VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

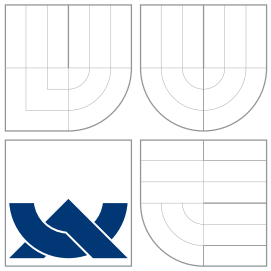
SYSTÉM PRO VZDÁLENOU SPRÁVU AKTUALIZACÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

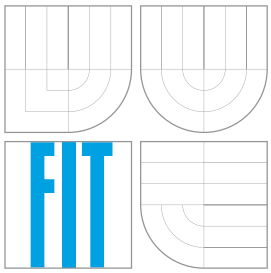
AUTOR PRÁCE
AUTHOR

JAKUB KADLUBIEC

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO VZDÁLENOU SPRÁVU AKTUALIZACÍ

SYSTEM FOR REMOTE UPDATE MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB KADLUBIEC

VEDOUCÍ PRÁCE
SUPERVISOR

Mgr. ROMAN TRCHALÍK

BRNO 2010

Abstrakt

Tato bakalářská práce se zaměřuje na návrh a implementaci systému pro správu aktualizací. Tento systém by měl nabídnout způsob, jakým mohou vydavatelé softwaru aktualizovat své produkty. Systém byl implementován v jazyce C# s využitím Windows Forms pro vytvoření uživatelského rozhraní, WCF jako komunikačního rozhraní a Microsoft SQL Serveru jako databázového serveru. Textová část práce popisuje průběh vývoje, od analýzy požadavků po samotnou implementaci a testování.

Abstract

The aim of this bachelor thesis is analysis and implementation of a system for remote update management. The implemented system should offer a way for software developers to update their products. The system is implemented in C# programming language using Windows Forms as an API for the graphical user interface, WCF as a communication API and Microsoft SQL Server as a database server. Textual part of the thesis describes each step, which was made to achieve the predetermined goals (this includes requirements analysis, system analysis, implementation and testing).

Klíčová slova

SOA, přenos souborů, aktualizace aplikací, Microsoft .NET Framework, C#, Windows Forms, WCF, MSSQL, třívrstvá architektura

Keywords

SOA, file transfer, application updates, Microsoft .NET Framework, C#, Windows Forms, WCF, MSSQL, multi-tier architecture

Citace

Jakub Kadlubiec: Systém pro vzdálenou správu aktualizací, bakalářská práce, Brno, FIT VUT v Brně, 2010

Systém pro vzdálenou správu aktualizací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Mgr. Romana Trchalíka

.....
Jakub Kadlubiec
19. května 2010

Poděkování

Děkuji svému vedoucímu Mgr. Romanu Trchalíkovi za trpělivost a ochotu, kterou prokázal při řešení tohoto projektu. V průběhu práce na projektu jsem od něj obdržel mnoho hodnotných rad, které mi velmi pomohly při řešení, a mnoho z nich nesporně využiju v dalším studiu.

Obsah

Kapitola 1

Úvod

Jedna ze zásad softwarového inženýrství říká, že každý složitější softwarový systém obsahuje nějaké chyby [15]. V případě, že se jedná o software vyráběny na zakázku, součásti smlouvy je většinou povinnost vydavatele svůj software dále aktualizovat a záplatovat chyby. Jako základní otázka se zde tedy nabízí to, jak tyto aktualizace realizovat. Jestliže je aplikace nasazena na minimálním počtu stanic, není nijak obtížné ani časově náročné provést aktualizaci ručně. Pokud je ovšem aplikace používaná na větším počtu počítačů, nastává problém, protože ruční instalace aktualizací a následná kontrola toho, zda všechno proběhlo v pořádku, je značně náročná na čas. A vzhledem k tomu, že čas jsou peníze, vydavatelé softwaru jsou většinou motivováni hledat řešení, jak urychlit proces aktualizací a kontroly jimi vydávaných aplikací. Systém vyvíjený v rámci této bakalářské práce by měl toto řešení nabízet.

1.1 Shrnutí účelu aplikace

Systém by měl sloužit především jako monitorovací a aktualizační nástroj. Následující dva cíle jsou popsány velmi obecně a slouží pro seznámení čtenáře s účelem systému. V kapitole 2 bude následovat seznam konkrétních požadavků kladených na systém.

1. **Aktualizace**
   Provádění vzdálených aktualizací běžících aplikací. Aktualizací se myslí nahrazení souborů aplikace. Obecně by měl aktualizační systém fungovat na jakékoliv aplikace, které nepotřebují ke svému běhu jiné informace než soubory v instalační složce.

2. **Monitorování**
   Zobrazování informací o klientských stanicích na jednom místě. Součástí těchto informací musí být aktuální stav stanice (připojena/nepřipojena) a aktuální verze instalovaných aplikací.

1.2 Úvod do použité technologie

Systém je určený pro použití na operačních systémech Microsoft Windows 2000 a vyšších. Je implementován kompletně pomocí technologií firmy Microsoft. Základním implementačním jazykem je C# a platformou Microsoft .NET Framework 3.5. Podrobnosti ohledně použitých nástrojů a technologií jsou k nalezení v kapitole 4.

1.3 Obsah technické zprávy

Technická zpráva popisuje všechny kroky, které byly podniknuty při vývoji aktualizačního systému. Proces vývoje softwaru je transformace požadavků uživatele na hotový softwarový produkt. Tento proces je složen z více částí.

- **Analýza požadavků**
  První etapou při vývoji softwarového produktu je analýza požadavků. Tento proces i s jeho výstupy jsou popsány v kapitole 2.

- **Architektonický návrh**
  Architektonický návrh navazuje na analýzu požadavků a jeho cílem je ujasnění koncepce systému a k jeho dekompozice. Dekompozice vyžaduje vymezení funkcionality jednotlivých podsystémů a definování vztahů mezi nimi (tedy definici komunikačního rozhraní).

  Architektonický návrh a jeho výstupy jsou popsány v kapitole 3.

- **Podrobný návrh**
  Podrobný návrh se soustřeďuje na podrobnou specifikaci softwarových součástí a na stanovení logické a fyzické struktury dat. Při podrobném návrhu se také plánují práce na implementaci součástí, s čímž souvisí vytvoření časového plánu, podle kterého by se měl vývoj řídit.

  Jelikož je systém pro správu aktualizací menší až střední softwarový projekt nebylo třeba provádět podrobný návrh ve velké míře. Veškeré výstupy podrobného návrhu jsou popsány v části Komponenty systému (kapitola 5.1).

- **Implementace**
  Implementace je etapa, ve které dochází k programové realizaci softwarových součástí.

  Popis systému po implementační stránce se nachází v kapitole 5.2. Tento popis může sloužit také jako stručná programová dokumentace.

- **Testování**
  Posledním krokem je testování, které by mělo odhalit případné chyby a ukázat zda-li byly splněny požadavky a systém je použitelný. Průběh testování je popsán v kapitole 5.3.

Text v této kapitole byl volně převzat z [15].

Kapitola 2

Analýza požadavků

Prvním krokem ve vývojovém procesu veškerých softwarových produktů je stanovení si cílů, kterých chce vývojář daným produktem dosáhnout. Tento proces se nazývá analýza požadavků.

Analýza požadavků se obvykle skládá ze tří kroků, jejichž následující popis byl volně převzat z [15].

1. **Sběr požadavků**
   Účelem prvního kroku je získat od pozdějších uživatelů aplikace jejich požadavky a představy o vyvíjeném systému. Sběr informací probíhá převážně formou prostého pohovoru člověka zodpovědného za vývoj s uživatelem.

   Systém pro správu aktualizací je určen pro vnitřní použití v softwarových firmách. Rozhovor mohl být proto veden na odborné úrovni, čímž odpadlo mnoho nejasností vzniklých rozdílnou technickou úrovní programátorů a uživatelů.

2. **Analýza získaných požadavků**
   Získané požadavky je třeba převést na formu jasně srozumitelnou, neobsahující nejasné, nekompletní nebo protichůdné informace.

3. **Zápis požadavků**
   Požadavky získané jako výstup předchozích kroků je nutné zaznamenat nejlépe do strukturované, přehledné formy. U každého požadavku je vhodné uvést odhadovanou prioritu a náročnost implementace. Podle těchto ukazatelů se následně řídí vývoj.

Sepsané požadavky byly, kvůli snadnější orientaci, kategorizovány podle jejich charakteru do čtyř kategorií.

- **Databázové požadavky**
  Databázové požadavky specifikují, jaké informace by měly být uloženy v perzistentní databázi v serverové části aplikace.

- **Funkcionální požadavky**
  Požadavky v této kategorii definují funkce, které by měl výsledný systém umět a upřesňují způsob realizace hlavních cílů systému specifikovaných v kapitole 1.1.

- **Požadavky na síťovou komunikaci**
  Tato kategorie obsahuje požadavky týkající se síťové komunikace.

- **Požadavky na uživatelské rozhraní**
  Poslední kategorie obsahuje požadavky, které definují, co všechno by mělo obsahovat uživatelské rozhraní a jak by se mělo chovat.

Následuje seznam požadavků setříděný podle kategorií. U každého požadavku je uvedena priorita (1 – nutnost až 5 – bonus) a odhadovaná náročnost implementace (1 – triviální až 5 – velmi náročné). Tyto požadavky byly sestaveny na základě procesu popsaného v úvodu kapitoly 2. K požadavkům byla po ukončení vývoje vedle předpokládané náročnosti přidaná skutečná náročnost implementace a u některých poznámka ke způsobu řešení.

2.1 Databázové požadavky

Požadavek č. 2.1.1 Databáze monitorovaných stanic
Schopnost serveru uchovávat informace o monitorovaných počítačích v databázi. Tato vlastnost umožní zobrazení informací i o stanicích, které nejsou aktuálně připojené k serveru.
Priorita: 1
Předpokládaná náročnost: 2
Skutečná náročnost: 3
Poznámka: Stanice jsou uloženy na serveru v databázi v tabulce `Workstation` viz. kapitola 3.1.

Požadavek č. 2.1.2 Databáze podporovaných aplikací
V databázi budou uloženy všechny aplikace, které aktualizační systém podporuje. U podporované aplikace musí být zaznamenáno její jméno, jméno spustitelného souboru (volitelné) a absolutní cesta k repozitáři, ve kterém se nacházejí verze.
Priorita: 1
Předpokládaná náročnost: 2
Skutečná náročnost: 2
Poznámka: Aplikace jsou uloženy na serveru v databázi v tabulce `SupportedApplication` viz. kapitola 3.1.

Požadavek č. 2.1.3 Databáze verzí aplikace
Ke každé podporované aplikaci může být přiřazeno 0 až n verzí číslovaných celými čísly počínaje 1. U každé verze bude možné specifikovat jméno, číslo verze a relativní cestu. Číslo verze určuje pořadí verze.
Priorita: 2
Předpokládaná náročnost: 1
Skutečná náročnost: 1
Poznámka: Verze aplikací se ukládají do tabulky `ApplicationVersion` viz. kapitola 3.1.

Požadavek č. 2.1.4 Databáze souborových změn
Každá verze se může lišit od verze předchozí minimálně jednou změnou v souborové struktuře. Tyto změny by měly být trvale zaznamenány u každé verze, aby nemuselo docházet k opakovanému počítání při každé aktualizaci. U změny je třeba zaznamenat, o jaký soubor (relativní cesta) a o jaký typ změny se jedná. Typ změny může být jeden z následujících:

- **Update** – obsah souboru byl změněn
- **Create** – soubor byl vytvořen
- **Delete** – soubor byl smazán

U verze číslo 1 se mohou nacházet pouze změny typu `Create`.
Priorita: 3
Předpokládaná náročnost: 2
Skutečná náročnost: 2
Poznámka: Připojení se ukládají do tabulky `FileChange` viz. kapitola 3.1.

Požadavek č. 2.1.5 Záznamy o instalacích[1] podporovaných aplikací

Na serveru budou zaznamenány všechny instalace podporovaných aplikací na pracovních stanicích společně s verzí, ve které se aplikace na dané stanici nachází. Podle toho seznamu bude následně rozhodováno, na které stanice se pošlou aktualizace.
Priorita: 1
Předpokládaná náročnost: 2
Skutečná náročnost: 3
Poznámka: Instalace se ukládají do tabulky `SupportedApplicationUsage` viz. kapitola 3.1.

Požadavek č. 2.1.6 Logování připojení klientů

Při každém připojení klienta (pracovní stanice) k serveru bude toto připojení zaznamenáno. Je nutné zaznamenat, která stanice se připojila a čas připojení (počáteční i koncový).
Priorita: 3
Předpokládaná náročnost: 1
Skutečná náročnost: 1
Poznámka: Připojení se ukládají do tabulky `WorkstationConnection` viz. kapitola 3.1.

Požadavek č. 2.1.7 Logování připojení ovládacího panelu k serveru

Každé připojení uživatelského panelu (tj. grafického rozhraní) serveru bude logováno. Kromě času připojení se bude zaznamenávat i IP adresa.
Priorita: 5
Předpokládaná náročnost: 1
Skutečná náročnost: 1
Poznámka: Připojení se ukládají do tabulky `ManagementConnection` viz. kapitola 3.1.

2.2 Funkční požadavky

Požadavek č. 2.2.1 Aktualizace souborů aplikací

Server musí být schopen poslat soubory potřebné pro aktualizaci aplikací na stanice specifikované uživatelem. Server bude posílat pouze soubory, které se změnily mezi verzemi, ve které se aktuálně nachází aplikace a nejnovější verzi. Uživatel může požádat o aktualizaci všech stanic, které mají nainstalovanou příslušnou aplikaci, anebo jedné konkrétní stanice.
Priorita: 1
Předpokládaná náročnost: 3
Skutečná náročnost: 5

Požadavek č. 2.2.2 Komunikace klienta s aktualizovanou aplikací

Klient musí být schopen poslat aplikaci základní příkazy typu „ukonči se“.
Priorita: 3

[1] Instalací se rozumí přihlášeni klienta k odběru aktualizací dané aplikace. Samotná instalace s sebou nenese vytváření adresářové struktury ani klasickou instalaci aplikace do systému.

Předpokládaná náročnost: 3
Skutečná náročnost: 2

Požadavek č. 2.2.3 Klient i server poběží jako service (služba)

Je nepřípustné aby klient či server běžel jako standardní exe aplikace. Služba má tu výhodu, že se spouští při startu systému a vždy běží jenom jeden proces nezávisle na počtu přihlášených uživatelů. Služby podporuje až Windows 2000 a vyšší. Jako exe aplikace poběží grafické rozhraní, které bude sbírat informace od služby a zobrazovat je.
Priorita: 1
Předpokládaná náročnost: 2
Skutečná náročnost: 2

Požadavek č. 2.2.4 Odolnost vůči chybám

Systém si musí umět poradit s výskytem nečekaných situací a dokázat řádně zareagovat. Především u serverové části je tento požadavek kritický, protože v případě spadnutí serveru přicházíme o statistiky, které server ukládá.
Priorita: 1
Předpokládaná náročnost: 3
Skutečná náročnost: 3

Požadavek č. 2.2.5 Centrální ovládání

Aktualizace budou prováděny a kompletně řízeny ze serveru. Klienti budou pouze přijímat obsahy aktualizovaných souborů a umisťovat je na správná místa. Aktualizace budou zasílány na klienty, kteří se předtím přihlásili k odběru aktualizací dané aplikace – pro přihlášení k odběru aktualizací se v této práci používá termín instalace.
Priorita: 2
Předpokládaná náročnost: 3
Skutečná náročnost: 3
Poznámka: Všechny aktualizace jsou spouštěny ze serveru. Klientská stanice se může jenom přihlásit k odběru aktualizací příslušné podporované aplikace, ale pak musí čekat až server iniciuje začátek aktualizace.

Požadavek č. 2.2.6 Automatická tvorba složek

Po obdržení příkazu na vytvoření souboru nacházejícího se v podadresáři klient automaticky vytvoří potřebnou adresářovou strukturu.
Priorita: 3
Předpokládaná náročnost: 1
Skutečná náročnost: 1

Požadavek č. 2.2.7 Automatické číslování verzí

Uživatel má možnost zadat ke každé verzi název a popisek. Systém inkrementálně dopočítá číslo verze, podle které se bude následně určovat pořadí verzí.
Priorita: 3
Předpokládaná náročnost: 2
Skutečná náročnost: 2

Požadavek č. 2.2.8 Automatické generování souborových změn mezi verzemi

Při vytváření nové verze systém automaticky vygeneruje seznam změn, kterými se nová verze liší od předchozí na základě kompletní adresářové a souborové struktury obou verzí.

Seznam změn bude trvale uložen.
Priorita: 2
Předpokládaná náročnost: 4
Skutečná náročnost: 3
Poznámka: Popis procesu generování seznamu změn se nachází ve scénářích použití, kapitola 3.2.5.

Požadavek č. 2.2.9 Seznam připojených a registrovaných stanic bude udržován aktuální

Pokud dojde k připojení, odpojení nebo registrování pracovní stanice server upozorní na tuto skutečnost uživatelské rozhraní.
Priorita: 4
Předpokládaná náročnost: 1
Skutečná náročnost: 2
Poznámka: Je zde nutné použít tzv. callback[2].

2.3 Požadavky na síťovou komunikaci

Požadavek č. 2.3.1 Funkčnost přes NAT

Aplikace musí fungovat i v případě, že klient bude za NATem.
Priorita: 1
Předpokládaná náročnost: 2 – 4 (Podle zvoleného řešení)
Skutečná náročnost: 2
Poznámka: Řešení popsáno v kapitole 3.3.

Požadavek č. 2.3.2 Problém „živého“ klienta

V případě ztráty konektivity se musí klient pokoušet o opětovné připojení. Nesmí nastat situace, že nebude žádný síťový problém, ale klient i přesto nebude připojen.
Priorita: 3
Předpokládaná náročnost: 2
Skutečná náročnost: 2

Požadavek č. 2.3.3 Asynchronní volání vzdálených funkcí

Každá síťová operace musí být spuštěna v samostatném vlákně, aby čekání na odpověď neblokovalo běh hlavní smyčky programu.
Priorita: 2
Předpokládaná náročnost: 4
Skutečná náročnost: 3

2.4 Požadavky na uživatelské rozhraní

Požadavek č. 2.4.1 Seznam registrovaných pracovních stanic

Server musí být schopen zobrazovat informace o klientech v přehledném grafickém rozhraní. Uživatelské rozhraní serveru musí zobrazovat všechny registrované stanice společně s informací, zda je stanice aktuálně přípojena a seznamem aplikací, které mají tyto stanice nainstalované.

[2]Callback je odeslání zprávy ze služby na klienta. Více informací se nachází v kapitole 4.2.3

Priorita: 1
Předpokládaná náročnost: 3
Skutečná náročnost: 3

Požadavek č. 2.4.2 Seznam podporovaných aplikací

Uživatelské rozhraní serveru musí zobrazovat seznam podporovaných aplikací, jejich verze a změny mezi verzemi.
Priorita: 1
Předpokládaná náročnost: 3
Skutečná náročnost: 3

Požadavek č. 2.4.3 Ovládání klienta přes uživatelské rozhraní

Nastavení na klientovi (např. adresa serveru, cesty k instalovaným aplikacím) se bude provádět skrze uživatelské rozhraní.
Priorita: 3
Předpokládaná náročnost: 3
Skutečná náročnost: 3

Požadavek č. 2.4.4 Napojení uživatelského rozhraní na výpočetní část

Výkonná část serveru i klienta bude poskytovat rozhraní, na které se bude připojovat grafické rozhraní. Toto připojení bude povoleno pouze z lokálního počítače, ale v případě potřeby by neměl být problém povolit připojení i ze vzdáleného počítače.
Priorita: 3
Předpokládaná náročnost: 4
Skutečná náročnost: 3

Požadavek č. 2.4.5 Výběr ze seznamu hodnot místo ručního psaní

Uživatelské rozhraní serveru i klienta by mělo uživateli ve všech možných případech nabízet výběr z přípustných hodnot místo vyzývání ho k ručnímu psaní hodnoty. Takové chování zrychluje práci a eliminuje chyby.
Priorita: 4
Předpokládaná náročnost: 2
Skutečná náročnost: 2
Poznámka: Výběr hodnoty byl použit ve všech případech zadávání cesty a v případě instalace aplikace na klientovi (uživatel si vybírá jednu z podporovaných aplikací).

Kapitola 3

Návrh systému

V této kapitole bude čtenář seznámen s návrhem systému po teoretické stránce. Budou diskutovány různé přístupy a možnosti řešení vybraných problémů.

Při návrhu samotné aktualizační logiky přicházely v úvahu dva možné způsoby řešení. První možnost počítá s tím, že se bude vždy aktualizovat pouze na nejnovější verzi, kdežto u druhé možnosti se uchovává seznam změn pro každou verzi, což umožňuje aktualizovat aplikace nejen na poslední verzi.

- **Aktualizace pouze na nejnovější verzi**
  Bude možné nahrávat jenom nejnovější verzi. Na serveru budou uloženy soubory vždy aktuální verze aplikace. Klient vždy zkontroluje kontrolní součty všech souborů, a když se bude nějaký soubor lišit od verze na serveru, tak si vyžádá aktuální verzi a provede nahrazení. U této možnosti se nepracuje s čísly verzí jako takovými. Podstatné je pouze to, zda-li je soubor na klientovi aktuální, tj. stejný jako na serveru.

- **Manipulace s verzemi aplikace**
  U každé verze aplikace bude uložen seznam změn, kterými se liší od předchozí verze. Při aktualizaci aplikace na novější verzi se budou iterativně provádět změny uložené u předchozích verzi. Oproti možnosti *Aktualizace pouze na nejnovější* je tento přístup náročnější na výkon (při vytváření nové verze je třeba generovat seznam změn oproti předchozí verzi) a na správu (je nutné udržovat každou verzi v samostatné složce a při úpravě jediného souboru vytvářet novou verzi).

Z důvodu větší flexibility, robustnosti a rozšiřitelnosti byla výsledná aktualizační logika implementována podle metody *Manipulace s verzemi aplikace*.

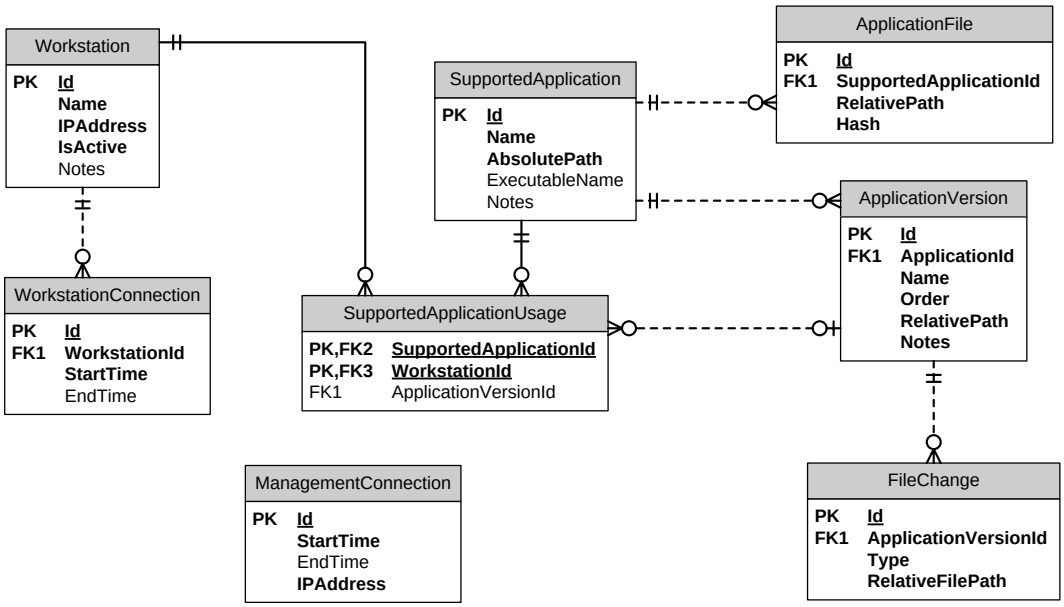
3.1 Návrh databáze

Na serveru je nutné udržovat data na fyzickém úložišti, kvůli perzistenci mezi jednotlivými běhy aplikace.

ER diagram na obrázku 3.1 popisuje návrh entitních množin a vztahů mezi němi. ER diagram používá notaci Crow's Foot pro znázornění kardinality [2].

3.1.1 Popis entitních množin

Entitní množina `Workstation` popisuje jednotlivé stanice. Ke každé stanici může být přiřazeno 0 až n připojení popsaných entitní množinou `WorkstationConnection`. Ka-



Obrázek 3.1: ER diagram serverových dat

ždá stanice může mít taky nainstalováno 0 až n aplikací reprezentovaných množinou `SupportedApplicationUsage`. V této množině je také zahrnuta informace o jakou aplikaci se jedná (odkaz na `SupportedApplication`) a na jaké verzi se nachází (`ApplicationVersion`). U každé aplikace je udržován seznam souborů (`ApplicationFile`) a u každé verze seznam změn, kterými se tato verze liší od verze předchozí (`FileChange`). Loguje se také připojení uživatelského rozhraní (`ManagementConnection`).

3.1.2 Zvolené fyzické úložiště

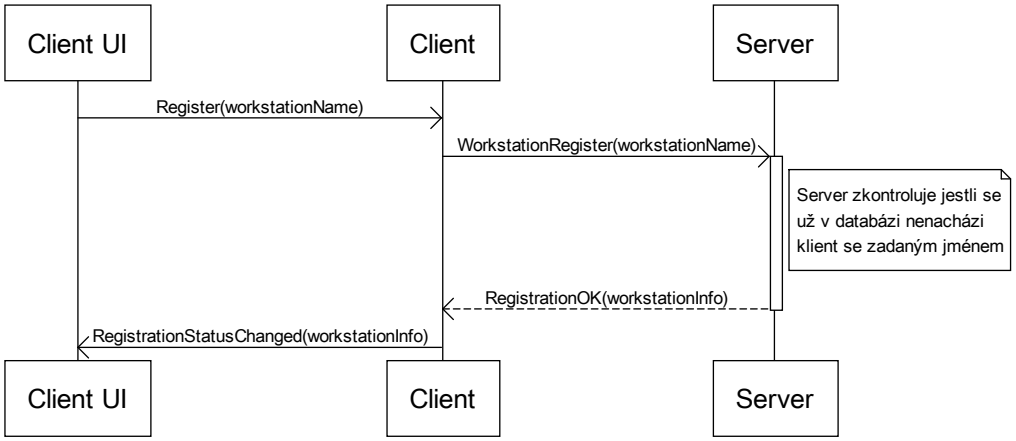
V serverové části aplikace jsou data uložena v relační databázi, konkrétně databázi Microsoft SQL Server. Uložení dat do relační databáze zaručuje oproti uložení v konfiguračních souborech rychlejší výběr dat, snadnější implementaci vztahů 1:n a v neposlední řadě přísnější typovou kontrolu a automatickou kontrolu integrity dat.

3.2 Scénáře použití

V této kapitole bude čtenář seznámen se způsobem realizace základních úkonů potřebných pro správný chod systému.

3.2.1 Registrace pracovní stanice

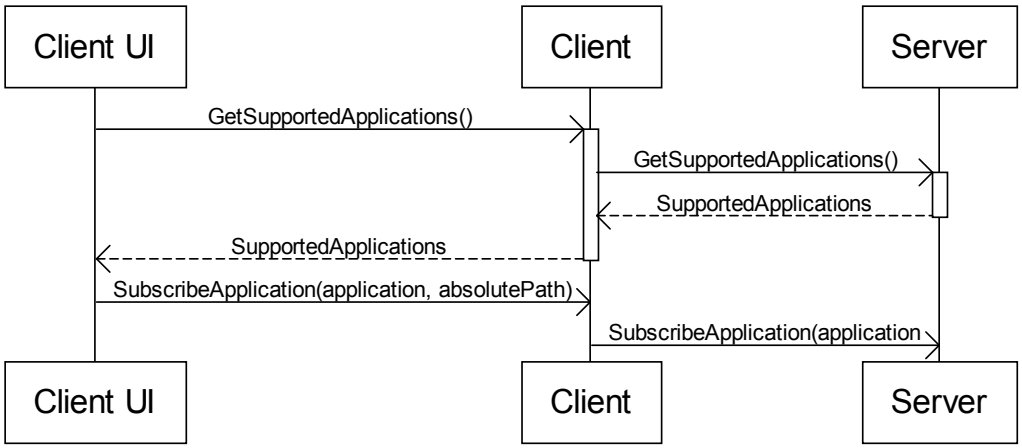
Před zahájením činnosti se musí každá pracovní stanice registrovat na serveru (pokud už registrace proběhla, stačí se přihlásit). Po úspěšné registraci obdrží stanice své identifikační číslo, které následně používá k přihlášení. Postup registrace je znázorněn na sekvenčním diagramu č. 3.2.



Obrázek 3.2: Registrace pracovní stanice

3.2.2 Přihlášení pracovní stanice k odběru aktualizací (instalace aplikace)

Před tím, než se uživatel přihlásí k odběru aktualizací dané aplikace, musí nejdříve získat seznam všech podporovaných aplikací a z něj si vybrat. Klient následně předá požadavek pro přihlášení k odběru aktualizací na server, a ten si přihlášení uloží do databáze (tabulka `SupportedApplicationUsage` viz. návrh databáze na obrázku 3.1) Průběh přihlášení stanice k odběru aktualizací znázorňuje diagram 3.3.



Obrázek 3.3: Přihlášení klienta k odběru aktualizací (instalace aplikace)

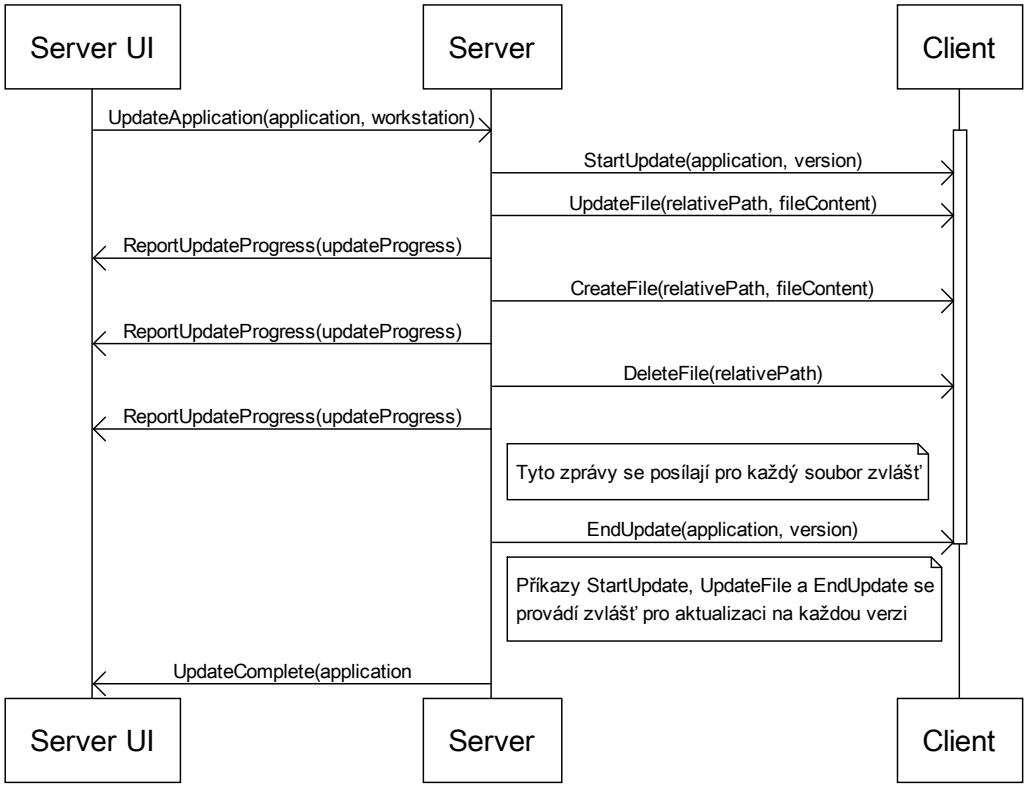
3.2.3 Aktualizace aplikace

Při aktualizaci aplikace na jedné pracovní stanici je nezbytné nejdříve zjistit kolik a jakých verzí se nachází mezi aktuální a nejnovější verzi aplikace. Poté server pro každou verzi posílá klientovi následující příkazy:

1. `UpdateStart(application, version)`, kde parametr `application` udává, která aplikace se aktualizuje a parametr `version`, na kterou verzi se aktualizuje.

2. Změny jednotlivých souborů. Pro každý měněný soubor se provede jeden z následujících příkazů, na základě toho, o jakou změnu se jedná.
   - `UpdateFile(relativePath, fileContent)` pro nahrazení souboru.
   - `CreateFile(relativePath, fileContent)` pro vytvoření nového souboru.
   - `DeleteFile(relativePath)` pro smazání souboru.
3. `EndUpdate(application, version)`. Tento příkaz se pošle pracovní stanici po provedení aktualizací všech souborů. Parametry jsou stejné jako u příkazu `UpdateStart`. Po tomto kroku si může klientská stanice zaznamenat, že má aplikaci ve verzi `version`.

Průběh aktualizace aplikace na jedné stanici nejlépe znázorní diagram 3.4.



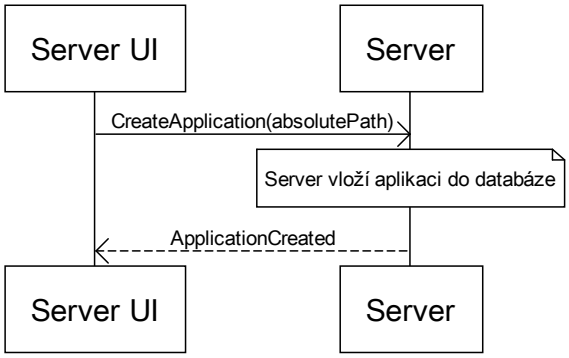
Obrázek 3.4: Aktualizace aplikace

3.2.4 Vytvoření nové podporované aplikace

Při vytváření nové podporované aplikace musí server provést tyto dva kroky:

1. Kontrola zadané cesty. Absolutní cesta, určující umístěni kořene aplikace, musí vést na složku s právem zápisu. Do této složky se budou následně vkládat verze aplikace.
2. Vložení informací o aplikaci do databáze. V tomto kroku se zároveň kontroluje porušení unikátnosti názvu aplikace.

Po úspěšném provedení těchto kroků je aplikace připravena pro přidávání verzí. Průběh vytvoření nové podporované aplikace je zobrazen na sekvenčním diagramu č. 3.5.



Obrázek 3.5: Vytvoření podporované aplikace

3.2.5 Přidání nové verze aplikace

Přidání nové verze aplikace představuje nejnáročnější operaci skládající se z mnoha kroků.

1. Nejdříve uživatel zadá název verze a relativní cestu k nové verzi. V tomto případě bude vždy relativní cesta pouze název podsložky umístěné pod kořenovým adresářem aplikace. Uživatel má také možnost zadat textový popis právě vytvářené verze.

2. Následně server zkontroluje, jestli zadaná cesta existuje, a jestli obsahuje adresář `root`, obsahující soubory aktuálně přidávané verze aplikace.

3. V tomto kroku server vloží údaje o nové verzi do databáze. Současně se spočítá pořadové číslo nové verze způsobem, že se inkrementuje číslo aktuální verze u dané aplikace.

4. **Generování seznamu změn**
   Generování seznamu změn se skládá z následujících kroků:

   (a) Získáme seznam všech souborů v adresáři nové verze a procházíme ho.

   (b) Vložíme relativní cestu aktuálně procházeného souboru do seznamu `relativePathList`. Pokud se už v adresáři nenachází žádný soubor, pokračujeme krokem 4i.

   (c) Pro aktuálně procházený soubor spočítáme hash (konkrétně MD5 [20]).

   (d) Najdeme v databázi (tabulka `ApplicationFile`) soubor se stejnou relativní cestou patřící k dané aplikaci.

   (e) Pokud soubor nebyl nalezen, znamená to, že je v této verzi nově přidán. Musíme proto vytvořit záznam v tabulce `ApplicationFile` a zároveň vytvořit záznam v tabulce `FileChange` s informacemi o souboru a příznakem `Create`. Dále pokračujeme krokem 4b.

   (f) Pokud soubor byl nalezen, znamená to, že už v aplikaci existuje. V tomto případě porovnáme hash aktuálně procházeného souboru s hashem uloženým v databázi u nalezeného souboru.

   (g) Pokud jsou hashe stejné pokračujeme krokem 4b.

(h) V opačném případě upravíme hash v databázi a vytvoříme nový záznam v tabulce `FileChange` s příznakem `Update`. Dále pokračujeme krokem 4b.

(i) Získání všech souborů, které se nachází v tabulce `ApplicationFile`, ale nejsou obsaženy v seznamu `relativePathList`. Pro každý soubor je vygenerována změna typu `Delete` a je odstraněn záznam z tabulky `ApplicationFile`.

3.3 Síťová komunikace

3.3.1 Transportní protokol

Windows Communication Foundation nabízí více transportních protokolů[1] na výběr. Mezi základní patří protokol http a net.tcp. Transportní protokol je jedna z charakteristik tzv. bindingů (podrobnosti viz. kapitola 4.2.3).

Protokol http je vhodný pro vytváření služeb používaných i jinými platformami než WCF – používá se především pro webové služby. Je textový, neoptimalizovaný a neposkytuje pokročilé možnosti nastavení. Oproti tomu protokol net.tcp (NetTcpBinding) je binární, optimalizovaný a poskytuje nejvíc možností nastavení. Jeho omezením je, že je podporován pouze platformou WCF. Toto omezení v našem případě není podstatné, protože všechny subjekty jsou napsány pod .NET a pracují s WCF. Z těchto důvodů byl jako transportní protokol vybrán net.tcp.

3.3.2 Komunikační vztahy mezi subjekty

Jedním ze základních požadavků kladených na systém je, že musí být funkční i v případě, že se klient nachází za NATem. Z toho důvodu je nerealizovatelná možnost, že byl klient kontaktován serverem pouze v případě nutnosti zjištěni stavu nebo provedení úkonů. Jako nejlepší řešení se jeví varianta, kdy se klient při spuštění připojí k serveru a udržuje spojení otevřené po celou dobu činnosti. Pomocí takto otevřeného spojení má server kdykoli možnost „přebít“ NAT a poslat klientovi zprávu. Server také loguje připojení klientů.

Schéma síťové komunikace znázorňuje obrázek 3.6. Směr šípek označuje, který ze subjektu iniciuje spojení.

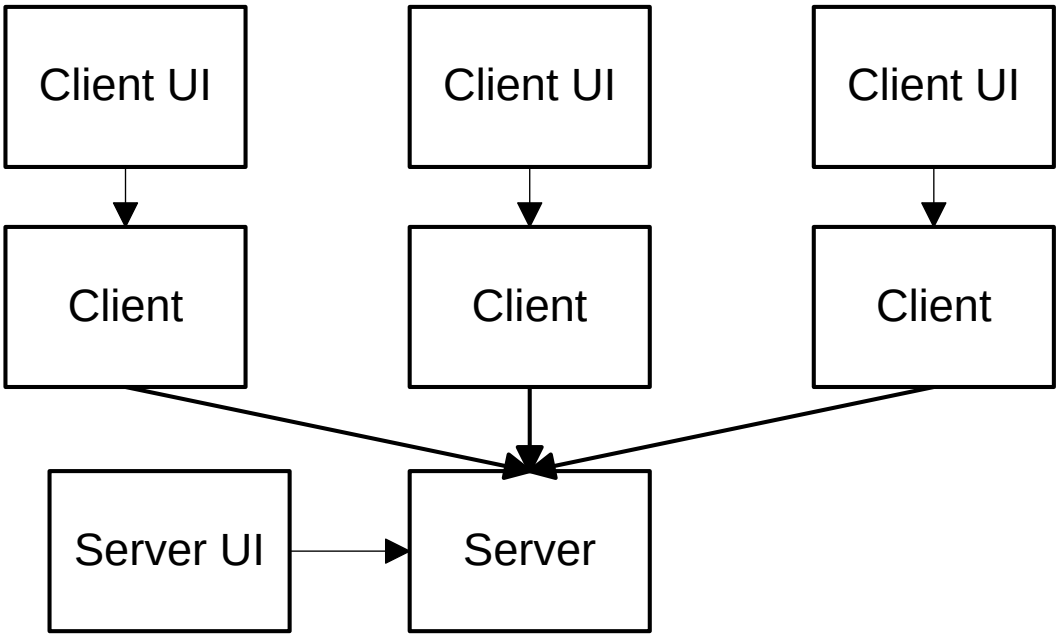
3.4 Grafické uživatelské rozhraní

Klient i server musí být ovládaní pomocí grafického uživatelského rozhraní. V úvahu přichází implementace uživatelského rozhraní jako webové nebo okenní aplikace.

Jako ovládací panel pro klientskou část byla zvolena okenní aplikace, která se ke klientské službě připojuje pomocí TCP spojení.

U serverové části je výběr vhodného uživatelského rozhraní komplikovanější, jelikož je možné použít obě varianty. Na serverový počítač není problém nainstalovat webový server, který by zprostředkovával uživatelské rozhraní jako webovou aplikaci. Webové rozhraní je výhodné především proto, že si uživatel vystačí s webovým prohlížečem a nenutí ho k používaní specializované aplikace. Naproti tomu okenní aplikace netrpí omezeními danými protokolem http a umožňuji serveru kontaktovat klienta. Díky tomu spuštěna aplikace sama reaguje na změny a promítá aktuální stav v reálném čase.

[1] Transportním protokolem se zde myslí protokol, který přenáší SOAP data [6]. Nejde o protokol transportní vrstvy.



Obrázek 3.6: Schéma síťové komunikace

Především z důvodu výhody dané možnosti kontaktování klienta serverem byla jako uživatelské rozhraní serveru zvolena okenní aplikace. Detaily ohledně implementace uživatelského rozhraní jsou popsány v kapitole 5.2.3. Nicméně pro tento účel by dobře sloužila i webová aplikace. Je to jedno z možných rozšíření do budoucna.

Kapitola 4

Použité nástroje a technologie

Tato kapitola obsahuje stručný přehled nástrojů a technologií použitých pro vývoj systému pro vzdálenou správu aktualizací. Tabulka s kompletním seznamem použitých nástrojů se nachází v příloze B.

4.1 Visual Studio 2008

Visual Studio 2008 je vývojové prostředí od firmy Microsoft určené především pro vývoj .NET aplikací. Visual Studio 2008 poskytuje pokročilé vývojové nástroje, ladicí funkce a funkce pro práci s databázemi usnadňující vývoj. Může být použito pro vývoj mnoha typů aplikací, od konzolových aplikací po aplikace s grafickým rozhraním, ať už webovým (ASP.NET, Silverlight) nebo desktopovým (Windows Forms, Windows Presentation Foundation).

Aplikace vyvíjené ve Visual Studio jsou organizované do tzv. solution (řešení), zapouzdřující projekty, které tvoří dílčí, sémanticky příbuzné části aplikace [4].

4.2 Jazyk C# a prostředí .NET Framework

4.2.1 Microsoft .NET Framework

.NET Framework je platforma pro vytváření a provozování aplikací. Skládá se ze společného jazykového běhového modulu – CLR[1] a knihovny tříd (Framework Class Library – FCL). CLR abstrahuje služby operačního systému a slouží jako vykonávací jádro pro *řízené aplikace* – aplikace, jejichž všechny akce podléhají schválení u CLR. FCL poskytuje objektově orientované API, z něhož aplikace řízené CLR čerpají. Aplikace rámce .NET, vypouští WinAPI, MFC, ATL, COM a další nástroje a technologie a místo nich pracují s FCL. Volání funkcí API Windows je možné, ale obecně se nedoporučuje, protože takové přechody mezi vykonáváním řízeného a strojového kódu omezují výkonnost a bezpečnost aplikace [19].

[1] Common Language Runtime, běhový modul či runtimový modul. CLR u .NET je obdobou Java Virtual Machine u Javy.

4.2.2 Jazyk C#

C# je relativně nový[2] programovací jazyk vyvinutý firmou Microsoft společně s prostředím Microsoft .NET Framework. Jazyk C# společně s jazykem VB.NET jsou základními a nejčastěji používanými jazyky pod .NET Frameworkem [17].

Jazyk C# je založený na jazycích C++ a Java (a je tedy nepřímým potomkem jazyka C, ze kterého čerpá syntaxi).

C# lze využít k tvorbě databázových programů, webových aplikací a stránek, webových služeb, formulářových aplikací ve Windows, softwaru pro mobilní zařízení (PDA a mobilní telefony) atd.

4.2.3 Windows Communication Foundation

O veškerou síťovou komunikaci v této bakalářské práci se stará Windows Communication Foundation (WCF). WCF bylo uvedeno současně s platformou .NET 3.0 jako API speciálně určené pro vytváření distribuovaných systémů [21]. WCF poskytuje jednotný, rozšiřitelný programovací objektový model, který může být použit pro interakci s množstvím dříve používaných technologií pro vytváření distribuovaných služeb (DCOM, .NET remoting, webové služby, atd.). Oproti dříve používaným technologiím nabízí WCF jednotné rozhraní pro tvorbu aplikací orientovaných na služby (SOA). Pomocí WCF lze publikovat služby mnoha různými způsoby. Například pokud se k navrhované aplikaci budou připojovat pouze klienti z operačního systému Windows, je vhodné použít různé binární protokoly založené na TCP pro zajištění maximální rychlosti. Stejná služba může být publikována pomocí XML webové služby, aby byla použitelná i pro klienty používající jiný programovací jazyk nebo operační systém.

Windows Communication Foundation ABC

Aby mohl klient komunikovat se službou, obě strany se nejdříve musí dohodnout na tzv. ABC. ABC ve světě WCF představuje zkratku pro základní stavební pilíře WCF, tj. *Address*, *Binding* a *Contract*.

- **Address** (adresa)
  Adresa je lokace služby v síti. Skládá se z transportního protokolu (http, net.tcp, atd.), názvu stanice (IP adresa nebo doménové jméno), portu a cesty ke službě.
- **Binding** (vazba)
  Aplikace hostující službu musí definovat *binding*, který bude používat ke komunikaci s klienty. *Binding* specifikuje následující charakteristiky
  - Transportní protokol používaný pro přenášení dat (HTTP, TCP, pojmenované roury, MSMQ).
  - Typ spojení (jednosměrné, požadavek-odpověď, obousměrné).
  - Kódování zpráv (binární, XML).
  - Další rozšíření specifické pro jednotlivé *bindings* (bezpečnost, transakce, atd.).

[2] Jazyk C# byl poprvé veřejně oznámen v roce 2000 pod názvem Cool (C-like Object Oriented Language) [10].

WCF nabízí na výběr několik připravených *binding*. V případě potřeby existuje možnost definovat si *binding* vlastní. Z nejdůležitějších předdefinovaných *binding* je vhodné vypsat pár reprezentantů založených na HTTP a pár na TCP.

  - **Bindingy založené na HTTP**
    `BasicHttpBinding`, `WSHttpBinding`, `WSDualHttpBinding` jsou *bindingy* založené na HTTP. Všechny zapouzdřují přenášená data do textového SOAP protokolu a používají HTTP jako transportní protokol. Díky těmto vlastnostem jsou vhodné pro používání s webovými službami, které budou konzumovány i jinými než Windowsovými aplikacemi.
  - **Bindingy založené na TCP**
    Pokud jsou klienty služby publikované pomocí WCF aplikace používající rovněž .NET Framework 3.0 nebo novější může být výrazně zvýšená rychlost přenosu použitím *bindingu* založeného na TCP, což zajistí, že data budou kódovaná do kompaktního binárního formátu místo XML. Pokud se klient i služba nachází na jedné stanici je nejvhodnější použít `NetNamedPipeBinding`, který používá pojmenované roury pro přenos dat. Pokud se klient i služba mohou nacházet na různých stanicích je nejlepší volbou `NetTcpBinding` používající klasické TCP jako protokol transportní vrstvy. Tento *binding* navíc nabízí pokročilé možnosti nastavení bezpečnosti, transakci a spolehlivých relací.

- **Contract** (smlouva)
  Contract popisuje metody (tj. název metody, návratovou hodnotu a parametry) publikované službou. Většinou jde o *interface* s příslušnými atributy[3]. Implementace služby je pak třída implementující daný *interface*.

Při definování služby hostitelská aplikace publikuje tzv. *endpointy*, které specifikují ABC služby. Z toho plyne, že každá služba může být publikována na více adresách, přístupná pomocí více protokolů, atd..

4.2.4 LINQ

LINQ (Language Integrated Query) je programovací model nabízející novou metodologii, která zjednodušuje a sjednocuje implementaci libovolného typu přístupu k datům [18]. LINQ nenutí programátora používat specifické knihovny pro přístup k různým typům dat. Namísto toho definuje jednotné rozhraní pro dotazování se, třídění, propojování i vyhledávání ve velkém množství zdrojů dat.

LINQ se poprvé objevil v září 2005 jako technický náhled. Od té doby se vyvinul z rozšíření Microsoft Visual Studia 2005 do integrální součástí .NET Frameworku 3.5 a Visual Studia 2008, které byly vydány v listopadu 2007. První vydaná verze LINQ přímo podporuje několik datových zdrojů:

- LINQ pro objekty
- LINQ pro ADO.NET
  - LINQ pro SQL
  - LINQ pro datové sady

[3] `ServiceContract` na úrovni *interfaceu* a `OperationContract` na úrovní metod

  - LINQ pro entity
- LINQ pro XML

Prostřednictvím LINQ to SQL (LINQ pro SQL) probíhají veškeré dotazy na databázi v tomto projektu.

4.2.5 Windows Forms

Windows Forms je API pro programování grafických uživatelských rozhraní. Windows Forms je součásti .NET Frameworku a zapouzdřuje přístup k nativním Windows API objektům. Windows Forms aplikace je aplikace řízená událostmi.

Microsoft uvedl společně s .NET Frameworkem 3.0 nové grafické API Windows Presentation Foundation (WPF). WPF používá značkovací jazyk XAML pro vytvoření uživatelského rozhraní [11].

Jelikož bylo WPF v době vyvíjení této bakalářské práce relativně novou a neověřenou technologií, bylo jako grafické uživatelské rozhraní zvoleno Windows Forms.

4.3 Databáze SQL Server 2008

SQL Server 2008 je relační databázový systém firmy Microsoft. Jeho primárními dotazovacími jazyky jsou SQL a T-SQL [7]. Edice Express SQL Serveru 2008 je dostupná zdarma [8]. Právě tato edice je použitá jako databázový server pro uložení dat serverové části systému.

4.3.1 SQL Server Management Studio

SQL Server Management Studio je komplexní integrované prostředí pro správu databázového serveru SQL Server 2008, jeho součástí je i prostředí pro zadávání a ladění SQL příkazů [16]. V tomto nástroji byla navrhnuta a vytvořena celá struktura databáze pro tuto bakalářskou práci.

4.4 Virtualizační prostředí VirtualBox

VirtualBox je mocný virtualizační nástroj pro firemní i domácí použiti, který umí pracovat s 32 i 64 bitovými procesory. VirtualBox je vydáván jako svobodný software pod licencí GPL [13].

VirtualBox poskytuje svým uživatelům mnoho možností nastavení virtuálních počítačů. Pro testování aktualizačního systému je nejpodstatnější nastavení síťového módu. VirtualBox umí virtualizovat čtyři síťová rozhraní u každého virtuálního počítače. Každé z těchto rozhraní může být nastaveno do jednoho z podporovaných síťových módů: Not attached, Network Address Translation (NAT), Bridged networking, Internal networking nebo Host-only networking. Podrobné informace o síťových módech a dalších možnostech nastavení se nachází v manuálu [9].

Pro účely testování aplikace typu klient-server se nejvíce hodí mód Host-only networking. Pomocí módu Host-only networking lze vytvořit síť, ve které se bude nacházet hostitelský počítač a množina virtuálních počítačů bez nutnosti použiti fyzického rozhraní hostitelského počítače. Na hostitelském počítači je vytvořeno virtuální síťové rozhraní podobné lokální smyčce, které zprostředkovává spojení mezi virtuálními a hostitelským počítačem.

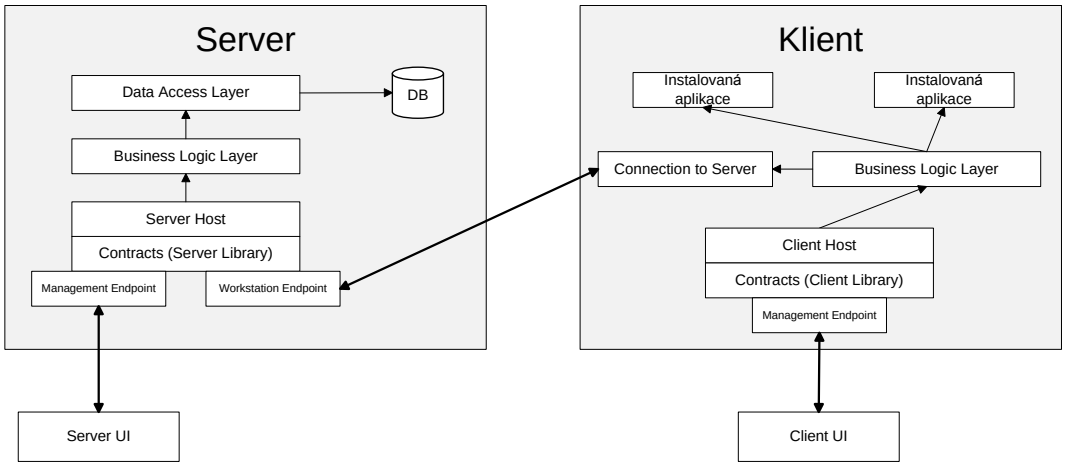
Kapitola 5

Implementace a testování

V této kapitole bude čtenář seznámen s vývojem systému WorkstationManager (tak byla nazvána vyvíjená aplikace) po implementační stránce. Nejdříve bude systém představen z té nejobecnější stránky (popis komponent systému) a následně budou jednotlivé součásti rozebrány do hloubky. Poslední část této kapitoly se zaměřuje na popis testování.

5.1 Komponenty systému

Systém je rozdělen na několik částečně nezávislých komponent. Každá komponenta splňuje jeden konkrétní úkol. Na obrázku 5.1 je znázorněn diagram všech komponent systému a vztahů mezi nimi.



Obrázek 5.1: Diagram součásti systému

Součásti nazvané endpoint znázorňují koncové body, ke kterým se připojují jiné aplikace. Na diagramu jsou znázorněny čtyři aplikace. `Server` a `Klient` představují výkonné části, které běží jako Windows Service na pozadí a starají se o veškerou funkčnost systému. Tyto aplikace jsou dále rozděleny na projekty, které se starají o konkrétní úkoly v rámci aplikace. Rozdělení jedné aplikace na projekty usnadňuje dodržování zásad objektového programování, kdy každý projekt plní jednu funkci a „má dovoleno“ volat třídy pouze z odkazovaných projektů. Na diagramu představují šipky odkazy mezi projekty. Projekt, na který ukazuje šipka, je možné volat z projektu, u kterého šipka začíná.

Komponenty `Server UI` a `Client UI` reprezentují aplikace uživatelských rozhraní, která se napojují na výpočetní služby pomocí TCP spojení.

5.1.1 Serverová část

Serverová část je navrhnuta podle zásad vzoru třívrstvé architektury [12].

- **Data Access Layer**
  Data Access Layer, neboli databázová vrstva zprostředkovává přístup k databázi. Definuje také objektovou reprezentaci databázových entit, které umí načítat z databáze, upravovat, mazat nebo vkládat do databáze. Tyto entity reprezentují data v celém systému.

- **Business Logic Layer**
  Business Logic Layer, neboli aplikační vrstva je centrální část, která vykonává veškerou logiku aplikace.

- **Contracts**
  Součást s názvem Contracts definuje koncové body (tzv. endpointy), na které se mohou připojovat koncové stanice nebo uživatelské rozhraní. Tato část plní funkci komunikačního rozhraní mezi klienty a aplikací, přes kterou jdou veškeré příkazy, uživatelské vstupy, atd. I když tento projekt nemá nic společného se zobrazováním informací uživateli, v modelu třívrstvé architektury plní roli vrstvy prezentační, protože přijímá příkazy a zprostředkovává výsledky pro uživatele.

- **Server Host**
  ServerHost je vstupním bodem serverové aplikace a jeho hlavní funkcí je otevření příslušných portů pro poslouchání příchozích spojení.

- **DB**
  Součást DB představuje databázový server, který uchovává veškerá serverová data.

Komponenta `Server UI` představuje Windows Forms aplikaci sloužící pro ovládání serveru. Přes tuto aplikaci můžeme spravovat pracovní stanice, vytvářet nové verze aplikaci, atd. Tato aplikace se připojuje k hlavní serverové části pomocí WCF spojení.

5.1.2 Klientská část

V klientské části není třeba pracovat s databází, proto je její architektura co se týče přístupu k datům jednodušší. Na rozdíl od serveru musí však klient umět iniciovat spojení k serveru.

- **Business Logic Layer**
  Komponenta Business Logic Layer je jádro klientské aplikace a obstarává veškerou logiku. Jak je patrné z diagramu tento projekt přijímá příkazy od části `Contracts`, která zprostředkovává komunikaci s uživatelským rozhraním.

- **Contracts**
  Podobně jako u serverové části, komponenta Contracts definuje koncový bod pro připojení uživatelského rozhraní.

- **Connection to Server**
  Tato součást se umí připojit na koncový bod `Workstation Endpoint` vystavený do světa serverem.

- **Client Host**
  Projekt ClientHost je vstupním bodem klientské aplikace a stará se o otevření portů pro připojení klientského uživatelského rozhraní.

- **Instalovaná aplikace**
  Součást Instalovaná aplikace představuje aplikaci instalovanou na daném klientovi a spravovanou aktualizačním systémem.

Aplikace `Client UI` představuje uživatelské rozhraní klienta. Přes tuto aplikaci lze nastavovat parametry spojení, instalovat aplikace, atd.

5.2 Objektový model

Tato kapitola se zaměřuje na popis vyvíjeného systému po implementační stránce. Bude zde postupně rozebrána implementace klíčových součástí u jednotlivých komponent. Jelikož je systém rozdělen na více nezávislých částí (komponent), budou jednotlivé komponenty popsány samostatně.

Pokud není specifikováno jinak, tak se každá třída nachází ve stejnojmenném souboru.

5.2.1 Serverová služba

Tato sekce se bude zabývat implementací výkonné části serverů. Popis implementace uživatelského rozhraní následuje níže.

Databázová vrstva

Veškerá komunikace s databází probíhá skrze databázovou vrstvu definovanou v projektu `ServerDataAccessLayer`.

Práce s databází v databázové vrstvě je implementována pomocí LINQ to SQL. Pro používání LINQ to SQL je nutné nadefinovat tzv. datový kontext, který mapuje databázovou strukturu na objektovou reprezentaci.

Třída reprezentující datový kontext se jmenuje `WMDataClassesDataContext` a je definována v souboru WMDataClasses.dbml. Tento datový kontext definuje entity s názvy odpovídajícími názvům databázových tabulek (`Workstation`, `SupportedApplication`, atd.). Tyto entity slouží k reprezentaci databázových dat v celém projektu.

V databázové vrstvě se nachází, kromě datových entit, třídy *dotazovací* obsahující dotazy na databázi, které jsou určeny pro získávání, měnění, vytvoření nebo odstranění dat z databáze. Pro každou tabulku je definovaná jedná *dotazovací* třída. Všechny *dotazovací* třídy jsou potomky třídy `DALBase` a jejich název je složen z názvu databázové tabulky a přípony DAL (například `WorkstationDAL`, `SupportedApplicationDAL`.

Abstraktní třída `DALBase` má jedinou roli, a to zjednodušovat svým potomkům přístup k instanci objektu datového kontextu. Definuje proto chráněnou (protected) vlastnost `WMDataContext`, která vrací novou instanci třídy `WMDataClassesDataContext`.

Aplikační vrstva

Serverová aplikační vrstva, jejíž třídy jsou umístěny v projektu `ServerBusinessLogicLayer`, je odpovědná za veškerou logiku aplikace. Pouze tato vrstva má přístup ke členům vrstvy databázové – v některých případech pouze zprostředkovává výsledky volání databázové vrstvy vrstvě prezenční, v jiných případech je nutné aplikovat aplikační logiku (například pro kontrolu validity dat, doplnění údajů, úpravu výsledků, aj.).

V aplikační vrstvě má každá databázová tabulka svého reprezentanta (třídu) pobdobně jako v databázové vrstvě s tím rozdílem, že jméno třídy v aplikační vrstvě se skládá ze jména tabulky a přípony BLL (například `FileChangeBLL`, `SupportedApplicationBLL`).

Třída `SupportedApplicationBLL` je typickým reprezentantem třídy z aplikační vrstvy. Její metoda `GetAllSupportedApplicationsExtendedSummary()` zprostředkovává volání databázovou vrstvu a navíc upravuje výsledky na požadovaný tvar (tj. přidává informaci o poslední verzi). Metoda `CreateSupportedApplication()` také volá databázovou vrstvu, ale nejdříve zkontroluje, jestli specifikovaná absolutní cesta k aplikaci opravdu existuje.

Singletony v aplikační vrstvě Zvláštní pozornost si zaslouží třídy v aplikační vrstvě, které si pro svoji činnost potřebují pamatovat informace v paměti. Tyto třídy jsou implementovány podle návrhového vzoru Singleton [14], což zaručuje, že vždy bude existovat jenom jedna instance dané třídy.

Jednou z nejdůležitější tříd vůbec je třída `UpdateManagerBLL`. Jejím úkolem je kompletní řízení aktualizací. To obnáší rozhodování které soubory a komu se mají posílat. V souboru UpdateManagerBLL.cs se rovněž nachází definice abstraktní třídy `AUpdateFollower`, která obsahuje abstraktní metody potřebné k provedení aktualizace. Třída `UpdateManagerBLL` v sobě uchovává seznam instancí třídy `AUpdateFollower`, pomocí kterého předává komunikační vrstvě příkazy k provedení aktualizací.

Další třídou v této kategorii je třída `WorkstationBLL`. Tato třída v sobě obsahuje seznam všech pracovních stanic, u kterých si navíc pamatuje, zda-li je daná stanice právě připojená. Při požadavku na seznam všech stanic, vrací tato třída instance z vnořeného seznamu a ne z databáze, jak to je u klasických tříd aplikační vrstvy.

Rozšíření datových entit Ve složce DataContracts se nachází třídy, které rozšiřuji entity definované v databázové vrstvě, anebo vytvářejí zcela nové datové třídy určené pro přenos informací.

Všechny třídy v této složce mají atribut `DataContract` a jejich vlastnosti atribut `DataMember`, což jim dovoluje být serializovanými a deserializovanými. Serializovatelnost tříd je vyžadovaná při jejich přenášení při WCF operaci.

K nejdůležitějším třídám ve složce DataContracts patří třída `SupportedApplicationExtendedSummary`, která přidává ke třídě `SupportedApplication` informací o poslední verzi. Dále třída `WorkstationExtendedSummary`, která rozšiřuje třídu `Workstation` o aktuální stav připojení dané stanice. Třída `UpdateProgressState` v sobě nese informace o průběhu aktualizace.

Prezentační (komunikační) vrstva

Prezentační vrstva u této bakalářské práce není klasickou prezentační vrstvou specifikovanou modelem třívrstvé architektury [12], protože jejím cílem není zobrazovaní informací. Z

tohoto důvodu je tato vrstva nazývaná také vrstvou komunikační. Komunikační vrstva je implementována v projektu `ServerLibrary`. Tento projekt obsahuje referenci na aplikační vrstvu, a proto může tato vrstva používat třídy z aplikační vrstvy. Aplikační vrstva kontaktuje komunikační vrstvu pomocí události (eventu) definovaných v některé ze Singleton tříd nebo pomocí principu popsaného výše u třídy `UpdateManagerBLL`.

Důležitou součástí prezentační vrstvy je definice rozhraní `IServerManagementEndpoint` a `IServerWorkstationEndpoint`. Tato rozhraní slouží jako smlouva (contract) pro otevřené endpointy. Serverová služba specifikuje dva endpointy. Jeden pro připojení uživatelského rozhraní (ManagementEndpoint) a druhý pro připojení pracovních stanic – klientů (WorkstationEndpoint). Detaily ohledně principu fungování WCF komunikace jsou popsány v kapitole 4.2.3.

Třídy `ServerManagement` a `ServerWorkstation` implementují výše zmíněna rozhraní, čímž specifikují funkčnost, která bude provedena při volání jedné ze služeb.

5.2.2 Klientská část

Klientská služba odpovídá za podstatně méně úkolů než serverová. Její hlavní funkcí je připojení se k serveru a následné vykonávání jeho příkazu. V této sekci se nachází popis realizace těchto příkazů.

Aplikační logika

Jádrem serveru je projekt `ClientBusinessLogicLayer` a v něm se nacházející třída `ClientCore`. Tato třída je implementována podle návrhového vzoru Singleton a skrze ní probíhají veškeré operace, které provádí klient.

Součástí aplikační logiky je i statická třída `ProcessManipulation`, která umí najít a ukončit procesy, které se mají právě aktualizovat. Ve složce DataContracts se podobně jako u serveru nachází třídy určené pro přenos informací.

Přípojení k serveru

Projekt `ClientConnection` obsahuje referenci na službu serveru pro obsluhu klientských stanic[1], a proto dokáže jako jediný komunikovat se serverem. Tento projekt definuje třídu `ConnectionToServer`, která zapouzdřuje volání serverových metod.

Endpoint pro připojení uživatelského rozhraní

V projektu `ClientLibrary` se nachází definice endpointu pro připojení klientského uživatelského rozhraní ke službě. Endpoint používá interface `IClientManagementEndpoint` jako *Conract* a třídu `ClientManagemet` jako implementaci funkčnosti.

Tento projekt obsahuje referenci na Aplikační logiku, a proto může přímo předávat příkazy od uživatelského rozhraní třídě `ClientCore`. Třída `ClientCore` následně vykoná příkaz a pokud je nutné komunikovat se serverem, zavolá třídu `ConnectionToServer` z projektu `ClientConnection` a ta obstará komunikaci se serverem.

[1] Klientská služba se připojuje k Workstation Endpoint a má tedy k dispozici funkce definované v kontraktu `IServerWorkstationEndpoint`

5.2.3 Uživatelské rozhraní

Jak už bylo v tomto textu zmíněno, uživatelské rozhraní klienta i serveru je implementováno jako okenní Windows Forms aplikace. Uživatelská rozhraní fungují jako samostatné aplikace a k výkonným službám se připojují pomocí TCP spojení.

Po spuštění uživatelského rozhraní se zobrazí dialog umožňující přípojení k výkonné službě. Dialog připojení k serveru je zobrazen na obrázku C.1[2]. Klientský dialog pro přípojení ke službě vypadá obdobně.

Všechny síťové operace jsou prováděny v samostatném vlákně, aby čekání na odpověď neblokovalo běh hlavní smyčky. Tohoto chování je dosaženo pomocí komponenty `BackgroundWorker` [1].

Uživatelské rozhraní serveru

Hlavní okno uživatelského rozhraní serveru je rozděleno na dvě záložky. Na první záložce se jménem *Workstations* je v horní části zobrazen seznam všech registrovaných stanic, společně s informací, jestli jsou právě přihlášeny nebo odpojeny. Ve spodní části jsou zobrazeny instalované aplikace právě zvolené stanice. Z této záložky se dá spustit aktualizace jedné aplikace na jedné stanici pomocí tlačítka *Update Application*. Po kliknutí na toto tlačítko se aktualizuje právě zvolená aplikace. Snímek obrazovky s vybranou záložkou *Workstations* je na obrázku C.4.

Druhá záložka se jmenuje *Supported Applications* a věnuje se zobrazení a správě podporovaných aplikací. V horní části se nachází seznam všech podporovaných aplikací a pod ním seznam verzi právě vybrané aplikace. Po kliknutí na tlačítko *Create Application* se vyvolá dialog pro přidání nové aplikace. Tlačítko *Create Version* po kliknutí vyvolá dialog pro vytvoření nové verze právě zvolené aplikace. V dolní části se nachází také tlačítko *Show Changes*, které po kliknutí zobrazí okno se seznamem změn u jednotlivých verzí. Velmi důležité je tlačítko *Update All Workstations*, které po kliknutí spustí aktualizaci vybrané aplikace na všech stanicích. Snímek záložky *Supported Applications* je na obrázku C.5.

Po spuštění aktualizace se objeví dialog `UpdateForm`, ve kterém se bude promítat stav aktualizace. Tento dialog lze skrýt na pozadí pomocí tlačítka *Run on Background*. Snímek dialogu `UpdateForm` se nachází na obrázku C.6.

Uživatelské rozhraní klienta

Na hlavním formuláři klientského uživatelského rozhraní je zobrazen stav spojení klientské služby k serverové službě. Ve spodní části formuláře je tabulka s registračními údaji stanice. Po přihlášení stanice k serveru se zpřístupní tlačítko *Applications Settings*, které po kliknutí vyvolá formulář `ApplicationSettingsForm` určený pro správu instalovaných aplikací na klientské stanici. Snímek hlavního okna je zobrazen na obrázku C.3 a formulář `ApplicationSettingsForm` na obrázku C.2.

5.2.4 Společné knihovny

V řešení (solution) existuje mimo projektů zobrazených na diagramu 5.1 projekt `Library`, který poskytuje knihovny používané jak klientem, tak serverem.

[2] Všechny obrázky uživatelského rozhraní byly pořízeny v průběhu testování popsaného v kapitole 5.3 a nachází se v příloze C

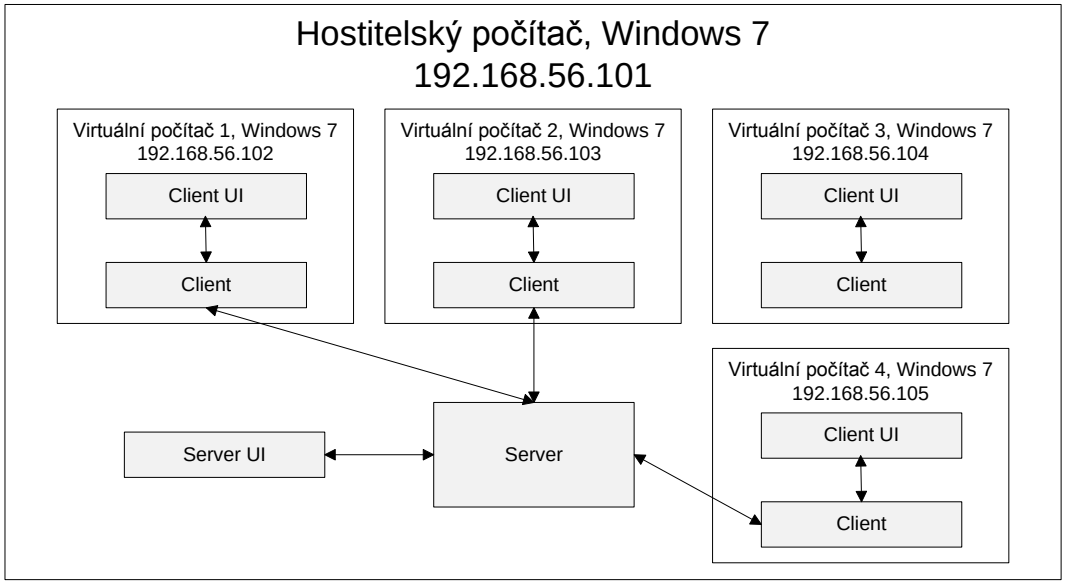
V této knihovně se nachází důležité bázové třídy usnadňující implementaci tříd účinkujících ve WCF komunikaci. Jedná se o třídu `EndpointBase`, která slouží jako bázová třída pro všechny implementace endpointů a třída `ConnectionToHostBase`, která je předkem všech klientských tříd.

Dále se zde nachází bázové třídy formulářů a dialogů (složka Forms), komponenty (složka Components) a uživatelské ovládací prvky (složka UserControls).

V tomto projektu se nachází rovněž statická třída `FileManipulation`, která definuje metody pro operace se soubory a adresáři (například `CreateFile()`, `UpdateFile()`, `GetFileContents()`, atd.).

5.3 Testování

Testování probíhalo na jednom počítači s 64-bitovým operačním systémem Windows 7 a nainstalovaným virtualizačním prostředím VirtualBox[3]. Ve VirtualBoxu byly nainstalovaný čtyři klientské počítače rovněž se 64-bitovým systémem Windows 7, které plnily roli klientů. Hostitelský počítač byl serverem. Schéma testovací soustavy je znázorněno na obrázku 5.2.



Obrázek 5.2: Testovací sestava

Systém byl testován na hudebním přehrávači jménem foobar2000 [3]. Tato aplikace byla zvolena pro testování, z důvodu, že je možné ji nainstalovat do jedné složky včetně konfiguračních souborů.

Na hostitelském počítači byl vytvořen repozitář, ve kterém se nacházely jednotlivé verze hudebního přehrávače foobar2000. Tyto verze byly následně přidány do databáze. Struktura repozitáře a obsah databáze se nachází v příloze D.

S takto připraveným repozitářem a databází byl systém připraven na provádění aktualizací. Výsledek testu byl pozitivní a aplikace foobar2008 byla po aktualizaci plně funkční na všech čtyřech virtuálních stanicích. V příloze C se nachází snímky uživatelského rozhraní pořízené v průběhu testování.

[3]Popis prostředí VirtualBox se nachází v kapitole 4.4

Kapitola 6

Závěr

Cílem této bakalářské práce bylo navržení a implementace systému pro vzdálenou správu aktualizací. Tato technická zpráva popisuje všechny etapy vývoje softwarového produktu, které byly podniknuty pro splnění dříve stanovených cílů. První etapou byl sběr a analýza požadavků, po které následoval teoretický návrh systému, na základě kterého byl systém implementován. Poslední etapa zahrnovala testování navrženého produktu.

Systém pro správu aktualizací je zajímavý především z důvodu, že je implementován s využitím velkého množství technologii. Důležitou roli hraje síťová komunikace, která byla implementována pomocí inovativní technologie Windows Communication Foundation. Systém používá databázi Microsoft SQL Server jako datové úložiště a pro přístup k datům byla použitá technologie LINQ. Dalším odvětvím, které bylo nutné si osvojit, byla tvorba uživatelského rozhraní, které rovněž představuje důležitý prvek celého projektu.

Systém je určen pro použití na operačních systémech Windows a nutnou podmínkou pro správný běh je nainstalovaný .NET Framework ve verzi 3.5 nebo vyšší. Testování proběhlo na sestavě se čtyřmi klienty a výsledky byly pozitivní. Pomocí implementovaného systému se podařilo aktualizovat v praxi používanou aplikaci foobar2000.

Implementovaný systém by mohl zaujmout především svou jednoduchostí a přímočarostí použití. Velký důraz byl kladen na kvalitu návrhu systému, který se zaměřoval na robustnost a snadnou rozšiřitelnost výsledného systému. V případě, že by byl používán na aktualizaci velkých softwarových systémů, bylo by nesporně nutné implementovat alespoň některé body ze seznamu rozšíření, který se nachází níže v této kapitole. Systém trpí především omezením na aktualizace aplikací, které ke svému běhu potřebují pouze soubory a malou flexibilitou.

Na závěr je vhodné poznamenat, že automatizované aktualizace aplikací jsou schopny značně urychlit proces údržby softwarových produktů, a jejich využití proto může být pro vydavatele softwaru velmi přínosné.

6.1 Možnosti rozšíření

Systém poskytuje velký prostor pro rozšiřovaní. Následující text popisuje možná rozšíření, která by po implementování nesporně zvýšila použitelnost systému v praxi.

Největší vadou systému je nemožnost aktualizovat registry, databázovou strukturu, konfigurační soubory na různých místech, atd. Tato omezení dovolují systému pracovat pouze s tzv. přenosnými aplikacemi (portable applications) [5].

V praxi by byla nesporně přínosná možnost naplánovat aktualizace na určitou hodinu, anebo nastavit, že se aktualizace provede automaticky po připojení stanice k serveru. V

současném stavu umožňuje systém aktualizovat pouze stanice, které jsou aktuálně k serveru připojené.

Pokud se provádí aktualizace například z verze 1 na 4, tak není třeba kopírovat soubor vytvořený ve verzi 2, pokud se ve verzi 3 zase odstraní. Pomohlo by implementování „chytrých“ iterativních aktualizací, které by samy uměly rozhodnout, které soubory je opravdu třeba přenášet.

Následuje heslovitý seznam dalších možných rozšíření:

- Aktualizace samotné klientské aplikace.
- Aktualizace na jinou verzi než poslední.
- Logování aktualizací.
- Přenášení pouze pozměněných částí místo celých souborů.
- Bezpečnost síťové komunikace.
- Webové rozhraní serveru.

Literatura

[1] BackgroundWorker Class (System.ComponentModel). [online], [cit. 8.5.2010].
URL http://msdn.microsoft.com/en-us/library/system.componentmodel.backgroundworker.aspx

[2] Crow's Foot Notation. [online], [cit. 8.5.2010].
URL http://www2.cs.uregina.ca/~bernatja/crowsfoot.html

[3] foobar2000. [online], [cit. 14.5.2010].
URL http://www.foobar2000.org/

[4] Microsoft Visual Studio: Přehled. [online], [cit. 15.5.2010].
URL http://www.microsoft.com/cze/msdn/produkty/vstudio/default.mspx

[5] Portable application — Wikipedia, The Free Encyclopedia. [online], [cit. 16.5.2010].
URL http://en.wikipedia.org/w/index.php?title=Portable_application&oldid=358914177

[6] SOAP Specifiaction. [online], [cit. 12.5.2010].
URL http://www.w3.org/TR/soap/

[7] SQL Server 2008 Overview, data platform, store data — Microsoft. [online], [cit. 15.5.2010].
URL http://www.microsoft.com/sqlserver/2008/en/us/

[8] SQL Server Express. [online], [cit. 15.5.2010].
URL http://www.microsoft.com/express/Database/

[9] Sun VirtualBox Manual. [online], [cit. 12.5.2010].
URL http://www.virtualbox.org/manual/UserManual.html

[10] The A-Z of Programming Languages: C#. [online], [cit. 15.5.2010].
URL http://www.computerworld.com.au/article/261958/a-z_programming_languages_c_/

[11] The Official Microsoft WPF and Windows Forms Site. [online], [cit. 15.5.2010].
URL http://windowsclient.net/

[12] Třívrstvá architektura — ITexpert.cz. [online], [cit. 5.5.2010].
URL http://www.itexpert.cz/trivrstva-architektura/

[13] VirtualBox. [online], [cit. 14.5.2010].
URL http://www.virtualbox.org/

[14] Bishopová, J.: *C#: návrhové vzory.* Zoner Press, 2010, ISBN 978-80-7413-076-2.

[15] Kočí, R.; Křena, B.: *Úvod do softwarového inženýrství, studijní opora.* 2006.

[16] Lacko, L.: *Jak vyzrát na Microsoft SQL Server 2008.* Computer Press, a.s., 2009, ISBN 978-80-251-2101-6.

[17] Nagel, C.: *C# 2008: programujeme profesionálně., [1].* Computer Press, a.s., 2009, ISBN 978-80-251-2401-7.

[18] Pialorsi, P.: *Microsoft LINQ: kompletní průvodce programátora / Vyd. 1.* Computer Press, a.s., 2003, ISBN 978-80-251-2735-3.

[19] Prosise, J.: *Programování v Microsoft .NET: webové aplikace v .NET Framework, C# a ASP.NET.* Computer Press, a.s., 2003, ISBN 80-7226-879-1.

[20] Rivest, R.: The MD5 Message-Digest Algorithm. RFC 1321 (Informational), [cit. 11.5.2010].
URL http://www.ietf.org/rfc/rfc1321.txt

[21] Troelsen, A.: *Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition.* Apress, 2007, ISBN 15-90598-84-9.

Seznam příloh

Dodatek A

Obsah CD

- `source\` – zdrojové kódy aplikace
- `bin\` – přeložený systém, každý projekt se nachází v samostatném adresáři
- `thesis\` – zdrojové kódy textové části v LaTeXu
- `thesis-print.pdf` – výsledná textová část ve verzi pro tisk
- `thesis-link.pdf` – výsledná textová část ve verzi pro prohlížení (obsahuje klikatelné odkazy)
- `README` – tento obsah CD plus požadavky na provoz systému

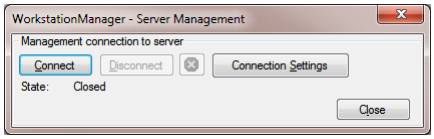
Dodatek B

Tabulka použitých nástrojů

Nástroj/technologie	**Verze**
Microsoft Visual Studio 2008	9.0.30729.1 SP
Microsoft .NET Framework	3.5 SP1
Microsoft Visual C# 2008	2008
VirtualBox	3.1.6r59338
Microsoft SQL Server Management Studio	10.0.2531.0
Microsoft SQL Server 2008 Express Edition	10.0.2531.0
Microsoft Office Visio 2007	12.0 SP2
WebSequenceDiagrams.com	6.5.2010

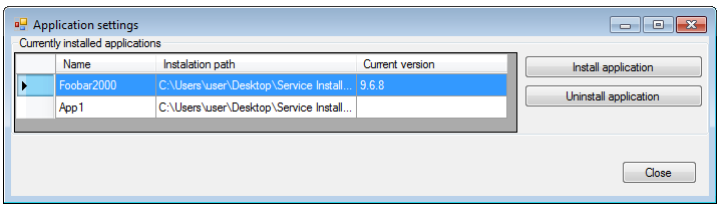
Dodatek C

Snímky uživatelského rozhraní

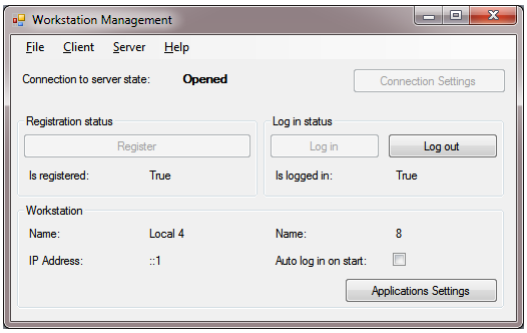


Obrázek C.1: Dialog přípojení k serverové službě (dialog připojení ke klientské službě vypadá obdnobně)

C.1 Uživatelské rozhrani klienta

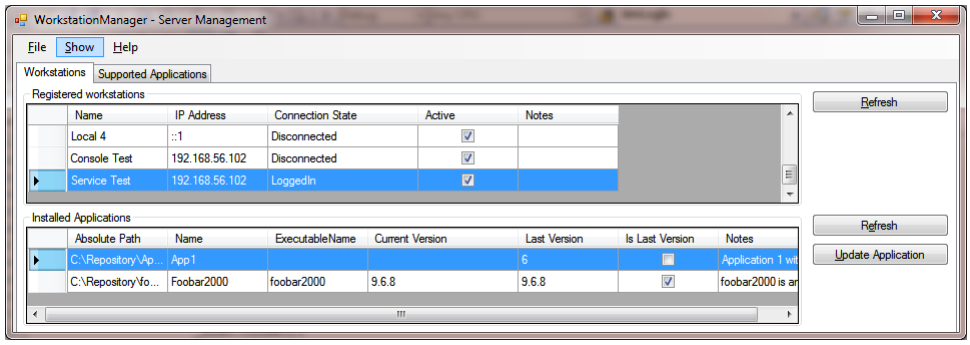


Obrázek C.2: Správa nainstalovaných aplikací

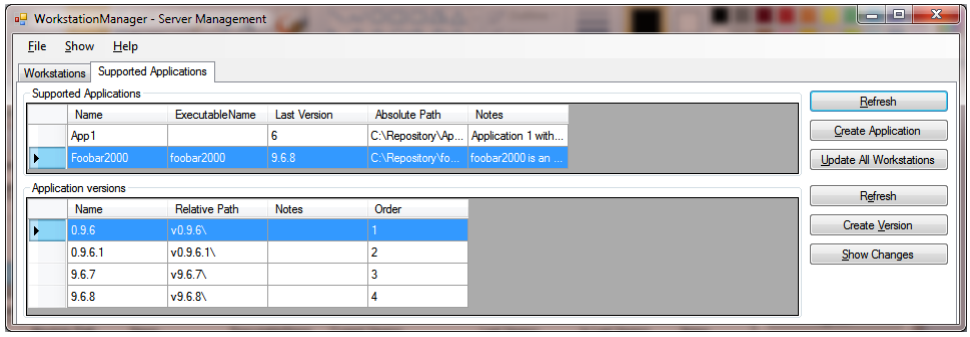


Obrázek C.3: Hlavní okno uživatelského rozhraní klienta

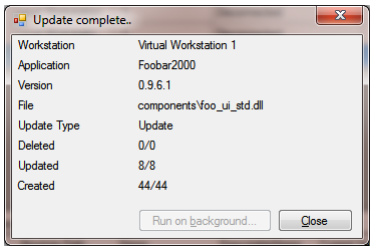
C.2 Uživatelské rozhrani serveru



Obrázek C.4: Správa pracovních stanic



Obrázek C.5: Správa podporovaných aplikací



Obrázek C.6: Dialog zobrazující průběh aktualizace

Dodatek D

Obsah repozitáře a databáze při testování

Adreářová struktura repozitáře je znázorněna níže:

C:\Repository
    \foobar2000
        \v0.9.6
            \root
                \components
                    \foo_input_std.dll
                \foobar2000.exe
                \installer.ini
                \ ... další soubory a složky verze 0.9.6
        \v0.9.6.1
            \root
                \components
                    \foo_input_std.dll
                \foobar2000.exe
                \installer.ini
                \ ... další soubory a složky verze 0.9.6.1
        \v9.6.7
            \root
                \components
                    \foo_input_std.dll
                \foobar2000.exe
                \installer.ini
                \ ... další soubory a složky verze 0.9.7
        \ ... další verze

V databázi se po přidání aplikace foobar2000 jako podporované aplikace a její verzi nacházely data představená v tabulkách D.1 až D.3.

Id	Name	AbsolutePath	ExecutableName	Notes
2	Foobar2000	C:\Repository\foobar2000\	foobar2000	foobar2000 is an (...)

Tabulka D.1: Obsah tabulky `SupportedApplication`

Id	ApplicationId	Name	Order	RelativePath	Notes
8	2	0.9.6	1	v0.9.6	
9	2	0.9.6.1	2	v0.9.6.1	
10	2	9.6.7	3	v9.6.7	
11	2	9.6.8	4	v9.6.8	

Tabulka D.2: Obsah tabulky `ApplicationVersion`

Id	ApplicationVersionId	Type	RelativeFilePath
18	8	1	foobar2000.cfg
19	8	1	foobar2000.exe
60	8	1	components\foo_input_std.dll
61	8	1	components\foo_ui_std.dll
... další změny ve verzi 8			
63	9	2	foobar2000.exe
68	9	2	components\foo_input_std.dll
69	9	2	components\foo_ui_std.dll
... další změny ve verzi 9			
71	10	2	foobar2000.exe
76	10	2	components\foo_input_std.dll
77	10	2	components\foo_ui_std.dll
... další změny ve verzi 10			
78	11	2	foobar2000.cfg
79	11	2	foobar2000.exe
86	11	2	components\foo_ui_std.dll
... další změny ve verzi 11			

Tabulka D.3: Obsah tabulky `FileChange`

Sloupec `Type` u tabulky `FileChange` určuje typ změny.

- 1 – změna typu `Create`
- 2 – změna typu `Update`
- 3 – změna typu `Delete`