 4: Možnosti zobrazení položek daného typu

Při ukončení aplikace tlačítkem zavřít se zobrazí seznam souborů, které se za běhu aplikace stáhli. U každého souboru je zobrazeno zaškrtávací tlačítko. Každý soubor, který bude označen, se po stisku tlačítka *Ok* odstraní z adresáře TEMP, který se nachází v adresáři odkud se spouští aplikace a aplikace se ukončí. Neoznačené soubory v tomto adresáři zůstanou.



Obrázek 5: Poslední formulář zobrazující seznam stažených souborů

I u klienta jsou zobrazeny některé chybové hlášky, které uživateli usnadňují orientaci v programu. V případě, že se po stažení souboru nenajde spustitelný program, vypíše program zprávu, která vypadá takto



Obrázek 6: Chybová zpráva

Spuštění klienta

Pro spuštění klienta musí být na zařízení Pocket PC nainstalovaný Compact .NET Framework 2.0. Do adresáře, do kterého se nakopíruje spustitelný soubor klienta, musí být nahrány i adresáře obsahující soubory pro lokalizaci.