VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
ÚSTAV INFORMATIKY

FACULTY OF BUSINESS AND MANAGEMENT
INSTITUTE OF INFORMATICS

NÁVRH APLIKACE PRO SPRÁVU TENKÝCH KLIENTŮ

THIN CLIENT MANAGEMENT SOLUTION DESIGN

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. FILIP JUHAŇÁK
AUTHOR

VEDOUCÍ PRÁCE — Ing. VIKTOR ONDRÁK, Ph.D.
SUPERVISOR

BRNO 2013

Vysoké učení technické v Brně
Fakulta podnikatelská

Akademický rok: 2012/13
Ústav informatiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Filip Juhaňák

Informační management (6209T015)

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách, Studijním a zkušebním řádem VUT v Brně a Směrnicí děkana pro realizaci bakalářských, magisterských a doktorských studijních programů zadává diplomovou práci s názvem:

Návrh aplikace pro správu tenkých klientů

v anglickém jazyce:

Thin Client Management Solution Design

Pokyny pro vypracování:

Úvod
Vymezení problému a cíle práce
Analýza současného stavu
Teoretická východiska řešení
Návrh řešení
Zhodnocení a závěr
Seznam použité literatury
Přílohy

Seznam odborné literatury:

ECKSTEIN, R. a M. CASABIANCA. XML pocket reference. 2nd ed. Sebastopol, CA: O'Reilly, 2001, 96 p. ISBN 05-960-0133-9.
KREIBICH, J. A. Using SQLite. 1st ed. Sebastopol, CA: O'Reilly, 2010. ISBN 05-965-2118-9.
REYNDERS, D. a E. WRIGHT. Practical TCP/IP and Ethernet networking. London: Elsevier, 2003. 306 p. ISBN 0750658061.
RICHTER, J. CLR via C♯. 3rd ed. Sebastopol: Microsoft Press, 2010. ISBN 07-356-4281-8.
VERMA, D. C. Principles of computer systems and network management. London: Springer, 2009. 260 p. ISBN 03-878-9008-4.

Vedoucí diplomové práce: Ing. Viktor Ondrák, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2012/13.

L.S.

doc. RNDr. Bedřich Půža, CSc.
Ředitel ústavu

doc. Ing. et Ing. Stanislav Škapa, Ph.D.
Děkan

V Brně, dne 28.2.2013

Abstrakt

Tato diplomová práce pojednává o návrhu a vývoji aplikace určené pro centralizovanou správu a konfiguraci tenkých klientů nabízených společností OldanyGroup s.r.o. Účelem této aplikace je zjednodušení administrace a sledování nasazených zařízení pro zákazníky, ale také zvýšení konkurenceschopnosti celého řešení tenkého klienta nabízeného společností.

Abstract

This thesis focuses on designing and developing a centralized management and configuration solution for the thin clients offered by the OldanyGroup s.r.o. company. The purpose of this application is to simplify the thin client administration and control for customers as well as to improve the competitiveness of the whole thin client solution offered by the company.

Klíčová slova

Tenký klient, centralizovaná správa, databáze, Windows Embedded, Microsoft .NET, TCP/IP protokol, C#, XML, JSON, SQLite, Windows API

Keywords

Thin client, centralized management, database, Windows Embedded, Microsoft .NET, TCP/IP protocol, C#, XML, JSON, SQLite, Windows API

Bibliografická citace této práce

JUHAŇÁK, F. *Návrh aplikace pro správu tenkých klientů.* Brno: Vysoké učení technické v Brně, Fakulta podnikatelská, 2013. 92 s. Vedoucí diplomové práce Ing. Viktor Ondrák, Ph.D.

Čestné prohlášení

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

V Brně, dne …… ……...

Podpis

Poděkování

Tímto bych chtěl poděkovat vedoucímu mé diplomové práce, Ing. Viktoru Ondrákovi, Ph.D., za konzultace a rady. Taktéž bych rád poděkoval kolegovi Tomáši Andrlemu za spolupráci při vývoji aplikace a testování.

Obsah

1 Úvod

Informační a komunikační technologie jsou pro moderní podniky dnes již prakticky nepostradatelným nástrojem – prostředkem, který jim umožňuje efektivně vykonávat jejich každodenní činnost a dosahovat tak vytyčených cílů – od malých firem až po největší korporace s tisíci zaměstnanců. V době, kdy vše závisí na kvalitě a dostupnosti informací, jakožto jednoho z nejdůležitějších zdrojů, si lze jen těžko představit jiný způsob, jak být efektivní, než s pomocí správných nástrojů a technik pro práci s nimi.

Stejně jako kterékoliv jiné technické vybavení, také informační a komunikační technologie však vyžadují řádnou údržbu a správu, aby mohl být jejich potenciál, usnadňovat a urychlovat naši každodenní práci s informacemi, plně realizován. Důležité je však také uvědomit si fakt, že tyto technologie jsou pro firmu zpravidla podpůrným nástrojem – měly by tedy fungovat vždy spolehlivě, rychle, ale zejména efektivně: s nízkými náklady na jejich pořízení, provoz i správu.

V dnešní době již existuje mnoho různých technologií a postupů, jak firemní IT infrastrukturu navrhnout a optimalizovat takovým způsobem, aby byla výkonná, spolehlivá a přitom byl její provoz cenově přijatelný. Důležitým faktorem začíná být i ekologie – trendem je zejména snižování spotřeby elektrické energie, recyklace elektroniky a případné použití zařízení s delším životním cyklem. Jedním z trendů poslední doby je virtualizace serverů i samotných pracovních stanic, tedy centralizace výkonných prostředků informačních technologií. Právě zde nachází uplatnění také tenký klient – zařízení, které efektivně nahrazuje PC na pracovištích, v kioscích, apod., ve prospěch nižší spotřeby energie, kompaktnosti, snadnější správy a vyšší bezpečnosti. Při nasazení většího počtu tenkých klientů (respektive i jakýchkoliv jiných koncových zařízení) se však stále nabízí otázka efektivní správy lokálních konfigurací aplikací a systémů těchto zařízení – právě tato problematika bude předmětem diplomové práce.

2 Vymezení problému a cíle práce

Cílem této diplomové práce je navrhnout a vytvořit efektivní řešení pro centralizovanou správu tenkých klientů *TiCtOG* společnosti OldanyGroup s.r.o. Toto řešení umožní zákazníkům zlepšit přehled o tenkých klientech nasazených ve firemní síti, jejich stavu, konfiguraci a poskytne jim nástroje potřebné pro správu konfigurací místních softwarových prostředků tenkých klientů a jejich vzdálené ovládání. Návrh a vývoj vlastní aplikace pro správu tenkých klientů je dále také motivován požadavkem na zkvalitnění poskytovaného řešení tenkého klienta jako celku a zvýšení konkurenceschopnosti a prodejnosti tohoto nabízeného produktu. Návrh a implementace aplikace budou provedeny na základě požadavků definovaných společností (tyto požadavky pak vychází zejména ze zkušeností s manuální konfigurací zařízení a z funkcionalit podporovaných v konkurenčních řešeních centralizované správy tenkých klientů), které budou detailněji rozepsány v dalších částech práce. Diplomová práce zahrne návrh a implementaci všech důležitých součástí tohoto řešení – služby Management agenta, serveru a aplikaci ovládací konzole, včetně instalačních procesů, podpůrných programů a knihoven.

3 Analýza současného stavu

V této části diplomové práce budou uvedeny základní informace o společnosti, pro kterou je celý projekt realizován a zhodnocen aktuální stav nabízeného řešení tenkého klienta. Dále bude zahrnut stručný přehled konkurentů a jejich řešení správy nabízených tenkých klientů. V poslední části analýzy budou rozebrány požadavky společnosti na navrhovanou aplikaci.

3.1 Informace o společnosti

Název:	OldanyGroup s.r.o.
Identifikační číslo:	274 79 188
Právní forma:	Společnost s ručením omezeným
Sídlo:	Jaromírova 158/54 Praha 2 – Nusle 128 00
Předmět podnikání:	Výroba, obchod a služby neuvedené v přílohách 1 až 3 živnostenského zákona[1]

Společnost OldanyGroup s.r.o. působí na českém trhu od roku 2005 (od roku 2011 již i na slovenském trhu). Zabývá se moderními a inovativními technologiemi v oblastech IT/ICT, zejména pak serverovou a desktopovou virtualizací na produktech společnosti VMware. Mezi poskytované služby patří také komplexní správa počítačových systémů (MS Windows i Linux), sítí, poradenství v těchto oblastech a možnost outsourcingu správy a monitorování systémů a sítí klientů včetně podpory koncových uživatelů (service desk/help desk). Firma OldanyGroup je certifikovaným partnerem mnoha předních společností a leaderů v oblasti IT/ICT. Z certifikovaných partnerství lze v souvislosti s touto diplomovou prací zmínit například následující: *VMware Solution Provider Enterprise Partner, Citrix SMB Specialist Solution Advisor, Microsoft Small Business Specialist, Microsoft SPLA Partner, IBM Business Partner, Intel Technology Provider Gold* a další (1).

[1] Základní údaje o společnosti dle obchodního rejstříku.

Mezi významné projekty společnosti patří v současné době služba *BeeScale* – firmou vyvíjená a provozovaná cloud computingová platforma, která umožňuje jejím uživatelům provoz na míru sestavených virtuálních serverů v cloudu, kde vlastník platí pouze za zdroje, které aktuálně využívá.

Z pohledu této práce je nejvýznamnějším projektem společnosti tenký klient *TiCtOG* – zařízení primárně navržené pro účely nasazení ve VDI (virtuálních desktopových infrastrukturách). Tento tenký klient je vyráběn a prodáván společností OldanyGroup, zejména v rámci vlastních projektů VDI, ale i samostatně jako víceúčelové zařízení s podporou dalších služeb a technologií. Návrh tohoto zařízení je podrobněji popsán v rámci bakalářské práce *Návrh tenkého klienta* (2).

3.2 Současný stav poskytovaného řešení

Zde bude popsán aktuálně nabízený produkt – tenký klient *TiCtOG*, který bude v rámci této diplomové práce dále rozšířen o řešení centrální správy. Popisovaný stav zároveň reflektuje změny, které byly na konfiguraci tohoto produktu provedeny dodatečně (motivací pro tyto změny byla převážně potřeba inovace a dostupnost jednotlivých komponent), a je tedy do jisté míry odlišný od původního návrhu dokumentovaného v práci *Návrh tenkého klienta* (2).

3.2.1 Konfigurace hardware

Oproti původním dvěma variantám konfigurace HW (dokumentovaným v bakalářské práci), je nyní nabízena pouze jediná varianta modelu tenkého klienta, označená názvem *MATRIX*. Motivací pro tuto změnu byl zejména malý zájem o pomalejší/levnější variantu *VECTOR* a také ukončení výroby některých komponent. Stávající konfigurace je postavena na inovované verzi základní desky a procesoru Intel Atom, je tedy výkonnější a spolu s dvojnásobnou kapacitou paměti RAM a jednotkou SSD (oproti původnímu úložišti typu compact flash) umožňuje plynulý provoz moderních operačních systémů a aplikací. Tato konfigurace je také velmi konkurenceschopná – výkonově parametry zařízení spadají mezi nejvýkonnější sestavy v oblasti tenkých klientů. Detailní parametry hardware konfigurace tenkého klienta jsou uvedeny v následující tabulce.

Aktuální specifikace HW tenkého klienta		
Základní deska	Intel D2550MUD2	Mini ITX formát
Procesor	Intel Atom D2550	1.86GHz, 2 jádra
Operační paměť	SO-DIMM DDR3	2GB, 1066MHz
Grafický čip	Intel GMA3650	integrovaná
Zvukový čip	Intel HD Audio	integrovaná
Místní úložiště	SSD, rozhraní SATA II	16GB
Síťová karta	Intel Pro 1000	integrovaná

Tabulka 1 – Aktuální HW konfigurace tenkého klienta (vlastní zpracování)

3.2.2 Konfigurace software

Původní tenký klient byl založen na operačním systému *Microsoft Windows Embedded Standard 2009*. Kompatibilita s tímto systémem bude zachována (nadále budou podporovány i starší verze zařízení s původní SW konfigurací), v zájmu inovace a konkurenceschopnosti byl však nahrazen novějším systémem ***Microsoft Windows Embedded Standard 7***, vycházejícím z dnes již velmi rozšířeného operačního systému *Microsoft Windows 7*. Konfigurace systému (uživatelské účty, aplikace, filtr zápisů apod.) zůstává stejná jako v původní verzi, stejně tak podpůrné aplikace, předinstalovaný software a alternativní uživatelské prostředí, které doznaly jen minimálních změn (zejména pro zajištění souběžné kompatibility s oběma verzemi operačního systému).

3.2.3 Centrální správa a vzdálená konfigurace

V současném stavu není pro tenké klienty *TiCtOG* zavedena žádná možnost vzdálené správy, konfigurace, nebo síťového nasazení operačního systému (původní návrh obsahuje pouze možnost instalace systému z vyměnitelných médií a případně obnovu ze záložního oddílu). Jediným aktuálně dostupným řešením je možnost použití aplikace *Microsoft System Center Configuration Manager*, která je kompatibilní se systémem *Windows Embedded Standard 7*, avšak tento způsob není zcela ideální, zejména vzhledem k vysoké ceně a nemožnosti pokrýt důležité oblasti konfigurace, jako

je například nastavení klienta *VMware View*. Tento nedostatek bude odstraněn návrhem a implementací vlastní aplikace pro centrální správu tenkých klientů.

3.3 Konkurenční řešení

V současné době je již implementace tenkých klientů a VDI poměrně rozšířenou praxí (zejména u středních a větších podniků v zahraničí, tento trend však proniká i na český trh) a proto přibývají noví výrobci těchto zařízení a stále se zvyšují nároky na jejich kvalitu a vybavení. Mimo dobrých parametrů hardware a požadované softwarové výbavy je jako výhoda (a dnes již téměř nutnost z pohledu konkurenceschopnosti) považováno také poskytnutí vlastního řešení pro management nabízených zařízení. V této části práce bude pro referenci proveden stručný průzkum řešení centrální správy, nabízených leadery v oblasti tenkých klientů a VDI.

DELL Wyse: DW Device Manager (3)

Společnost DELL Wyse[2] nabízí majitelům svých tenkých klientů řešení *Dell Wyse Device Manager* – jedná se o placený balík nástrojů na platformě MS Windows, dostupný ve dvou edicích: *Workgroup Edition* a *Enterprise Edition*; tyto se liší zejména v maximálním počtu spravovaných stanic, podporovaných typech databází a jistých funkčních omezeních. Mezi uváděné klíčové vlastnosti řešení patří:

- Správa a monitoring aktiv v reálném čase
- Vzdálená instalace operačního systému
- Politika konfigurací zařízení
- Delegace administračních oprávnění
- Bezpečná komunikace prostřednictvím HTTPS

Hewlett-Packard: HP Device Manager (4)

Společnost Hewlett-Packard poskytuje svým zákazníkům aplikaci *HP Device Manager*, jejíž licence je vázána na koupi tenkého klienta HP a je poskytována zdarma. Nástroje jsou určeny pro platformu MS Windows a sestávají z 5 komponent (Console, Server, Gateway, Agent a FTP server). Klíčové vlastnosti tohoto řešení jsou následující:

[2] Nový název společnosti je výsledkem akvizice původního výrobce Wyse korporací DELL Inc.

- Management aktiv
- Nastavení tenkých klientů a klonování konfigurace připojení
- Aktualizace software
- Vzdálené ovládání
- Vzdálená správa napájení

IGEL: Universal Management Suite (5)

Výrobce tenkých klientů a terminálů IGEL poskytuje svým zákazníkům zdarma ke stažení jednoduché a efektivní rozhraní pro centrální správu – *Universal Management Suite*. Jedná se o multiplatformní aplikaci (podpora systémů MS Windows i Linux) se zaměřením na snadnou správu zařízení a jednoduchou integraci do firemního prostředí. Typická použití dle výrobce zahrnují například následující scénáře:

- Automatické nastavení tenkých klientů při prvním připojení do sítě
- Změny nastavení zařízení, místních protokolů a softwarových nástrojů
- Reinstalace či aktualizace systému, diagnostika a podpora

NComputing: vSpace Management Center (6)

NComputing Inc., je moderní společnost zabývající se desktopovou virtualizací. Ke svým tenkým klientům řady L/M/N-Series nabízí zdarma řešení centrální správy *vSpace Management Center*, aplikaci na platformě MS Windows s intuitivním webovým rozhraním konzole pro správu. Klíčové vlastnosti aplikace jsou následující:

- Konfigurace založená na profilech
- Samostatný i skupinový management zařízení (1:1/1:N)
- Aktualizace firmware
- Sledování stavu v reálném čase a zaznamenávání
- Delegace administračních oprávnění
- Vzdálený přístup a ovládání

3.3.1 Zhodnocení

Z výše uvedených informací vyplývá, že poskytování vlastního řešení centrální správy je u významných výrobců tenkých klientů dnes již zcela běžnou praxí, a mělo by

tedy být součástí každého projektu, který chce uspět na tomto stále více konkurenčním trhu. Dalším zajímavým zjištěním je také cena a licencování těchto aplikací – přední výrobci dodávají tento software zcela zdarma jako komplementární produkt k prodávaným tenkým klientům, případně za poplatek, jehož výše se odvíjí od rozsahu nasazení a zvolené funkcionality.

3.4 Základní požadavky na řešení správy tenkých klientů

V diplomové práci popisované řešení centrální správy tenkých klientů bude navrženo podle základních požadavků definovaných společností – z těchto budou vycházet veškeré úvahy o návrhu jednotlivých částí aplikace a způsob jejich implementace. Požadované vlastnosti budou podrobněji rozebrány.

3.4.1 Poskytovaná funkcionalita

S ohledem na funkcionalitu poskytovanou ve výše zmíněných konkurenčních řešeních a na časté konfigurační úkony při instalaci tenkých klientů byly stanoveny následující požadované funkce:

- Vyhledání tenkých klientů na lokální síti a přidání do správy
- Monitorování aktuálního stavu spravovaných zařízení
- Shlukování spravovaných zařízení do skupin
- Detailní reporting o stavu vybraného zařízení (konfigurace, protokol systému a aplikací, historická data o vytížení systému, grafy)
- Podpora uživatelských účtů pro správu s omezeným/plným přístupem
- Vzdálený přístup a ovládání zařízení, vzdálené zapnutí/vypnutí/restart
- Podporované konfigurační úlohy:
    - Konfigurace klienta VMware View
    - Konfigurace MS Remote Desktop Services
    - Konfigurace Citrix XenDesktop
    - Konfigurace klienta RealVNC
    - Zpřístupnění prohlížeče MS Internet Explorer, domovská stránka
    - Změna hesla lokálního účtu "Administrator"
    - Změna hesla VNC serveru pro vzdálený přístup

- Změna názvu počítače (hostname)
- Konfigurace TCP/IP – automatické nastavení s DHCP/statické nastavení
- Zaslání dávky příkazů
- Aktualizace firmware

3.4.2 Vlastnosti aplikace, požadované technologie a kompatibilita

Stejně jako požadavky na funkcionalitu byly stanoveny také určité požadavky na uspořádání součástí aplikace a použité technologie:

- Aplikace pro systém MS Windows
  - Možnost využití prostředků platformy Microsoft .NET Framework
- 3 komponenty aplikace (služba serveru, služba agenta, aplikace konzole)
- Komunikace prostřednictvím protokolu TCP/IP formou předdefinovaných zpráv ve formátu XML + šifrování přenášených dat
- Souborová místní databáze pro server
- HTTP server a repositář pro distribuci firmware a jiné účely
- Nasazení aplikace formou instalačního balíku Windows Installer
- Kompatibilita aplikací
  - Server + konzole: podpora serverových i desktopových systémů MS Windows XP/2003 a všech novějších verzí k datu
  - Management Agent: identická funkcionalita správy na tenkých klientech se systémem Windows Embedded Standard 2009 i Standard 7

4 Teoretická východiska řešení

Zde budou rozebrány základní teoretická východiska a pojmy důležité pro řešení této práce, zejména pak témata z oblasti informačních a komunikačních technologií. Důraz bude kladen na problematiku tenkých klientů, virtuálních desktopů a jejich systémů, síťovou komunikaci a protokoly relevantní pro výsledné řešení, relační databáze, programovací jazyky a další technologie použité při vývoji aplikace.

4.1 Tenký klient

Termín tenký klient popisuje počítačový systém (případně i pouhou aplikaci – existuje více významů, pro účely této práce však bude chápán ve smyslu hardware), který je sestaven minimalistickým způsobem, tedy obsahuje pouze součásti nutné ke zprostředkování požadované služby uživateli. Hardwarové prostředky tenkých klientů se odvíjí od způsobu implementace infrastruktury, používaného operačního systému a aplikací, oproti obyčejnému počítači PC (tedy tzv. tlustému klientu) jsou zde však sledovány rozdílné cíle – důraz není kladen na maximální výkon (samotné zpracování dat a vykonávání programů může fyzicky probíhat na jiném počítači, například ve virtuálním stroji na serveru), mnohem důležitějšími faktory jsou však vlastnosti jako je nízká spotřeba elektrické energie, omezení hluku, rozměry a zejména spolehlivost (7).

4.1.1 Hardware tenkých klientů

Jak bylo zmíněno výše, cílem hardwarové konfigurace tenkých klientů není maximalizace výkonu, namísto toho je preferována vyšší spolehlivost – důležitá vlastnost pro dosažení vysoké dostupnosti celého řešení IT infrastruktury firmy, ale také klíčový faktor pro prodloužení životního cyklu těchto zařízení (vzhledem k centralizaci a distribuovanému výkonu zde není nutný častý upgrade, jako je tomu u PC). Rizika selhání jsou u tenkých klientů minimalizována zejména absencí jakýchkoliv pohyblivých částí – neobsahují například pevný disk, ventilátory, optické mechaniky atd. Tímto je také eliminován hluk, který zařízení vytváří, a do značné míry snížena spotřeba elektrické energie. Zásadním krokem k dosažení úspory energie je však použití speciálních procesorů určených pro vestavěná zařízení – tyto procesory jsou navrženy s ohledem na spotřebu a produkci tepla tak, aby byly schopny úsporného provozu

v uzavřeném prostředí s pasivním chlazením. V návrhu tenkých klientů je dnes běžnou praxí použití komplexních řešení – základních desek malých rozměrů s již zabudovaným procesorem a integrovaným grafickým čipem včetně pasivního chlazení a s integrovanou zvukovou i síťovou kartou. Mezi takováto řešení patří například Mini-ITX desky osazené procesory Intel Atom, systémy APU společnosti AMD a další. Velikost operační paměti tenkého klienta se odvíjí od operačního systému a aplikací, které budou na zařízení provozovány. Jako místní uložiště dat (v případě tenkého klienta se jedná pouze o systémové úložiště – na zařízení nejsou obvykle ukládány žádná pracovní či uživatelská data) jsou obvykle využívány paměti typu flash, případně modernější jednotky SSD; v některých případech však tenký klient neobsahuje žádné úložiště a jeho operační systém je zaváděn ze síťového zdroje (7).

4.1.2 Softwarové prostředky tenkých klientů

Z charakteristik tenkých klientů vyplývá, že jejich hlavní softwarové prostředky, tedy ty, které jsou prezentovány uživateli, jsou provozovány na jiném stroji. Přesto však musí tenký klient obsahovat alespoň minimální softwarové vybavení, které mu umožní přístup ke vzdálenému zdroji – v základu je tedy vyžadována aplikace, která toto připojení zprostředkuje a operační systém, který umožní běh této aplikace (7).

Operační systémy

Operační systémy pro tenké klienty jsou obdobou běžných desktopových systémů, avšak s omezenou funkcionalitou (ve prospěch minimální velikosti) a často pouze zjednodušeným či omezeným uživatelským rozhraním. Mezi nejpoužívanější systémy patří následující:

- **Microsoft Windows Embedded** – modulární operační systém založený na desktopových systémech *MS Windows* (*NT, XP, 7*) [3].
- **Distribuce systému Linux** – minimalistické distribuce otevřeného operačního systému *Linux* jsou vhodným systémem pro tenké klienty, avšak je zde stále omezená podpora některých služeb.

[3] Systém MS Windows Embedded bude podrobněji rozebrán v samostatné kapitole.

- **Google Android** – v posledních letech rychle se rozšiřující operační systém, určený nejen pro mobilní zařízení, poskytuje služby vhodné i pro použití v tenkých klientech, například nativní klient *VMware View*.
- **Systémy výrobců tenkých klientů –** někteří výrobci tenkých klientů dodávají ke svým zařízením vlastní systém (často založený na systémech *Linux*) (7).

Software vzdálené relace

Existují různé nástroje a technologie pro přístup ke vzdáleným zdrojům a terminálovým relacím, mezi nejběžnější a nejčastěji používané patří například následující:

- **Microsoft Remote Desktop Services (RDS) –** terminálové služby systému *Microsoft Windows Server*, využívají protokol RDP (Remote Desktop Protocol), klient je součástí operačních systémů Windows.
- **VMware View –** virtuální desktopová infrastruktura společnosti VMware, multiplatformní klient *View* umožňuje vzdálené připojení k virtuálním strojům serveru *VMware ESX* pomocí protokolů PCoIP (PC over IP), případně RDP.
- **Citrix XenDesktop** – VDI v podání společnosti Citrix, poskytuje podobné funkce jako *VMware View*, k připojení je využíván protokol ICA (Independent Computing Architecture) (7).

4.2 Systémy Microsoft Windows Embedded

Microsoft Windows Embedded je rodina operačních systémů, určených zejména pro specializovaná a jednoúčelová zařízení. Využívají stejné technologie jako desktopové systémy Windows a umožňují tak vývojářům využívat zde nabyté znalosti a stejné nástroje. Mezi zařízení využívající tyto systémy patří například pokladny, různá specializovaná zařízení na sběr dat, bankomaty, kiosky a veřejné terminály, navigace, tencí klienti, a další. Speciální edice Embedded systémů jsou využívány také v automobilovém průmyslu a dalších oblastech. Všechna zařízení s tímto systémem mohou jednoduše využívat technologie Microsoft a spolupracovat s ostatními systémy a službami (8).

4.2.1 Klíčové vlastnosti

Jak bylo zmíněno výše, systémy *MS Windows Embedded* vychází z desktopových systémů Windows – tím je zajištěna kompatibilita s převážnou většinou běžně používaných aplikací a zároveň koncovým uživatelům poskytnuto důvěrně známé prostředí. Oproti svým desktopovým ekvivalentům (systémům *Windows NT, XP, 7* a *8*) jsou Embedded systémy rozdílné zejména z technického pohledu – prioritou je zde přizpůsobivost cílové platformě, minimalizace velikosti a možnost upravit systémové součásti na míru požadavkům; toho je docíleno rozdělením systémových součástí do volitelných komponent, ze kterých vývojáři sestaví systém pro své zařízení (mohou zvolit jaké komponenty a soubory bude výsledný systém obsahovat a případně přidat své vlastní). Systémy Embedded nejsou určeny k běžnému prodeji – jsou dostupné pouze výrobcům zařízení OEM (Original Equipment Manufacturer) (8).

4.3 Síťová komunikace

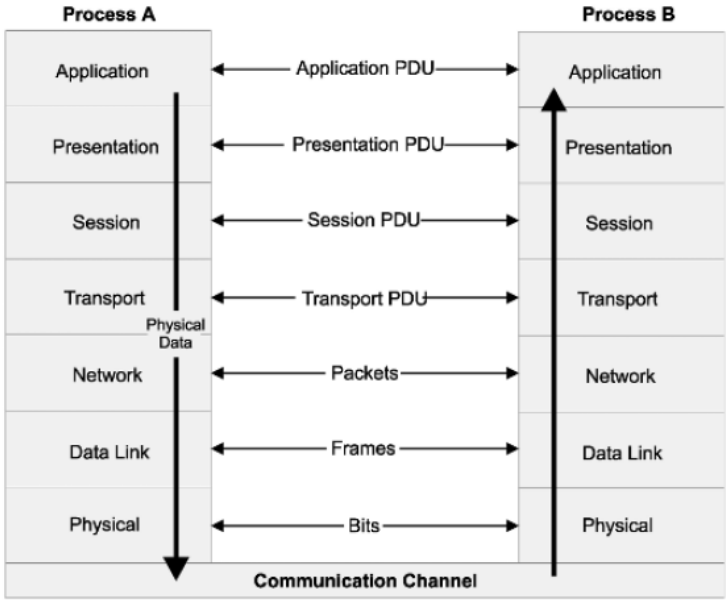
Aplikace navrhovaná v této diplomové práci je z velké části založena na komunikaci mezi jejími aplikačními komponentami prostřednictvím počítačové sítě a protokolu TCP/IP – na tomto místě budou proto rozebrány důležitá témata z oblasti počítačových sítí.

4.3.1 Referenční model ISO/OSI

Referenční model komunikace mezi otevřenými systémy OSI (Open System Interconnection) byl definován v roce 1978 Mezinárodní organizací pro normalizaci (ISO) jako reakce na rostoucí výskyt uzavřených síťových systémů a nekompatibilních proprietárních technologií a následně přijat jako norma ISO 7498. Model ISO/OSI nedefinuje konkrétní technologie, protokoly a přesné způsoby propojení otevřených systémů, ale je zamýšlen spíše jako obecný vzor, model, který má posloužit jako kostra a reference pro další, podrobnější návrh konkrétní síťové architektury.

Model rozčleňuje datovou komunikaci do sedmi hierarchicky uspořádaných vrstev, každou se specifickým účelem a poskytovanými funkcemi či službami. Důležitou základní skutečností je, že každá vrstva N "virtuálně" komunikuje vždy se stejnou protilehlou vrstvou N na druhé straně komunikačního kanálu, nikoliv však

vrstvou vyšší či nižší, tzv. horizontální komunikace. Pravidla komunikace protilehlých vrstev jsou definována pomocí protokolů, určujících jak bude komunikace probíhat a jaký bude formát předávaných dat – PDU (Protocol Data Unit, datová jednotka protokolu specifická pro každou vrstvu, sestává z hlavičky a těla) a jakým způsobem budou ošetřeny chybové stavy. Vertikální komunikace probíhá mezi vrstvou N a její bezprostředně vyšší/nižší vrstvou této strany komunikace, tedy vrstvou N+1 respektive N-1. Hierarchie a interakce jednotlivých vrstev modelu ISO/OSI je znázorněna na následujícím obrázku (9).



Obrázek 1 - Architektura referenčního modelu ISO/OSI (9)

V architektuře modelu OSI využívá každá vrstva služeb nižší vrstvy a zároveň poskytuje své vlastní funkce/služby vrstvě vyšší. Dále bude rozebrána funkcionalita a vlastnosti jednotlivých vrstev ve vzestupném pořadí.

- **Fyzická vrstva** je nejnižší vrstvou hierarchie, zajišťuje přenos jednotlivých bitů po médiu ke druhé straně komunikace. Na druhé straně pak musí být tyto bity korektně přijaty (správné kódování apod.). Fyzická vrstva přijatá data neinterpretuje. Na této úrovni neexistuje možnost adresace.
- **Linková vrstva** využívá služeb fyzické vrstvy pro přijímání a odesílání jednotlivých bitů – může takto přenášet tzv. rámce (frame) – jednotky přenosu

na úrovni linkové vrstvy. Linková vrstva musí dále zajistit rozpoznání jednotlivých rámců a jejich částí, detekci chyb vznikajících na fyzické vrstvě a případné opravy, tok a řízení přístupu k médiu. Linková vrstva dokáže přenášet rámce pouze k sousedním uzlům, tedy adresovat pouze lokálními adresami.

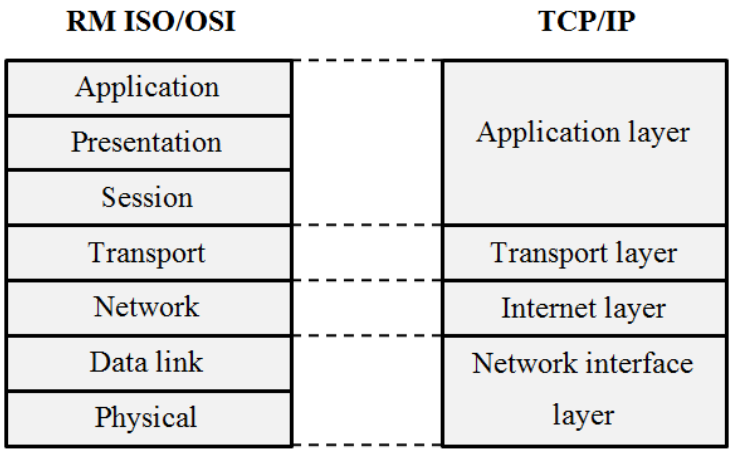
- **Síťová vrstva** umožňuje komunikaci mezi uzly bez přímého spojení, čehož je docíleno směrováním (routing) – mezilehlý uzel (uzly) na cestě mezi odesílatelem a adresátem přepošle přenášený blok – paket – na úrovni síťové vrstvy vhodnou cestou směrem k příjemci. Mimo směrování odpovídá síťová vrstva také za překlad fyzických adres na síťové, vytváření a udržování logického spojení mezi dvěma uzly a rozdělování velkých paketů dat na rámce přenositelné linkovou vrstvou (tyto jsou protilehlou síťovou vrstvou opět spojeny). Adresace na úrovni síťové vrstvy je realizována globálními adresami.
- **Transportní vrstva** je umístěna na rozhraní mezi horními, na aplikaci silně závislými vrstvami a dolními, síťově orientovanými vrstvami. Jejím účelem je poskytnutí služeb nižších vrstev aplikacím požadovaným způsobem – například spolehlivý vs. nespolehlivý přenos (oprava chyb nebo zahození chybných dat, ovlivňuje rychlost a režii) a spojovaný vs. nespojovaný přenos. Na transportní vrstvě se přenáší tzv. datagramy a je již nutné odlišit příjemce v rámci jednoho uzlu – jednotlivé aplikace. Toho je docíleno adresací pomocí portů vedoucích na konkrétní procesy.
- **Relační vrstva** využívá služeb transportní vrstvy, odpovídá za synchronizaci a udržování relace dialogu komunikujících vyšších vrstev (udržování spojení). Adresace v této ani vyšších vrstvách již neexistuje – cílový proces byl již vymezen portem na transportní vrstvě. Jedná se o nejméně vytíženou a často kritizovanou vrstvu modelu ISO/OSI.
- **Prezentační vrstva** zajišťuje převod přijatých dat do podoby, se kterou dokáže nadřazená aplikační vrstva pracovat – může se jednat o konverzi kódování znaků, kompresi/dekompresi dat, změny reprezentace datových struktur apod.
- **Aplikační vrstva** je nejvyšší vrstvou modelu, jejím účelem je poskytnout aplikacím přístup k síti a umožnit tak úlohy jako jsou přenosy souborů, přijímání/odesílání elektronické pošty apod. Služby aplikační vrstvy jsou

výrazně rozsáhlejší než služby ostatních vrstev a obsahují ty části aplikací, které je možné standardizovat (9).

4.3.2 Architektura TCP/IP

Architektura TCP/IP je dnes nejrozšířenější síťovou architekturou. Historie této architektury sahá až do 60. let a souvisí se vznikem počítačové sítě ARPANET ministerstva obrany USA a postupným rozvojem počítačových sítí až po dnešní internet. Základním cílem architektury je poskytnout otevřené řešení propojení sítí na různých přenosových technologiích (internetworking).

Oproti modelu ISO/OSI se architektura TCP/IP značně liší v koncepci a přístupu tvůrců – je již navržena podle reálných, praktických požadavků, řešení vznikají jako jednodušší a postupně se rozšiřují, využívá již existující technologie a provazuje je s vlastním řešením. Tato skutečnost se pak odráží v uspořádání, počtu vrstev a výsledné funkcionalitě, kterou zastávají (9).



Obrázek 2 - Vrstvy architektury TCP/IP (vlastní zpracování dle 9)

Vrstvy architektury TCP/IP a jejich funkce jsou následující (od nejnižší):

- **Vrstva síťového rozhraní** – nejnižší vrstva architektury, nedefinuje přenosové technologie, fyzickou ani linkovou vrstvu, pouze způsob propojení s vyšší síťovou vrstvou.

- **Síťová vrstva –** primárním účelem síťové vrstvy (nazývané také internetová vrstva) je směrování paketů napříč sítěmi – každý paket obsahuje adresační informace pro směrování až k cíli komunikace. Hlavním protokolem této vrstvy je protokol IP (Internet Protocol, podrobněji popsán v další kapitole). Mezi další důležité protokoly definované na této úrovni patří protokol ARP (Address Resolution Protocol, umožňuje překlad IP adres na hardwarové MAC adresy), RARP (Reverse ARP, překlad MAC adres na IP) a ICMP (Internet Control Message Protocol, protokol zasílání řídících a chybových zpráv).
- **Transportní vrstva** – zajišťuje tzv. "host-to-host" komunikaci, zodpovídá za integritu dat přenášených mezi odesílatelem a příjemcem. Na této úrovni rovněž probíhá detekce chyb a korekce. Transportní vrstva již adresuje konkrétní procesy uzlu komunikace pomocí čísla portu. Na této úrovni existují dva transportní protokoly: TCP (Transmission Control Protocol) a UDP (User Datagram Protocol).
- **Aplikační vrstva** – vrstva poskytující aplikačním procesům rozhraní pro komunikaci prostřednictvím TCP/IP. Pro aplikace vyžadující prezentační a relační služby jsou zde implementovány protokoly RPC (Remote Procedure Call, relační služby) a XDR (External Data Representation, prezentační služby). Aplikační vrstva definuje širokou škálu dalších účelově orientovaných protokolů, mezi nejznámější patří např. protokol elektronické pošty SMTP, přenos souborů FTP, připojení ke vzdálené relaci TELNET, atd. (9).

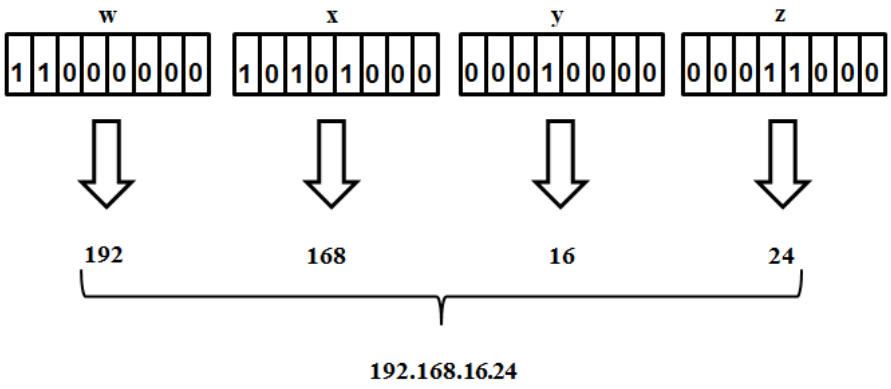
4.3.3 Protokol IP

Internet Protocol (IP) je jádrem celého řešení architektury TCP/IP. Protokol pracuje na úrovni síťové vrstvy a jeho účelem je zajistit přenos paketu (tzv. IP datagramu) mezi uzly, a to i v rámci různých sítí – umožňuje tedy směrování paketů na základě IP adres obsažených v hlavičce každého přenášeného paketu. Samotný protokol IP nezajišťuje spolehlivost přenosu dat ani udržování spojení – poskytovaná přenosová služba je tedy v základu nespojovaná a nespolehlivá, tyto vlastnosti jsou dosaženy až na vyšší transportní vrstvě prostřednictvím protokolů TCP a UDP. Protokol IP je založen na principu přepojování paketů.

Dnes nejpoužívanější verzí protokolu je verze IPv4, využívající 32-bitovou adresaci (2^{32}, tedy přibližně 4.3 mld. adres). Z důvodu neúsporného rozdělení adres a stále rostoucího počtu koncových uzlů se stává celkový počet adres tohoto protokolu nedostatečným a je postupně nahrazován novějším protokolem IPv6 využívajícím 128-bitové adresování (9).

4.3.4 Adresace v TCP/IP (IPv4)

Jak již bylo zmíněno výše, pro adresaci v TCP/IP se využívají 32-bitové adresy. Tato čísla v binárním formátu (1 a 0) jsou optimální pro zpracování počítačem. V zájmu lepší čitelnosti pro člověka bývají jednotlivé bity adresy rozděleny do čtyř oktetů, vzniklá čísla převedena do dekadické podoby a oddělena tečkami (9).



Obrázek 3 - IP adresa (vlastní zpracování dle 9)

Maska sítě a třídy adres (9)

Jednotlivé bity IP adresy mohou reprezentovat buď adresu sítě (prvních N bitů zleva), anebo adresu koncového uzlu (zbývající část adresy). Identifikace části adresy určené pro adresaci sítě nebo koncového uzlu je možná pomocí tzv. masky sítě – čísla ve stejném formátu jako samotná IP adresa, kde každý bit s hodnotou 1 znamená příslušnost k adrese sítě a hodnoty 0 adresu uzlu. Alternativou je tzv. prefix CIDR (Class Inter-Domain Routing), číslo udávající počet bitů spadajících do adresy sítě (tedy například 192.168.1.16/24, kde první 3 čísla tvoří adresu sítě). Přestože počet bitů adresy sítě a uzlu je volitelný, standardní a doporučené je následující rozdělení:

Třída	Maska	Počet sítí	Počet uzlů v síti
A	255.0.0.0	256	16 777 216
B	255.255.0.0	65 536	65 536
C	255.255.255.0	16 777 216	256

Tabulka 2 - Přehled základních tříd IP adres (vlastní zpracování dle 9)

4.3.5 Protokoly transportní vrstvy

Zde budou popsány protokoly transportní vrstvy architektury TCP/IP, umožňující přizpůsobení vlastností přenosových služeb protokolu IP potřebám aplikace.

TCP - Transmission Control Protocol (10)

Protokol TCP realizuje na úrovni transportní vrstvy spojovanou službu se zaručením spolehlivého doručení zprávy příjemci. Mezi dvěma protilehlými procesy tedy vzniká spojení, fungující na principu proudu bytů (stream), které je plně duplexní (současný obousměrný přenos dat) a z pohledu procesu spolehlivé – data jsou příjemci doručena tak, jak je odesílatel vyslal (bez duplicit/výpadků a ve správném pořadí). Pro efektivní využití komunikačního kanálu je zaveden buffer – teprve po jeho naplnění jsou data vložena do datagramu a předána k odeslání, přijímající strana pak tato data postupně předává procesu.

Spolehlivost v TCP je zajištěna metodou kladného potvrzování – příjemce musí informovat odesílatele o přijetí dat s daným sekvenčním číslem (na chybné přijetí se nereaguje), pokud tak neučiní v časovém limitu, jsou data odeslána znovu. Pro vyšší efektivitu je zavedeno kontinuální kladné prověřování, kdy dalších N segmentů může být odesláno nezávisle na obdržení potvrzení o přijetí segmentu předchozího – tzv. metoda okénka (sliding window), jehož velikost se řídí dostupností volného místa v bufferu příjemce, který ji avizuje při každém potvrzení. Je-li některý segment přenesen chybně, je přenesen znovu, včetně již přenesených segmentů po něm následujících (opakování s návratem).

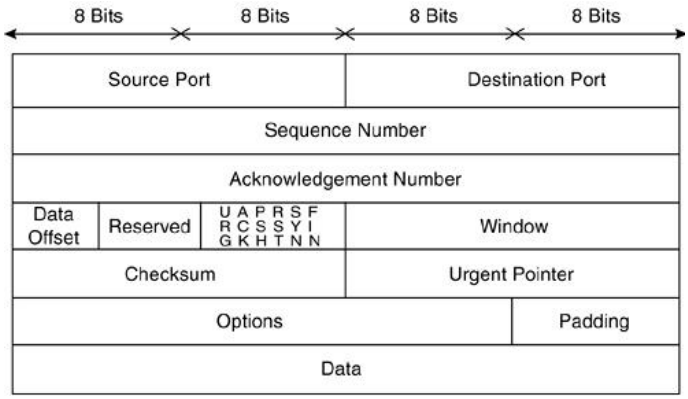
Spojení TCP je realizováno pomocí tří kroků, tzv. **Three-way handshake**:

- Aktivní proces vytvoří TCP socket a zašle naslouchajícímu procesu datagram s příznakem SYN a novým sekvenčním číslem.

- Naslouchající proces ověří, že lze socket přijmout a zašle zpět datagram s příznaky SYN a ACK s vlastním sekvenčním číslem, a přijatým sekvenčním číslem druhé strany zvýšeným o 1 v poli ACK.
- Aktivní proces po přijetí datagramu potvrzení synchronizace zašle zpět datagram potvrzující jeho přijetí s nastaveným příznakem ACK.

TCP spojení může být ukončeno dvěma způsoby:

- **Uzavření spojení** – strana ukončující spojení zašle druhé straně datagram s nastaveným příznakem FIN, přičemž stále přijímá data, dokud druhá strana neodpoví se stejným příznakem – následně je informována vyšší vrstva, uvolněny zdroje a spojení ukončeno. Přijímající strana po obdržení ukončovacího datagramu odešle zbývající připravená data, odešle potvrzení s příznakem FIN a ukončí spojení stejným způsobem.
- **Reset spojení** – pokud nejsou zasílané datagramy v časovém limitu potvrzeny, provede proces několik pokusů o opětovné doručení. Pokud ani po opakovaných pokusech nedojde k potvrzení, je odeslán datagram s příznakem RST – pokud jej druhá strana příjme, je nucena zrušit spojení. TCP následně informuje vyšší vrstvu, ukončí spojení a uvolní zdroje.



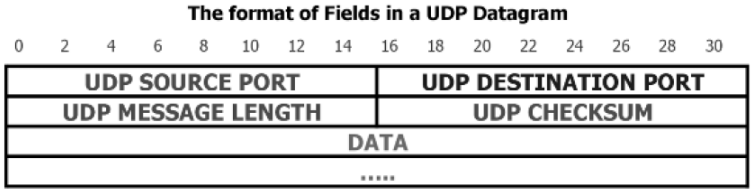
Obrázek 4 - Formát TCP segmentu (9)

UDP - User Datagram Protocol (10)

Protokol UDP poskytuje na úrovni transportní vrstvy nespojovanou přenosovou službu s nezaručenou spolehlivostí. Pro zasílání dat není vytvářeno spojení a protokol

neošetřuje potvrzování, opětovné zasílání ani pořadí přijetí dat. Na rozdíl od protokolu TCP je přenos blokový – bloky dat jsou přímo vkládány do datagramu a odesílány. UDP rovněž zajišťuje identifikaci cílového procesu v rámci koncového uzlu doručováním na konkrétní port.

Hlavní výhodou protokolu UDP je velmi malá režie, vzhledem k absenci synchronizačních parametrů, sekvenčních čísel, časovačů, potvrzování a opětovného odesílání (typických pro spolehlivý přenos TCP), protokol je tedy jednoduchý a rychlý. Hlavní využití protokolu UDP spočívá v komunikaci s více koncovými uzly – broadcast/multicast, a zejména v aplikacích vyžadujících rychlé přenosy dat v reálném čase s přiměřenou chybovostí (streaming videa, přenos telefonních hovorů, apod.).



Obrázek 5 - Formát UDP datagramu (9)

4.3.6 Model klient-server

Aplikace komunikující pomocí protokolů TCP/IP jsou založeny na modelu klient-server. V tomto modelu je lze rozdělit na aplikace poskytující určité služby – servery, a aplikace využívající tyto služby – klienti.

Server (10)

V TCP/IP neexistuje mechanismus pro vytvoření procesu při příchodu zprávy – z tohoto důvodu musí existovat proces, který umožní pasivně čekat na požadavek klienta, na jehož základě vyvolá určitou akci (zpravidla odpověď). Server je aplikací, která naslouchá na určitém portu a poskytuje připojujícím se klientům určitou službu.

Server má obvykle přístup k určitým zdrojům (např. soubory či databáze) a privilegia pro operace nad těmito zdroji, avšak není vhodné, aby každý klient mohl k těmto zdrojům přistupovat – aplikace serveru by tedy měla mimo samotné poskytování služby zabezpečit i následující body:

- Ověřování uživatele – přihlášení jménem a heslem
- Autorizace – oprávnění uživatele k práci se zdrojem
- Bezpečnost dat – data nelze neúmyslně zničit či modifikovat
- Ochrana zdrojů – zabezpečení před zneužitím skrz komunikační síť

Servery mohou poskytovat služby spojované (komunikace prostřednictvím TCP protokolu), anebo služby nespojované (UDP protokol), rozhodnutí o typu komunikace se odvíjí od požadované funkcionality serveru.

Jestliže server neudržuje v paměti žádnou informaci o spojení klient-server, nazývá se *bezstavový* – tyto servery jsou obvykle aplikačně jednodušší, avšak vyžadují větší množství režijních dat. *Stavový* server udržuje informaci o spojení a parametry relace v paměti a umožňuje tak efektivnější komunikaci bez nutnosti opakovaně přenášet některá data.

Server, který má obsloužit více klientů zároveň – tzv. *konkurentní* server, musí být navržen tak, aby umožňoval současné zpracování více požadavků – toho může být dosaženo různými prostředky, od skutečného paralelního zpracování a sdílení času, až po vytváření separátních procesů či rozdělení aplikace do samostatně pracujících vláken. Tato vlastnost je typická pro většinu používaných serverů. Servery pracující pouze s jedním požadavkem v čase se nazývají *iterativní*.

Klient (10)

Klient je aplikací iniciující komunikaci a požadující od serveru poskytnutí určité služby. Oproti serveru se jedná o jednodušší aplikaci (protože obvykle nekomunikuje s více servery zároveň), která se připojí k naslouchajícímu serveru a následně s ním komunikuje. Obvykle nevyžaduje speciální privilegia ani nepracuje s privilegovanými porty.

Standardní klient komunikuje s tzv. well-known portem – jedním z rezervovaných portů, jako je např. elektronická pošta (25), FTP (20/21), HTTP (80) apod. Ostatní klientské aplikace jsou považovány za *nestandardní*. Aplikace, ve kterých je role klienta a serveru předem stanovena se nazývají *nesymetrické*, pokud však obě strany mohou zastávat roli klienta i serveru, jedná se o aplikaci *symetrickou*.

4.3.7 Rozhraní sockets

Rozhraní Socket interface (sockets) je jedním ze standardních prostředků komunikace mezi aplikačními procesy prostřednictvím počítačové sítě. Je založeno na obecném přístupu, aplikovatelném na protokolech TCP/IP i v jiných případech.

Socket je koncovým bodem komunikace (communication endpoint), jedná se o systémovou datovou strukturu, která obsahuje údaje vyžadované pro realizaci komunikace. Koncový uzel komunikace pro protokoly TCP/IP je definován pomocí IP adresy a čísla portu, je tedy možné jednoznačně identifikovat konkrétní stroj i na něm běžící proces a to kdekoliv v internetu.

Struktura socketu není přímo přístupná, aplikační procesy však na tuto strukturu mohou odkazovat pomocí tzv. *deskriptorů* (socket descriptor). Deskriptor je celé číslo, index v rámci tabulky deskriptorů, která obsahuje odkazy nejen na sockety, ale i na soubory. Při vytvoření socketu je vytvořena pouze struktura se základními informacemi a je postupně vyplňována daty na základě požadavků aplikace. Vytvořený socket může být *pasivní* – servery, anebo *aktivní* – klienti (10).

4.4 Databázové systémy

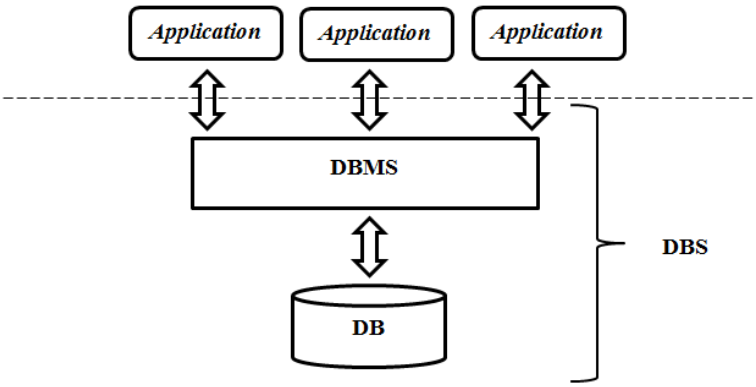
Databázové systémy jsou systémy umožňující efektivní ukládání velkých objemů dat různého charakteru, a následnou manipulaci s těmito daty. Počátky databázových systémů sahají až do 70. let – cílem jejich vzniku bylo nahradit dřívější neefektivní způsob práce s daty, kdy veškerá manipulace, ukládání, výpočty a strukturování dat byly zahrnuty v samotné aplikaci, což vedlo k nemožnosti přístupu více uživatelů, náročným změnám a praktické neproveditelnosti jakékoliv standardizace. Na rozdíl od tohoto přístupu, databázové systémy oddělují samotná data od logiky jejich zpracování. Takovýto přístup vede k zajištění fyzické a logické nezávislosti dat (tj. možnosti úprav dat a jejich struktury bez nutnosti zásahů do vlastní aplikace), umožňuje víceuživatelský přístup k datům, sjednocuje datové soubory, odstraňuje duplicity a umožňuje vytvořit standardní rozhraní pro přístup k databázi (11).

Každý databázový systém obecně sestává z následujících komponent:

- **Báze dat –** centrální datová struktura, DB (Database), obsahuje uložená data
- **Systém řízení báze dat –** DBMS (Database Management System) je aplikační systém, sloužící ke správě báze dat, poskytuje zejména následující služby:

- ***Definice dat –*** definuje datové struktury a organizuje datové soubory
- ***Manipulace s daty –*** funkce vkládání nových dat, úpravy existujících a aktualizace datových souborů
- ***Zobrazení dat*** – umožňuje přístup k datům, poskytuje metody výběru, zobrazování a prezentování dat
- ***Integrita dat –*** zajišťuje správnost dat a omezuje duplicity
- ***Programovací jazyk*** – jazyk zprostředkovávající služby vlastní aplikaci

- **Aplikace –** program využívající služeb databázového systému (DBS = DB + DBMS) pro ukládání a čtení dat, která používá při vykonávání své činnosti (11).



Obrázek 6 – Obecné schéma databázového systému (vlastní zpracování dle 11)

Podle rozmístění komponent DBS mezi počítačové systémy, na kterých jsou provozovány, lze rozlišovat následující architektury databázových systémů:

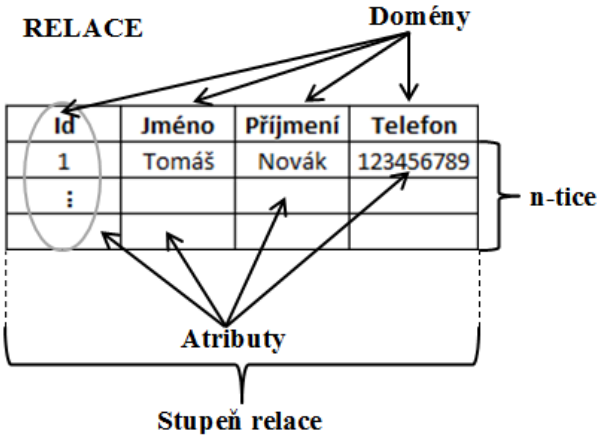
- **Centralizovaný systém** – všechny komponenty (tedy DB, DBMS i aplikace) jsou umístěny na jednom hostitelském počítači, ke kterému se připojují uživatelé prostřednictvím terminálu, webu, apod. Toto řešení umožňuje současnou práci více uživatelů a centrální zabezpečení dat. Je vyžadován vysoký výkon serveru a náročná údržba.
- **Architektura file-server** – DBMS a aplikace běží na klientském počítači, databáze je umístěna na souborovém serveru. Výhodou je snížení požadavků na výkon serveru, zejména při větším počtu uživatelů, a obecně nižší náklady. Limitujícím faktorem je pak ovšem výkon klientských počítačů a vysoké zatížení počítačové sítě přenosy velkých souborů.

- **Architektura klient-server** – nejčastěji používaná architektura, typickým scénářem je databázový server provozující DB+DBMS a klientský počítač, případně aplikační server, provozující odděleně aplikaci. Komunikace probíhá na bázi dotaz – odpověď. Zásadní výhodou modelu je rozdělení výkonu mezi dva systémy a snížení síťového provozu oproti architektuře file-server (11).

4.4.1 Relační model dat

Relační databáze jsou založeny na relačním datovém modelu. Tento model byl formulován v roce 1969 a dnes patří k nejpoužívanějším datovým modelům. Relační datový model je založen na teorii množin a predikátové logice, definuje způsob reprezentace dat, integritních omezení a operací s daty. Základem relačního modelu je množina, tzv. **relace**, obsahující data a odpovídající určité struktuře, tzv. **schéma relace**; sestává z názvu relace, domén + atributů (tj. sloupců) a popisu domén (integritních omezení). Relace lze interpretovat jako tabulky dat, sestávající ze sloupců a řádků, a takto lze i s daty pracovat. Existují následující typy relací:

- **Pojmenované relace**
  - Reálné tabulky
  - Virtuální tabulky (pohledy)
  - Snapshot (fyzicky uložený pohled)
  - Dočasná tabulka
- **Nepojmenované relace -** výsledky dotazů a mezivýsledky (11)



Obrázek 7 - Relace (vlastní zpracování dle 11)

Vztahy relací (11)

V relačním datovém modelu figurují vztahy mezi jednotlivými relacemi, propojení dvou relací je realizováno tzv. **relačním klíčem**. Jedná se o položku, která je shodná v obou tabulkách, a to z pohledu názvu, datového typu, velikosti i obsahu, a zároveň jednoznačně identifikuje daný objekt. Součástí propojení je rovněž určení **relačního poměru**, tj. kolik vět v relaci B odpovídá větám z relace A:

- **Poměr 1:1** – jedné větě relace A odpovídá jedna věta relace B.
- **Poměr 1:N** – jedné větě relace A odpovídá více vět relace B.
- **Poměr M:N –** M větám relace A odpovídá N vět relace B. Tento poměr není v rámci databáze realizovatelný, proto bývá transformován na oboustranný poměr 1:N prostřednictvím spojovací tabulky – tzv. **dekompozice M:N**.

Integrita dat (11)

Integrita dat v databázi znamená jejich konzistentnost, korektnost a platnost. Je to důležitý aspekt návrhu databáze a klíč k zajištění bezchybné práce s daty. K tomuto účelu slouží tzv. **integritní omezení** – pravidla a nástroje, jejichž cílem je zajistit integritu databáze na následujících úrovních:

- **Entitní integrita** – integrita na úrovni tabulky, je zajištěna tzv. primárním klíčem. **Primární klíč** (PK) je množinou atributů relace, která je jednoznačná (identifikace položky) a minimální. Jedná se o základní prostředek adresace vět, a proto nesmí v žádném atributu domény PK chybět hodnota. **Kandidátní klíč** je jedna z domén, ve které jsou všechny atributy unikátní. Jeden z kandidátních klíčů je následně zvolen jako primární klíč tabulky.
- **Doménová integrita** – integrita na úrovni pole, zajišťuje platnost hodnot atributů spadajících pod určitou doménu.
- **Referenční integrita –** integrita na úrovni vztahů, zajišťuje spolehlivost a soulad vztahu mezi dvěma tabulkami. **Cizí klíč** je atribut, pro který existuje věta jiné relace s primárním klíčem stejné hodnoty. Cizí klíč a odpovídající primární klíč druhé relace musí být definovány nad stejnou doménou a nesmí se vyskytovat žádný cizí klíč bez korespondujícího primárního klíče.

4.4.2 Jazyk SQL

SQL (Structured Query Language) je speciální, účelově zaměřený programovací jazyk, určený pro práci s daty v relačních systémech řízení báze dat (RDBMS). Jazyk byl vytvořen v roce 1970 společností IBM pod původním názvem SEQUEL (Structured English Query Language) a později přejmenován na dnešní název SQL. Jako standard byl poprvé přijat v roce 1986 (SQL86) a později ve verzi opravující určité nedostatky SQL92 (11).

SQL je deklarativní, neprocedurální programovací jazyk, založený na množinovém přístupu k datům a dotazování pomocí požadavků založených na relační algebře a n-ticovém relačním kalkulu. Jazyk SQL lze rozdělit na tyto základní oblasti:

- **Data Definition Language (DDL)** – jazyk definice dat, umožňuje vytváření datových struktur a schématu databáze. Mezi základní příkazy DDL patří *CREATE, DROP* a *ALTER*.
- **Data Manipulation Language (DML)** – jazyk manipulace s daty, umožňuje vkládání, mazání a aktualizaci dat v databázi. DML zahrnuje například příkazy *SELECT, INSERT, UPDATE* a *DELETE*.
- **Data Control Language (DCL)** – jazyk řízení dat, umožňuje spravovat přístup uživatelů k datům v databázi na základě příkazů *GRANT*, anebo *REVOKE* (11).

Datové typy v SQL

Jazyk SQL podporuje práci s mnoha typy dat v rámci tabulek relační databáze. Přehled základních podporovaných datových typů je uveden v následující tabulce.

Datový typ	Popis
INT	Celé číslo, rozsah -2 147 483 648 až 2 147 483 647
SMALLINT	Celé číslo, rozsah -32 768 až 32 767
FLOAT	Číslo s plovoucí desetinnou čárkou
CHAR (N)	Řetězec fixní délky s N znaky (max. 255)
VARCHAR (N)	Řetězec variabilní délky s N znaky (max. 255)
DECIMAL (P)	Desetinné číslo s P platnými číslicemi
DECIMAL (P, D)	Desetinné číslo s P platnými číslicemi a D desetinnými místy
DATETIME	Datum a čas ve formátu YYYY-MM-DD HH:MM:SS
MONEY	Peněžní částka
BLOB	Binary Large Object, typ velkých binárních objektů

Tabulka 3 - Datové typy jazyka SQL (vlastní zpracování dle 11)

Procedurální rozšíření

Moderní databázové systémy poskytují mimo základní jazykové elementy SQL a datové typy také pokročilé metody procedurálních programovacích jazyků – ty jsou zajištěny prostřednictvím tzv. procedurálních rozšíření. Tato rozšíření umožňují používat v databázovém kódu procedurální funkcionalitu, tedy prvky jako jsou řídící programové struktury (podmínky, cykly, skoky), strukturované a speciální datové typy, deklarace vlastních proměnných, procedur a funkcí, transakce, kurzory, triggery atd. Mezi nejznámější systémy implementující procedurální rozšíření patří například databázový systém *Microsoft SQL Server* a proprietární rozšíření Transact-SQL, nebo systém *Oracle Database* s rozšířením PL/SQL (11).

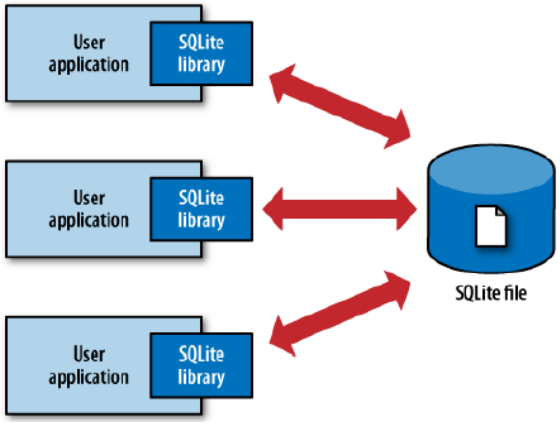
4.4.3 Souborová databáze, relační databázový systém SQLite

Souborové databáze jsou databáze uložené přímo fyzicky v souboru, k jehož obsahu lze s použitím odpovídajícího programového vybavení (speciální aplikace, programová knihovna) přistupovat a pracovat s ním. Výhodou těchto databází je zejména jejich kompaktnost, jednoduchost a nízké náklady, jsou však méně praktické pro distribuované a víceuživatelské scénáře (nevýhoda velkého toku dat typického pro architekturu file-server) a obvykle nedosahují výkonu plnohodnotného databázového stroje. Souborové databáze nachází uplatnění zejména jako podpůrné lokální databáze komplexních aplikací, repositáře konfiguračních dat, ale i jako databáze webových systémů pro správu dat (12).

Databázový systém SQLite

SQLite je kompaktní relační databázový systém založený na jednoduché souborové databázi a malé programové knihovně, dostupný zdarma pod licencí Public domain. Hlavním aspektem tohoto systému je začlenění celého databázového řešení do vlastní aplikace formou reference aplikační knihovny (není tedy vyžadován separátní proces databázového serveru) a zajištění minimalistického, avšak plně funkčního databázového stroje pro potřeby daného programu, bez nutnosti jakékoliv instalace a konfigurace uživatelem při nasazení aplikace. Systém SQLite je plně transakční a splňuje pravidla ACID (Atomic Consistent Isolated Durable), zajišťuje tedy konzistenci

databáze i v případě selhání programu. Díky minimální velikosti knihovny (základní sestavení od 350kB) a malým požadavkům na operační paměť je databázový systém SQLite vhodný k provozu na jakémkoliv stroji, včetně mobilních zařízení (12).



Obrázek 8 - Architektura databázového systému SQLite (12)

4.5 Značkovací jazyk XML

XML (Extensible Markup Language) je obecný značkovací jazyk a standard zpracování dokumentů, vyvinutý a obecně doporučovaný konsorciem W3C (World Wide Web Consortium). Vychází ze staršího, velmi komplexního jazyka SGML (Standard Generalized Markup Language) a je předurčen zejména pro scénáře s dynamicky generovaným obsahem, serializaci dat a výměnu dat mezi aplikacemi. První pracovní návrh jazyka XML byl dokončen v roce 1996 a v roce 1998 byla verze XML 1.0 přijata jako standard W3C, přičemž aktuální verzí jazyka je verze 1.1. Dnes je jazyk XML hojně využíván v mnoha internetových aplikacích, ale díky své univerzálnosti a přehlednosti i kdekoliv jinde a to nejen jako formát dokumentů, ale také formát vhodný pro ukládání komplexních datových struktur, konfigurací a dalších strukturovaných dat. Důležitým aspektem tohoto jazyka je, že v základu nedefinuje žádné značky, ani definice datových struktur a vzhledu – jejich vytvoření a aplikace je ponechána čistě na vývojáři. Dokumenty XML mají určitá závazná pravidla, která musí být při jejich vytváření dodržena (budou popsána v další části kapitoly). Procesní zpracování dat v XML bývá prováděno pomocí tzv. parseru, tj. programu syntaktické

analýzy, a to buď typu DOM (Document Object Model, práce s XML stromem), anebo SAX (Simple API for XML – zpracování dat založené na událostech) (13).

4.5.1 Dokument XML

Každý dokument XML se skládá z elementů. Element je tvořen počáteční a ukončovací značkou a vlastním obsahem, kterým může být text, anebo vnořená struktura dalších elementů. Jednoduchý element je zobrazen v následujícím příkladu.

```
<polozka vlastnost="hodnota">data</polozka>
```

První položkou XML dokumentu je deklarace, tj. značka identifikující soubor jako dokument XML, specifikující použitou verzi jazyka, kódování a případný odkaz na externí DTD schéma pro účely validace. Jednoduchý příklad deklarace je následující:

```
<?xml version="1.0" encoding="utf-8" ?>
```

Druhou položkou, která je však volitelná a v XML dokumentu se nemusí vyskytovat vůbec, je propojení s XSL šablonou (tyto šablony budou popsány dále).

```
<?xsl-stylesheet type="text/xsl" href="./sablona.xsl" ?>
```

Další položkou je již samotný kořenový element dokumentu, obsahující vlastní obsah souboru, strukturu dalších elementů a dat (příklad elementu uvedený výše) (13).

Základní pravidla zápisu v jazyce XML (13)

Jazyk XML uplatňuje pro definici datových struktur přísnější pravidla než jiné značkovací jazyky (např. HTML), proto je nutné zohlednit následující pravidla:

- **Citlivost na velikost písma** – elementy jazyka XML jsou vždy "case-sensitive".
- **Každý neprázdný element musí mít počáteční i koncovou značku** – pro každý neukončený element generuje XML parser chybu.
- **Každý prázdný element musí obsahovat ukončovací lomítko** – element bez obsahu, vyjádřený jednou značkou, musí vždy obsahovat lomítko před koncovým znakem (např. `<polozka />`).
- **Hodnoty atributů musí být v uvozovkách** – hodnota atributu bez ohraničení uvozovkami (platí i pro číselné hodnoty) je považována za syntaktickou chybu.

- **Vnořené elementy se nesmí křížit** – každý vnořený element musí být zakončen dříve, než je zakončen element jemu bezprostředně nadřazený.

4.5.2 Definice DTD

Definice typu dokumentu DTD (Document Type Definition) je soubor pravidel, vymezující množinu XML dokumentů, spadající pod daný typ. Struktura typu je popsána formou uspořádaného seznamu elementů, jejich atributů a přípustných datových typů obsahu jednotlivých elementů. DTD není souborem jazyka XML, pouze je s dokumentem propojen v rámci deklarace, za účelem validace struktury a obsahu vlastního dokumentu. Příklad jednoduché definice je uveden dále (13).

```
<!ELEMENT zprava (od, komu, text)>
<!ELEMENT od (#PCDATA)>
<!ELEMENT komu (#PCDATA)>
<!ELEMENT text (#PCDATA)>
```

4.5.3 Rozšiřitelný stylovací jazyk XSL

XSL (Extensible Stylesheet Language) je rodina jazyků, umožňující transformování a prezentaci XML dokumentů na základě předdefinovaných pravidel. Tyto jazyky byly vytvořeny skupinou W3C ve snaze poskytnout jazyk sémantiky a specifikace stylu pro XML a poprvé publikovány v roce 1998. Jazyky se dělí do tří účelových kategorií (13).

XSL Transformace (13)

Jazyk XSL transformací (XSLT) je nejdůležitější částí XSL. Jeho účelem je transformace XML dokumentu na jiný typ dokumentu, vhodnější pro prezentaci ve webovém prohlížeči, například HTML. Tato transformace je obvykle prováděna převodem XML elementů na elementy HTML s možností přidání/odebrání elementů, atributů, změny jejich pořadí, provádění testů nad daty a podmíněného formátování, přidání dalšího obsahu atd. XSLT využívá pro navigaci XML stromu jazyk XPath.

Šablony jazyka XSL jsou samy dokumentem XML, odpovídají tedy typické struktuře dokumentu XML. Kořenový element šablony je vždy nazván `xsl:stylesheet` a obsahuje element `xsl:template`, který pomocí atributu `match` mapuje šablonu na

kořenový element transformovaného dokumentu. Uvnitř tohoto elementu již může být definován samotný výstupní kód transformace, a to pomocí textové definice dokumentu HTML (jehož vzhled může být dále libovolně upraven, např. připojenou šablonou CSS) a speciálních XSL elementů. Hlavní elementy pro řízení transformace jsou následující:

- **xsl:value-of** – vložení obsahu XML elementu do výstupu
- **xsl:for-each** – výběr všech XML elementů vybraného uzlu
- **xsl:sort** – seřazení položek v rámci výběru xsl:for-each
- **xsl:if** – podmínka testující platnost výrazu

XSL Formátovací objekty (13)

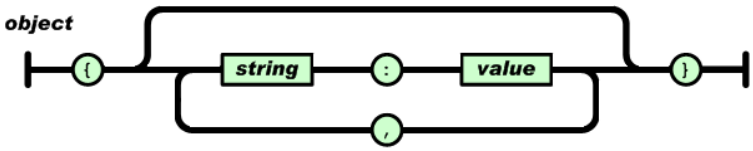
Formátovací objekty XSL (XSL-FO) popisují formátování dat XML dokumentu pro různé typy výstupu, například obrazovku, tisk a další média. Tato technologie poskytuje komplexní metody formátování výstupu a v kombinaci s XSLT je využívána například pro generování tiskových sestav ve formátu PDF. Formátovací objekty jsou založeny na jazyku XML a jsou standardem W3C.

XPath – Jazyk XML Path (13)

XPath je jazyk určený pro navigaci v elementech XML dokumentu prostřednictvím výrazů, umožňuje vybírat individuální elementy a pracovat s jejich hodnotami a atributy. XPath je také základem transformací XSLT a poskytuje množství vestavěných funkcí pro textové a numerické hodnoty, práci s datem a časem a podporu operátorů. Jazyk XPath definuje elementy XML dokumentu jako uzly, mezi nimiž mohou figurovat hierarchické vztahy – rodič (element obsahující další elementy), potomek (element uvnitř vyššího elementu), sourozenec (element sousedící s jiným elementem v rámci stejného nadřazeného elementu), a další. Selekce je prováděna pomocí názvu uzlu (elementu) rozšířeného o operátory selekce, sestávající z lomítek, teček, případně znaku @, anebo bez operátoru. Adresace je také možná po tzv. osách, kdy je cesta specifikována relativně, např. aktuální uzel, předchozí sourozenec, rodič, apod. XPath podporuje také základní aritmetické a logické operátory.

4.6 Datový formát JSON

Formát JSON (JavaScript Object Notation) je jednoduchý, odlehčený datový formát. Hlavním aspektem tohoto formátu je jednoduchá čitelnost a zapisovatelnost člověkem, snadná analýza a univerzální strojové generování i zpracování, což jej předurčuje jako ideální formát pro ukládání jednoduchých strukturovaných dat, například aplikačních konfigurací. Formát je založen na podmnožině programovacího jazyka JavaScript a využívá konvenci programovacích jazyků rodiny C. JSON je založen na dvou univerzálních datových strukturách, dostupných ve všech moderních programovacích jazycích, a to kolekci páru název-hodnota (typicky v programovacím jazyce dostupné jako objekt, záznam, struktura či asociativní pole) a tříděném seznamu (typicky pole, vektor či seznam). Reprezentace objektu ve formátu JSON je znázorněna na následujícím obrázku (14).



Obrázek 9 - Grafická reprezentace objektu JSON (14)

4.7 Microsoft .NET Framework

Platforma *Microsoft .NET Framework* je technologií společnosti Microsoft, určenou pro operační systémy *Windows*. Jejím primárním účelem je poskytnout prostředky pro efektivní vývoj funkčně bohatých aplikací, knihoven a webových služeb a zajistit rychlý a bezpečný běh programového kódu ve společném prostředí s pokročilou správou paměti. První oficiální verze platformy *.NET Framework 1.0* byla uvedena v roce 2002 spolu s balíkem vývojářských nástrojů *Visual Studio .NET*. Platforma pak byla dále rozvíjena až po dnešní verzi 4.5, vydanou v roce 2012.

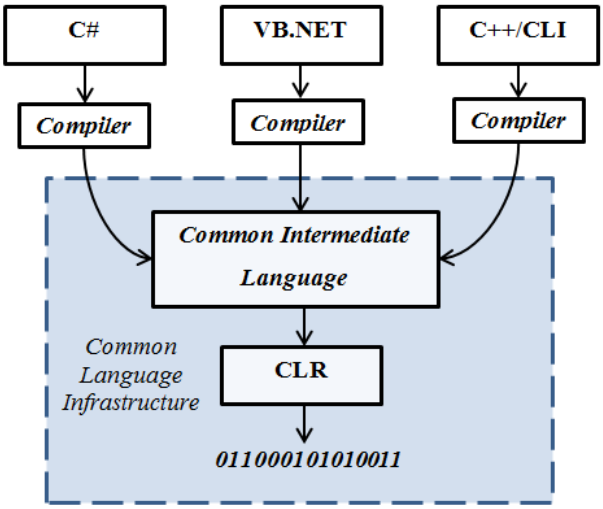
Hlavním přínosem platformy .NET je zejména jazyková nezávislost vývoje aplikací (všechny podporované jazyky jsou překládány do společného jazyka), rozsáhlá knihovna tříd a předdefinovaná standardní funkcionalita, pokročilé služby řízení paměti a bezpečný běh aplikace (15).

4.7.1 Architektura

Platforma *.NET Framework* sestává ze dvou základních systémových komponent – společného jazykového modulu CLR a rozsáhlé knihovny tříd FCL.

CLR (16)

Společný jazykový modul CLR (Common Language Runtime) je jádrem celé platformy .NET. Je to virtuální stroj, zajišťující prostředí pro řízený běh aplikací. CLR poskytuje aplikacím služby správy paměti, kontrolu kompatibility datových typů, práci s více vlákny, sledování výjimek apod. CLR spadá pod tzv. společnou jazykovou infrastrukturu CLI (Common Language Infrastructure), která pracuje s výstupem kompilátorů podporovaných .NET jazyků, neutrálním společným jazykem CIL (Common Intermediate Language). Tento neutrální kód je za běhu překládán pomocí CLR (závislého na cílové platformě) na strojový kód dané architektury a vykonáván.



Obrázek 10 - CLR v architektuře .NET (vlastní zpracování dle 16)

Knihovna tříd (16)

Knihovna tříd .NET Framework Class Library je rozsáhlou kolekcí tříd, rozhraní a datových typů, umožňující přístup k základním i pokročilým systémovým funkcím. Jsou zde zahrnuty již vytvořené funkční prostředky pro práci se soubory, textem, grafikou, databázemi, XML, sítí, apod., které mohou vývojáři využívat ve své aplikaci a zefektivnit tak proces vývoje. Knihovna tříd je hierarchicky rozčleněna na jmenné

prostory, podle funkcionality, která je v dané oblasti zahrnuta. Knihovnu lze také rozdělit na základní knihovnu BCL (Base Class Library, podmnožina celé knihovny obsahující základní komponenty CLR a funkcionalitu dostupnou pro všechny podporované jazyky) a plnou knihovnu FCL (Framework Class Library, obsahuje veškerou dostupnou funkcionalitu pro danou verzi platformy .NET).

.NET Base Class Library (BCL)

System.Web		System.Windows.Forms	
Services *Description* *Discovery* *Protocols*	UI *HTMLControls* *WebControls*	Design	ComponentModel
		System.Drawing	
Caching	Security	Drawing2D	Printing
Configuration	SessionState	Imaging	Text
System.Data		**System.Xml**	
OleDB	SqlClient	XSLT	Serialization
Common	SqlTypes	Xpath	
System			
Collections	IO	Security	Runtime
Configuration	Net	ServiceProcess	InteropServices
Diagnostics	Reflection	Text	Remoting
Globalization	Resources	Threading	Serialization

Tabulka 4 - Přehled základní knihovny BCL (vlastní zpracování dle 16)

4.7.2 Podporované programovací jazyky CLI

Platforma .NET Framework a její aplikační infrastruktura CLI (Common Language Infrastructure) podporuje všechny programovací jazyky, jejichž kód je kompilován do společného neutrálního jazyka. Zásadní výhodou tohoto řešení je zaměnitelnost jazyků a bezproblémová integrace programových komponent vytvořených v různých, podporovaných jazycích.

Platforma .NET je dnes již velmi rozšířená a existuje tedy velké množství CLI podporovaných jazyků. Mezi nejznámější patří tyto programovací jazyky:

- **C#** - nejrozšířenější jazyk platformy .NET (bude dále podrobněji popsán).
- **Visual Basic .NET** – modernizovaný jazyk Visual Basic s podporou objektového programování, CLR a knihovny tříd .NET.
- **C++/CLI** – jazyk C++ s podporou CLR, nahradil původní .NET řízený C++.
- **F# -** funkcionální programovací jazyk platformy .NET (15).

4.7.3 Programovací jazyk C#

C# je moderní, objektově orientovaný programovací jazyk se silnou typovou bezpečností, charakteristický syntaxí a základními rysy programovacích jazyků rodiny C, avšak s přístupem k programování velmi blízkým jazyku Java. Jazyk C# byl vyvinut společností Microsoft a poprvé představen spolu s první verzí platformy *.NET Framework* v roce 2002 jako součást vývojářských nástrojů *Visual Studio .NET* a později v roce 2006 schválen jako standard ECMA-334 a ISO/IEC 23270:2006. Jazyk nativně podporuje aplikační infrastrukturu CLI a její služby (zejména automatickou správu paměti a řízení běhu v CLR) a knihovnu tříd FCL, díky které je vývoj aplikací v tomto jazyce velmi rychlý a efektivní. Od původního vydání byl jazyk C# aktualizován souběžně s platformou .NET; přehled verzí je uveden v tabulce (16).

Verze jazyka	Rok vydání	Nástroj	Přidaná funkcionalita
C# 1.0	2002	Visual Studio .NET	Základní verze jazyka
C# 2.0	2005	VS 2005	Generika, částečné třídy, iterátory, anonymní delegáty, nulovatelnné typy, …
C# 3.0	2007	VS 2008, VS 2010	Implementace LINQ (integrovaný dotazovací jazyk), částečné metody, anonymní typy, …
C# 4.0	2010	VS 2010	Kovariance, pojmenovatelné volitelné parametry, vylepšení vícevláknového programování, …
C# 5.0	2012	VS 2012	Asynchronní metody, informace o volajícím metody, …

Tabulka 5 - Vývoj verzí jazyka C# (vlastní zpracování dle 16)

5 Návrh řešení

V této části bude rozebráno samotné řešení vlastní aplikace pro správu tenkých klientů, která je předmětem diplomové práce. Budou zde popsány postupy návrhu, použité technologie a rozhodnutí, vedoucí k řešení dílčích problémů. Popis nebude zaměřen na tvorbu či analýzu programového kódu, ale spíše na zmapování technik a postupů, stojících za funkcionalitou jednotlivých komponent aplikace. V závěru kapitoly pak bude nastíněn průběh tohoto projektu, zhodnocena časová a nákladová náročnost a posouzen celkový přínos realizace projektu pro firmu.

5.1 Zhodnocení požadavků a základní koncept

Z požadavků definovaných v analytické části práce je patrné, že je požadována funkčně bohatá aplikace, sestávající ze tří komponent, na jejichž rozhraní bude probíhat spolehlivá síťová komunikace prostřednictvím TCP/IP socketů, přičemž procesy mají komunikovat formou předdefinovaných a zašifrovaných zpráv ve formátu XML. Dalším požadavkem je použití souborové databáze pro serverovou část aplikace (z důvodu zjednodušení nasazení aplikace) a poskytnutí HTTP repositáře rovněž jako součásti aplikace serveru. Všechny aplikační komponenty jsou předurčeny pro systémy *Windows*, očekává se tedy kompatibilita se všemi aktuálně používanými serverovými i desktopovými variantami tohoto systému a v případě služby Management Agenta pro tenké klienty kompatibilita se systémy *Windows Embedded Standard 2009* a *Standard 7*, instalovanými do tenkých klientů *TiCtOG*. Pro usnadnění vývoje byla schválena možnost použití platformy *Microsoft .NET Framework*, avšak vzhledem ke škále podporovaných operačních systémů a s ohledem na minimalizaci velikosti instalovaného software byla zvolena verze 2.0.

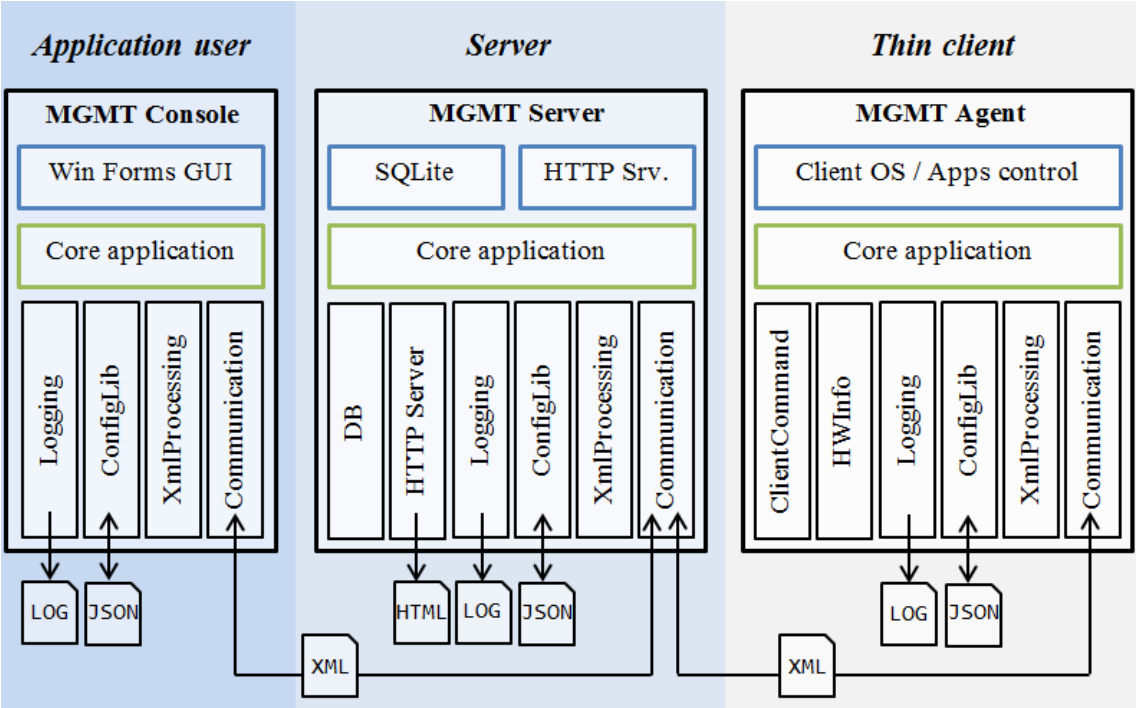
5.1.1 Volba technologií a nástrojů

S ohledem na výše uvedené informace navrhuji pro vývoj aplikace využít programovací **jazyk C#** verze 2.0, korespondující s platformou ***Microsoft .NET Framework*** verze 2.0. Jako hlavní vývojářský nástroj pak poslouží integrované vývojové prostředí ***Microsoft Visual Studio 2010*** (zde primárně za účelem vývoje aplikací a služeb v jazyce C#, ale také jako nástroj pro editaci souborů XML a práci

s tabulkami databáze SQLite), v kombinaci se systémem správy verzí a repositářem ***Git*** (včetně integračního rozšíření a grafického rozhraní *Git Extensions*), provozovaným na serverech společnosti. Pro účely vedení databáze Management Serveru navrhuji použít jednoduchý (avšak velmi efektivní) souborový databázový systém ***SQLite***.

5.1.2 Základní koncept aplikace

Jak již bylo zmíněno, celá aplikace má sestávat ze tří hlavních komponent, jejichž interakce bude zajištěna prostřednictvím výměny XML zpráv přenášených po síti. Každá z aplikací bude zajišťovat určitou část funkcionality celého řešení a bude využívat univerzální funkce a datové struktury, poskytované dynamickými knihovnami společnými pro všechny části aplikace. Pro lepší představu o architektuře celého řešení lze použít následující obrázek, ve kterém jsou schematicky znázorněny aplikační komponenty, jejich prostředky, využívané knihovny a ve zjednodušeném pojetí i vzájemná interakce. Návrh dynamických knihoven, XML komunikace a samotných aplikací/služeb bude detailněji rozepsán v dalších částech této kapitoly.



Obrázek 11 - Zjednodušené schéma komponent aplikace (vlastní zpracování)

5.2 Společné komponenty aplikací

Důležitou součástí aplikace o více komponentách je návrh vhodného způsobu interakce mezi jednotlivými komponentami, na základě dat, která si mají vyměňovat – zde popíšu návrh v aplikaci použitého systému komunikace prostřednictvím předdefinovaných XML zpráv. Dále budou popsány sdílené dynamické knihovny, jejichž účelem bude zajistit specifickou funkcionalitu univerzálním způsobem pro všechny tři komponenty aplikace tak, aby tyto funkce nebylo nutné implementovat pro každou aplikaci zvlášť a případně bylo možno snadno upravovat kód pro všechny komponenty jednorázově.

5.2.1 Komunikační XML zprávy

Základním rozhodnutím při návrhu aplikace byla volba formátu XML pro výměnu dat mezi komponentami aplikace. Tento formát jsem zvolil zejména pro jeho přehlednost, snadné strojové zpracování a schopnost uchovávat strukturovaná data, jako jsou např. tabulky. Pro komunikaci v této aplikaci bude navržen jednoduchý systém zpráv, definující povinnou strukturu XML zprávy a její obsah, kterému budou jednotlivé části aplikace rozumět a na základě zprávy parametricky vykonávat určenou činnost. Tento způsob výměny dat bude použit ve všech součástech aplikace.

Struktura a obsah XML zprávy

Každá zpráva bude jednoduchým dokumentem XML obsahujícím kořenový element `message`. V tomto elementu pak již mohou být samotné příkazy či informace pro příjemce zprávy (jeden či více), definované strukturou dalších elementů obsaženou v elementech `cmd`. Obsah tohoto elementu je závazný, a sestává z následujících položek:

- `group` – definuje typ interakce, pod kterou zpráva spadá – umožňuje filtrování přijatých zpráv a větvení při zpracování u příjemce, je specifikována zkrácenými řetězci ve tvaru `M2S` (Management-to-server), `C2S` (Client-to-server), `S2M` a `S2C`[4].
- `auth_user` – pro interakce `M2S` obsahuje jméno autentizovaného uživatele, který zprávu zaslal, pro ostatní interakce je pole prázdné.
- `route` – definuje elementy `from` a `to`, obsahující IP adresy odesílatele a příjemce.

[4] Jsou definovány výhradně tyto interakce, přímá interakce Management Console M a tenkého klienta C (M2C/C2M) není žádoucí a v aplikaci nebude použita.

- **`version`** – obsahuje informaci o verzi aplikace generující zprávu (např. 1.0.7.0)
- **`action`**[5] – specifikuje akci, která bude vykonána u příjemce, umožňuje větvení při zpracování a následné volání odpovídající metody s požadovanými parametry.
- **`data`** – hlavní datová oblast zprávy, její obsah je volitelný a závisí na uvedené akci, pro kterou obvykle poskytuje parametry či data, např. pro akci požadavku autentizace `AUTH_REQ` může obsahovat tabulku s uživatelským jménem a otiskem hesla (hash).
- **`schedule`** – volitelný parametr pro konfigurační úlohy tenkých klientů, umožňující jejich naplánování na stanovený čas, namísto okamžitého provedení.

V následujícím příkladu je uvedena vzorová zpráva požadavku uživatele konzole na autentizaci serverem (M2S) s vloženou XML reprezentací tabulky dat:

```
<?xml version="1.0" encoding="utf-8"?>
<message>
  <cmd>
    <group>M2S</group>
    <auth_user>Uzivatel</auth_user>
    <route>
      <from>192.168.1.45</from>
      <to>192.168.1.10</to>
    </route>
    <version>1.0.7.0</version>
    <action>AUTH_REQ</action>
    <data>
      <datatable>
        <credentials>
          <user>Uzivatel</user>
          <password>aGVsbG8gd29ybGQ=</password>
        </credentials>
      </datatable>
    </data>
  </cmd>
</message>
```

[5] Přehled dostupných interakcí a jejich popis je uveden v příloze č. 1.

5.2.2 Knihovna pro práci s XML zprávami

Programovací jazyk C# obsahuje již v základu pokročilou funkcionalitu pro práci s XML dokumenty, stále je však potřeba s přijatým XML řetězcem provést mnoho úkonů, než je možné číst a upravovat hodnoty jednotlivých elementů či atributů. V této aplikaci budou základní XML struktury univerzální, a proto je vhodné vytvořit metody a objekty pro zjednodušení jejich zpracování. Pro tento účel bude navržena knihovna `XmlProcessing.dll`. Poskytované metody a třídy budou dále stručně popsány.

XmlMessageInfo

Tato třída poskytne jednoduchou strukturu pro ukládání a rychlou práci s XML zprávami. Obsahuje proměnné, které jsou konstruktorem inicializovány na hodnoty obsažené v XML zprávě, a `get` metody pro čtení hodnot jednotlivých položek.

XmlInterface

Statická třída poskytující zejména pomocné metody pro převod XML řetězců na objekty výše uvedené třídy `XmlMessageInfo` a zpět; dále definuje metody pro jednoduchý převod tabulek typu `System.Data.DataTable` na textovou XML reprezentaci a zpět, pro účely ukládání do/načítání z datové oblasti XML zpráv.

Další prostředky

Mimo primární funkcionalitu pro práci s XML zprávami poskytne knihovna také konstanty a výčtové typy pro interakce (M2S, C2S,…) a datové struktury pro XML reprezentace konfiguračních úloh a jejich parametrů.

5.2.3 Knihovna pro síťovou komunikaci

Klíčovou součástí návrhu aplikace je knihovna **`Communication.dll`** – tato poskytne aplikacím prostředky pro komunikaci po síti prostřednictvím TCP socketů. Umožňuje jednotlivým komponentám buď naslouchat na portu a přijímat příchozí spojení se zpětným voláním (asynchronní programování), anebo se k těmto serverům připojovat a zasílat data. Knihovna také zajišťuje šifrování a dešifrování přenášených dat. Dále budou popsány hlavní třídy knihovny a princip jejich fungování.

ServerListener

Hlavní serverová třída, bude obsahovat pracovní socket serveru (typu TCP stream IPv4) pro naslouchání na volitelném portu a metody pro asynchronní zpětné volání při připojení klienta a vytvoření nové instance objektu třídy `HandleClient` pro zpracování příchozí komunikace. Tato třída také zajistí správu seznamu probíhajících připojení, ukončení všech spojení v případě uzavření aplikace a vyvolání události pro aplikaci, pokud dojde k přijetí dat.

HandleClient

Objekty třídy `HandleClient` budou vytvářeny pro jednotlivá příchozí připojení na server – každý takový objekt převezme přijatý socket od třídy `ServerListener` a bude pracovat pouze s tímto spojením, zatímco server dále naslouchá. Třída `HandleClient` zajistí čekání na příchozí data, jejich přijetí a předání aplikaci v řetězci, kontrolu vypršení časového limitu pro připojený socket a jeho odpojení.

Transmitter

Třída `Transmitter` umožní aplikaci připojení k naslouchajícímu serveru, zasílání zpráv a asynchronní přijímání odpovědi na tyto zprávy. V kódu bude možno komunikaci realizovat voláním metody `OpenTransmitter()`, zajišťující připojení na socket naslouchajícího serveru, po jejímž úspěšném dokončení půjde zasílat druhé straně data metodou `TransmitData()`, využívající systémovou třídu `System.IO.StreamWriter` pro zápis dat do síťového proudu socketu, anebo ukončit spojení. Třída `Transmitter` informuje stejně jako `ServerListener` aplikaci o přijetí dat na socket vyvoláním události, na kterou pak může aplikace zareagovat patřičnou akcí.

Šifrování dat

Součástí knihovny budou dále metody, pomocí kterých budou data před odesláním šifrována a po přijetí druhou stranou dešifrována – tímto způsobem je komunikace chráněna proti přečtení a manipulaci a rovněž zavedena další kontrola integrity. Pro šifrování navrhuji použít symetrickou šifru Triple DES v režimu CBC (Cipher Block Chaining) na základě statického klíče a výstupní řetězec kódovat pomocí Base64 z důvodu lepší kompatibility při přenosu.

5.2.4 Knihovna pro práci s konfiguračními soubory

Každá komponenta aplikace může obsahovat nastavitelné položky, vztahující se k určitým systémovým parametrům, přednastaveným cestám k souborům či navoleným preferencím uživatele. Tyto konfigurační údaje je nutné uchovávat v místních souborech, ke kterým může aplikace přistupovat a efektivně s nimi pracovat, avšak ve formátu čitelném a jednoduše upravitelném člověkem. Za tímto účelem bude vytvořena knihovna **`ConfigLib.dll`**, jejíž hlavní funkcí bude zajištění prostředků pro čtení a zápis hodnot proměnných do souborů datového formátu JSON. Tento formát volím zejména pro jeho strukturovanost, snadnou čitelnost a minimalistický způsob zápisu. Pro podporu zpracování těchto souborů bude využito funkcí knihovny *Newtonsoft Json.NET 2.0*. Ve vlastní knihovně `ConfigLib` jsou pak implementovány následující třídy a funkce.

CValue

Třída `CValue` poskytne univerzální datový typ pro proměnné konfiguračních souborů. Obsahem bude struktura pro uložení podporovaných datových typů konfigurace: `int`, `double`, `string`, `bool` a pole `string[]`. Volba typu je předurčena vytvořením objektu použitím jednoho z přetížených konstruktorů a ke každé proměnné bude možno přistupovat pomocí `get` metody daného typu: `GetInt()`, `GetDouble()`, apod.

Config

Statická třída `Config` bude již vlastním nástrojem aplikace pro čtení a zápis konfiguračních souborů. Poskytne základní metody `load()` pro načtení struktury souboru JSON do paměti a `save()` pro uložení struktury do souboru. Cesta k tomuto souboru je implicitně předvolena na soubor v pracovním adresáři aplikace, avšak je konfigurovatelná voláním nastavovací metody. Po načtení konfigurační struktury JSON do paměti bude možno s konfigurací v aplikaci pracovat pomocí metody `get()`, která vrátí objekt typu `CValue` s patřičnou proměnnou konfigurace odpovídající uvedenému klíči a metody `set()`, která je několikanásobně přetížena a umožní přiřadit proměnné odpovídající danému klíči hodnotu libovolného podporovaného datového typu, a v případě neexistujícího klíče vytvoření nové proměnné.

5.2.5 Knihovna protokolování

Za běhu aplikace bude prováděno velké množství operací, jejichž průběh není, zejména pro služby, viditelně indikován uživateli, stejně jako případné nestandardní situace, chyby či výjimky. Tyto informace je však nutno uchovávat pro potřeby ladění, ověřování korektnosti práce programu a sledování průběhu procesů. Za tímto účelem je nutné zavést systém protokolování– v této aplikaci pomocí knihovny `Logging.dll`.

Logger

`Logger` bude statickou třídou, využitelnou ve všech částech aplikace pro protokolování aplikačních událostí za běhu – bude definovat metody a výčtové typy protokolu. Základní poskytovanou metodou bude metoda `write()`, která umožní zapsat událost do protokolu, a to na čtyřech úrovních definovaných strukturou `Level`: `ERROR` – kritická chyba aplikace či výjimka, `WARNING` – upozornění na vzniklou netypickou situaci, `DEBUG` – zprávy určené pro události ladění a `INFO` – informace o standardních událostech při běhu aplikace. Protokolování může být volitelně konfigurací nastaveno do dvou režimů – `DEVELOPMENT` a `PRODUCTION`, přičemž v druhém z těchto režimů nebudou vypisovány zprávy úrovně `DEBUG`. Konfigurovatelná bude také cesta k souboru protokolu, přičemž výchozí je soubor v pracovním adresáři. Velikost výstupního souboru bude volitelně limitována a po překročení povolené velikosti provedena rotace. Třída `Logger` se také při spuštění aplikace přihlásí k odběru informací o neošetřených výjimkách a může tak zaznamenat informace o chybě i při pádu aplikace.

5.3 Management Server

Tato část se bude zabývat návrhem Management Serveru, služby, která bude jádrem celé aplikace a centrální komponentou zajišťující zprostředkování interakce administrátora se skupinami tenkých klientů vedených v centrální správě. Nejprve budou popsány serverem zajišťované funkce, návrh databáze a podpůrných knihoven, a následně rozebrána vlastní aplikační služba, její komponenty a principy fungování.

5.3.1 Přehled zajišťovaných funkcí

Management Server bude poskytovat základní centrální funkcionalitu, potřebnou pro provoz správy – tyto funkce lze obecně shrnout následujícími body:

- Vedení centrální databáze tenkých klientů, konfiguračních úloh, uživatelů a informací o skupinách.
- Provoz HTTP serveru pro přístup k souborovému repositáři.
- Přijímání, zpracování a odpovědi na požadavky uživatele konzole.
- Interakce s tenkými klienty – vyhledávání, přijímání periodických notifikací a reportů, zasílání příkazů.
- Řízení zpracování konfiguračních úloh v čase.

5.3.2 Návrh databáze

Součástí každého řešení centrální správy zařízení je databáze spravovaných komponent, uživatelů, konfigurací a dalších položek. V rámci této aplikace navrhuji vytvořit jednoduchou databázi souborového databázového systému *SQLite* (obsahující pouze 4 tabulky). Tento návrh bude dále rozepsán.

Tabulky databáze

Pro ukládání informací o uživatelích aplikace správy bude sloužit tabulka `users`. V této tabulce budou uvedeny přihlašovací údaje formou uživatelského jména a otisku přístupového hesla (SHA-1 hash). Dále zde budou informace o oprávnění uživatele k určitým akcím v podobě řetězce "RO" (read-only), respektive "RW" (read-write), informace o blokaci uživatele a volitelné kontaktní informace – e-mail a telefonní číslo. Pro číselnou identifikaci položek pak bude sloužit umělý primární klíč ID (tato identifikace bude použita i ve všech dalších tabulkách databáze). Struktura tabulky a použité datové typy jsou uvedeny dále.

Tabulka uživatelů *users*		
Sloupec	**Datový typ**	**Popis**
id	INTEGER	Identifikace, PK
user_name	VARCHAR	Uživatelské jméno
password	VARCHAR	Otisk hesla
acl	VARCHAR	Oprávnění ke změnám
blocked	BOOLEAN	Blokování uživatele
email	VARCHAR	E-mail
phone	VARCHAR	Telefonní číslo

Tabulka 6 - Struktura databázové tabulky uživatelů (vlastní zpracování)

Informace o tenkých klientech budou ukládány do tabulky **`clients`**. V tabulce budou obsaženy zejména stavové informace o zařízení a jeho systémových prostředcích pro účely zobrazení v konzoli a pro potřeby řízení konfiguračních úloh. Dále zde budou informace o síťovém názvu (hostname), IP adrese počítače a MAC adrese primárního síťového adaptéru, uživatelem přidaný titulek a další položky, viz popis.

Tabulka tenkých klientů *clients*		
Sloupec	**Datový typ**	**Popis**
id	INTEGER	Identifikace, PK
grp_id	INTEGER	Příslušnost ke skupině, FK
mac	VARCHAR	MAC adresa síťového adaptéru
name	VARCHAR	Síťový název počítače
ip	VARCHAR	Aktuální IP adresa
ewf	BOOLEAN	Status filtru EWF
lockdown	BOOLEAN	Status uzamčení tenkého klienta
cpu	INTEGER	Aktuální vytížení procesoru
total_hd	INTEGER	Velikost systémového oddílu
free_hd	INTEGER	Volné místo na systémovém oddílu
total_ram	INTEGER	Celková operační paměť
free_ram	INTEGER	Volná operační paměť
type	VARCHAR	Označení modelu zařízení
os	VARCHAR	Popis operačního systému
ver_fw	VARCHAR	Verze firmware TiCtOG
alive	BOOLEAN	Informace o běhu zařízení
last_notification	DATETIME	Čas přijetí poslední notifikace
description	VARCHAR	Volitelný uživatelský popisek

Tabulka 7 - Struktura databázové tablky tenkých klientů (vlastní zpracování)

Pro rozdělování tenkých klientů do skupin v rámci aplikace centrální správy (např. podle místností či jiného kritéria) budou sloužit skupiny evidované v jednoduché tabulce **`groups`**, obsahující vlastní název skupiny, identifikaci a identifikaci rodiče pro možnost vytvoření hierarchie vnořených skupin.

Tabulka skupin tenkých klientů *groups*		
Sloupec	**Datový typ**	**Popis**
id	INTEGER	Identifikace, PK
parent_id	INTEGER	Nadřazená skupina, FK
name	VARCHAR	Název skupiny

Tabulka 8 - Struktura databázové tabulky skupin (vlastní zpracování)

Probíhající konfigurační úlohy budou uloženy v tabulce **`jobs`**. Budou zde zahrnuty položky identifikující danou úlohu, uživatele, který ji zadal, cílové zařízení na kterém má být úloha provedena, stav provádění úlohy, název a samotná XML data obsahující strukturu příkazů a parametrů úlohy.

Tabulka konfiguračních úloh *jobs*		
Sloupec	**Datový typ**	**Popis**
id	INTEGER	Identifikace, PK
parent_id	INTEGER	Nadřazená úloha/kontejner, FK
usr_id	INTEGER	Identifikace zadavatele, FK
target_id	INTEGER	Identifikace zařízení, FK
entry_time	DATETIME	Čas zadání úlohy
execution_time	DATETIME	Plánovaný čas spuštění úlohy
completed_time	DATETIME	Čas dokončení úlohy
status	INTEGER	Stavový kód dokončení úlohy
extended_status	TEXT	Rozšiřující text pro případ chyby
job_name	VARCHAR	Název konfigurační úlohy
job_xml	TEXT	XML struktura s instrukcemi
desc	TEXT	Uživatelský popis úlohy
timeout	INTEGER	Časový limit (s) pro vykonání úlohy

Tabulka 9 – Struktura databázové tabulky úloh (vlastní zpracování)

Z výše uvedených návrhů tabulek je patrné, že jsou využity pouze základní datové typy jazyka SQL. Toto rozhodnutí vychází z nutnosti transformovat tabulky databáze na objekty `System.Data.DataTable`, vkládat textově do XML struktury pro přenos mezi komponentami aplikace a následně převádět textové hodnoty elementů zpět. Celá databáze bude uložena v jednom binárním souboru v adresáři služby serveru a spravována pomocí vlastní knihovny popsané v další části. Přestože databázový systém *SQLite* podporuje globální šifrování celé databáze, tato funkce nebude z výkonnostních důvodů použita.

5.3.3 Knihovny serveru

Služba serveru bude vyžadovat oproti ostatním součástem aplikace navíc knihovnu pro práci s databází a jednoduchý HTTP server.

Knihovna pro práci s databází

Knihovna **`db.dll`** umožní serveru přístup k databázi, kontrolu struktury, řízení přístupu z více vláken, a poskytne metody pro práci s daty v tabulkách. Základem této knihovny je ADO.NET provider `System.Data.SQLite`, poskytující třídy pro připojení k SQLite databází. Samotná knihovna při spuštění služby serveru zajistí připojení k databázi, ověří, že existuje soubor s databází a všechny potřebné tabulky, případně vytvoří novou databázi. Další funkce jsou již zaměřeny na práci s konkrétními tabulkami – z aplikace mohou být volány metody `CreateRecord`() pro vytvoření záznamu v tabulce, `UpdateRecord`() pro aktualizaci, `GetRecords`() a `RemoveRecord`() pro čtení/mazání. Všechny tyto příkazy jsou směrovány na bezpečnou funkci `Query`(), která je při volání vždy uzamčena pro ostatní vlákna a zabraňuje tak možným kolizím při pokusu více modulů zároveň o změny v databázi (požadavky jsou vyřízeny postupně). Tato funkce je navíc přístupná i z kódu serveru pro přímé vykonávání SQL dotazů, na které je, stejně jako u ostatních metod této knihovny, vrácena odpověď v tabulce typu `System.Data.DataTable`.

Knihovna HTTP serveru

Jedním z požadavků na aplikaci je poskytnutí interního HTTP serveru pro účely distribuce aktualizací operačního systému tenkých klientů a procházení detailních

reportů v prohlížeči konzole. Tento server bude zajištěn knihovnou `HTTPServer.dll`. Vzhledem ke komplexnosti a časové náročnosti vývoje vlastního webového serveru bude jako základ této knihovny použit vzorový kód *Sample HTTP Server Skeleton in C#*, volně dostupný ze serveru *CodeProject* (18).

5.3.4 Návrh služby serveru

Zde bude popsána vlastní serverová část aplikace. Tato komponenta bude navržena jako služba systému Windows, tedy ne uživatelská aplikace, ale systémový proces, běžící na pozadí. Popis bude zaměřen na jednotlivé třídy, tvořící serverovou službu, princip jejich fungování a jejich propojení v rámci funkčního celku aplikace.

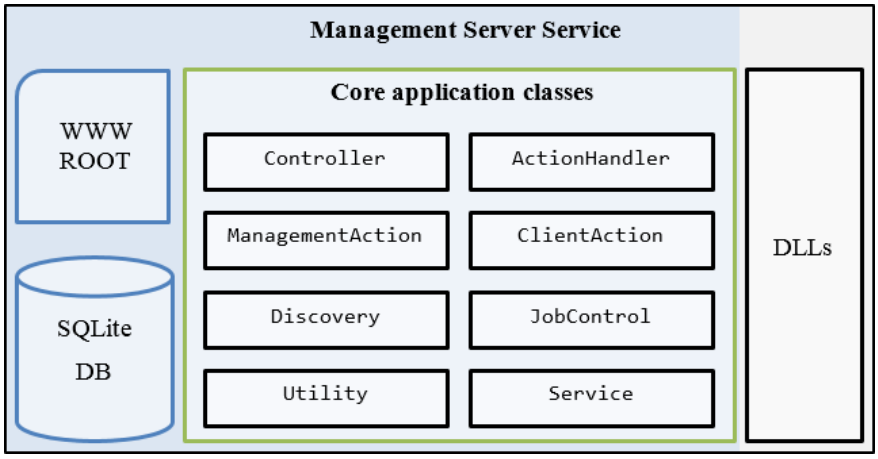
Vývoj služby pro systém Windows

V prostředí *Visual Studio* je vývoj služeb pro systém Windows velmi podobný vývoji např. konzolové aplikace. Na rozdíl od běžné aplikace s jednoduchým vstupním bodem definovaným metodou `Main()` je zde však nutnost vytvoření standardních tříd a metod, se kterými bude řídící systém služeb Windows – Service Control Manager (SCM) schopen pracovat. Prakticky se jedná o vytvoření třídy služby, dědící ze vzorové třídy `ServiceBase`, a definující vlastní implementaci systémových metod `OnStart()` (logika provedená při spouštění služby, v tomto případě start všech funkcí serveru), `OnStop()` (ukončovací logika při zastavování služby, zde zastavení naslouchání a podprocesů na separátních vláknech) a volitelně další metody řízení běhu služby SCM, překrývající definice rodičovské třídy, např. `OnShutdown()` a další. Dále je nutné nastavit základní vlastnosti služby – název služby v krátké (pro identifikaci a ovládání služby) a dlouhé variantě (pro uživatelské zobrazení v ovládacím panelu služeb a protokolování), oprávnění k naslouchání systémovým událostem, řízení vypnutí systému a případné závislosti na běhu jiných služeb. Dalším důležitým parametrem je uživatelský účet, pod kterým služba poběží; pro účely této aplikace jsem zvolil systémový účet `LocalSystem`, protože poskytne službě oprávnění k práci se všemi místními zdroji a nevyžaduje specifikaci přihlašovacích údajů. V neposlední řadě musí být pro každou službu definován výchozí typ spuštění – automatický, manuální anebo zakázáno; zde bude použit automatický režim – služba se spustí vždy při spuštění systému, bez ohledu na přihlášení uživatele (důležité zejména pro provoz na systémech *Windows Server*). Pro

účely nasazení služby je také nutné vytvořit instalační logiku, v prostředí *Visual Studio* je toto zajištěno vygenerovanou třídou `ProjectInstaller`. Nad tímto základem služby je již možno vytvořit vlastní aplikační logiku a požadovanou funkcionalitu.

Schéma komponent a tříd serveru

Následující schéma poskytuje zjednodušený přehled komponent a tříd serveru. Použité knihovny a návrh databáze byly již popsány v předchozím textu, nyní bude individuálně rozebrán návrh tříd, tvořících serverovou službu a jejich funkcionalita[6].



Obrázek 12 - Schéma komponent a tříd serveru (valstní zpracování)

Třída Controller (hlavní řídící třída)

Hlavní součástí serveru je třída `Controller`[7], zodpovídající za spuštění a inicializaci všech funkcí serveru, jejich zastavení při ukončování, načtení konfigurace a poskytnutí konfiguračních údajů ostatním třídám, řízení příchozí komunikace a provozu hlavního vlákna aplikace.

Spuštění serveru probíhá voláním metody `Start()` třídou služby v rámci systémové metody `OnStart()`. Tato metoda nejprve nastaví správný pracovní adresář a zkontroluje existenci potřebných souborů. Následně je spuštěno protokolování zajištěné knihovnou `Logging` a provedena inicializace databáze prostřednictvím databázové

[6] Úplný diagram tříd a metod služby serveru je uveden v příloze č. 2.
[7] Třída Controller, stejně jako všechny další zde popisované třídy služby serveru, je statická.

knihovny. Dalším krokem je načtení konfigurace ze souboru do proměnných pomocí knihovny `ConfigLib`. Obsah a forma konfiguračního souboru může být následující:

```
{
  "listening_port": 7130,
  "client_port": 7130,
  "http_port": 7131,
  "http_wwwroot": ".\wwwroot",
  "debug": false
}
```

Po načtení konfigurace je na základě získaných údajů vytvořena instance třídy `ServerListener` knihovny `Communication`, spuštěno naslouchání na stanoveném portu a vytvořena fronta pro příchozí komunikaci. Tato fronta je periodicky kontrolována metodou `CheckQueue()`, běžící na samostatném vlákně, které je rovněž iniciováno při startu. Následuje spuštění webového serveru poskytnutého knihovnou `HTTPServer`, spuštění třídy řízení úloh `JobControl` a vlákna pro periodickou kontrolu stavu tenkých klientů. Tímto je inicializace serveru dokončena.

Zastavení serveru je zajištěno voláním metody `Stop()`. Tato metoda ukončí naslouchání na portu, nastaví řídící proměnnou pro spuštěná vlákna kontrolních metod na ukončení (dojde k přerušení podmínky cyklů) a zastaví případné vyhledávání tenkých klientů třídou `Discovery`. Dále proběhne zastavení řízení úloh třídy `JobControl`, zastavení webového serveru a uzavření databáze. Server je zastaven a služba může být systémem ukončena.

Veškerá příchozí komunikace je směřována do fronty. Tato fronta je cyklicky kontrolována metodou `CheckQueue()` (běžící na separátním vlákně) každých 20ms. V případě že jsou ve frontě položky, je první položka odebrána a předána třídě `ActionHandler` ke zpracování. Třída `Controller` dále poskytuje ostatním třídám serveru metody pro přístup k načteným proměnným konfigurace.

Třída ActionHandler (větvení příchozích zpráv)

Větvení a rozhodnutí o způsobu zpracování příchozí XML zprávy je zajištěno třídou `ActionHandler`. Tato třída pracuje s objekty komunikace, přejatými z fronty vedené v hlavní třídě. Tyto objekty obsahují příchozí data, referenci na pracovní socket

komunikace s odesílatelem a metody pro odeslání odpovědi či ukončení komunikace. Třídou poskytovaná metoda `ProcessMessage()` převede přijatou XML zprávu na objekt typu `XmlMessageInfo` a na základě hodnoty položek group a action rozhodne o třídě a metodě, které bude komunikační objekt a objekt zprávy předán k provedení dané činnosti (parametrizované volání metod ve třídách `ManagementAction` pro zprávy typu M2S a `ClientAction` pro zprávy typu C2S). Pokud je přijata zpráva nepodporovaného typu, je zahozena a komunikace s odesílatelem ukončena. V případě že je typ zprávy správný, ale není definována metoda pro uvedenou akci, je zaslána odpověď o nepodporovaném požadavku.

Třída ManagementAction (zpracování požadavků uživatele)

V rámci třídy `ManagementAction` jsou zpracovávány, s výjimkou konfiguračních úloh, všechny příchozí zprávy typu M2S[8], tedy management-to-server – příkazy či dotazy zaslané z uživatelské konzole. Jsou zde obsaženy metody pro autentizaci uživatele, zasílání dat z tabulek databáze na konzoli pro účely výpisu a úprav, ovládání vyhledávání nových zařízení, aktualizaci a přípravu dat pro zobrazení v konzoli.

Autentizace uživatele aplikace a ověření funkčních podmínek je prováděna metodou `AuthRes()`, zajišťující ověření a odpověď postupně v následujících krocích:

1. Ověření shody verzí: nejprve je nutno porovnat verzi serveru a konzole – tyto se musí z důvodu kompatibility shodovat. K porovnání slouží položka `version` příchozí zprávy a interní číslo verze serveru. V případě rozdílu je autentizace zrušena a uživatel upozorněn na vyžadovanou verzi aplikace.
2. Ověření časové synchronizace: pro účely plánování úloh je vyžadována přibližná shoda času. V tomto kroku je porovnán čas vygenerování zprávy konzolí a čas na serveru. Pokud rozdíl časů spadá mimo toleranční interval <-2;2> minut (definovaný z praktických důvodů), je autentizace zrušena a uživatel upozorněn na časový rozdíl.
3. Vyhodnocení přihlašovacích údajů: přijaté přihlašovací údaje uživatele (řetězec uživatelského jména a SHA-1 otisk hesla) jsou porovnány s údaji v databázi. Pokud je nalezena shoda jména i hesla, je zaslána potvrzovací zpráva umožňující přihlášení konzole k serveru a zasílání dalších zpráv. V případě neshody je autentizace zrušena a uživatel upozorněn na chybné zadání jména či hesla.

[8] Seznam všech definovaných zpráv a jejich popis je uveden v příloze č. 1.

Pro práci se záznamy v tabulkách databáze jsou pro každou tabulku (uživatelé, skupiny, klienti, úlohy) definovány metody `Get()`, `Modify()`, `Add()` a `Delete()`, které umožňují předávat výsledky požadovaného výběru z databáze aplikaci konzole k dalšímu zpracování, přijímat zpracovaná data a zanášet provedené změny do databáze. Tato funkcionalita slouží zejména k úpravám a čtení seznamu uživatelů, rozřazování klientů do skupin a přidávání vlastních popisků. Pro účely zobrazování informací v konzoli slouží speciální metoda `MgmtViewRefreshRes()`, která zajistí aktualizaci dat na serveru a zaslání pouze vybraných informací nutných pro aktuální pohled uživatele, nikoliv celých tabulek.

Pro ovládání subsystému vyhledávání jsou zde definovány metody umožňující parametrizované spuštění či zastavení činnosti samostatné třídy `Discovery`, metody poskytující funkcionalitu zaslání průběžného výsledku vyhledávání na požadavek konzole (pro účely monitorování průběhu a zobrazení již nalezených zařízení) a metody pro vyžádání rozeslání požadavku na přidání vybraných zařízení do správy serverem.

Odpovědí serveru na každou přijatou zprávu od aplikace Management Console vyžadující určitá data je zpráva obsahující právě tato vyžádaná data. V případě, že se jedná o zprávu vyžadující provedení určité akce serverem, avšak žádná data nemají být zaslána zpět, je vrácena zpráva o úspěšnosti (XML zpráva `ACTION_REPORT`), v jejíž datové oblasti je uvedena informace o provedené akci a výsledku, či případné chybě.

Třída ClientAction (zpracování příchozích dat od klientů)

Zpracování základních zpráv typu C2S (client-to-server) přijatých od spravovaných tenkých klientů je zajištěno třídou `ClientAction`. Tato obsahuje metody `DiscoveryRes()`, pro zpracování odpovědi klienta na dotaz v rámci vyhledávání zařízení a předání získaných informací o zařízení vyhledávací třídě `Discovery`, `ClientNotificationProcess()`, pro zpracování pravidelných notifikací zapnutých tenkých klientů o jejich stavu a aktualizaci jejich záznamů v databázi a metodu `SaveReport()`, která zařídí přeformátování přijatého reportu klienta, přiřazení šablony transformace XSLT a uložení do HTTP repositáře pro přístup z aplikace konzole. Metody zpracování C2S zpráv informujících o průběhu konfiguračních úloh jsou dále z důvodu logické příslušnosti implementovány v rámci třídy `JobControl`.

Třída Discovery (vyhledávání nových zařízení)

Třída `Discovery` je hlavním nástrojem vyhledávání nových zařízení – implementuje jednoduchý systém pro postupné rozeslání dotazovacích zpráv na vybraný seznam IP adres a průběžný sběr odpovědí. Tento způsob byl zvolen na úkor jednorázového odeslání zprávy typu multicast, zejména z důvodu možnosti průběžného sledování, požadavku spolehlivou komunikaci pomocí TCP a v neposlední řadě i možné blokace multicast komunikace na aktivních prvcích v počítačové síti zákazníka.

Rozesílání zpráv při vyhledávání je provozováno na separátním vlákně, avšak omezeno na pouze jednu relaci (v případě více aktivních uživatelů aplikace) z důvodu snížení síťového provozu. Při přijetí požadavku uživatele na vyhledání zařízení je nejprve volána metoda `Start()`, která nastaví proměnné pro relaci vyhledávání (počet adres, přijaté pole IP adres, tabulka pro sběr výsledků, informace o běžící relaci) a spustí hlavní metodu `Discover()` na novém vlákně. Tato metoda pak v cyklu prochází pole cílových IP adres a rozesílá klientům zprávu `DISCOVERY_REQ`, přičemž pokud se do 1 vteřiny nepodaří připojit na otevřený port druhé strany, pokračuje se na další adresu. V případě úspěšného připojení je odeslána zpráva a maximálně 10 vteřin (prakticky se však jedná o desetiny vteřiny) vlákno čeká ve smyčce na přijetí odpovědi, následně importuje přijatá data do tabulky výsledků a pokračuje další adresou. Vyhledávání může být kdykoliv přerušeno metodou `Stop()`, k jejímu volání je však oprávněn pouze "majitel" dané relace vyhledávání, anebo třída `Controller`. Tato metoda přeruší cyklus, čímž dojde k zastavení vlákna a umožní spuštění nové relace vyhledávání.

Třída dále implementuje pomocné metody, umožňující informovat ostatní uživatele o běžící relaci, metodu pro přejímání přijatých odpovědí klientů ze třídy `ClientAction` a metodu `GetDiscoveryResults()`, umožňující předávat konzoli na vyžádání průběžné výsledky vyhledávání v podobě tabulky a počtu již obeslaných adres z přijatého seznamu pro účely indikace průběhu operace u uživatele.

Třída JobControl (řízení konfiguračních úloh)

Provádění konfiguračních úloh je primárním úkolem serveru a oblastí, ve které dochází k interakci všech tří komponent aplikace. Funkcionalita třídy `JobControl` je z tohoto důvodu velmi rozsáhlá. Poskytované funkce lze obecně popsat jako přijímání úloh od uživatele konzole, ukládání do databáze a řízení naplánovaného provádění úloh

(rozesílání úloh klientům a přijímání reportů). Prvky třídy jsou účelově rozděleny do šesti oblastí, v kódu reprezentovaných regiony. Tyto oblasti jsou následující:

Common – sekce základních prvků třídy, jsou zde obsaženy metody `Start()` a `Stop()` pro ovládání běhu pracovního vlákna řízení úloh a samotná pracovní metoda, zajišťující pravidelnou aktualizaci a zpracování úloh každých 10 vteřin. Dále je zde metoda `IsBusy()` indikující zaneprázdněnost řadiče úloh pro ostatní volání (musí počkat na odbavení) a metoda `RefreshData()` vynucující aktualizaci dat pro zobrazení. Řadič úloh může být zaneprázdněn jednou z následujících činností: pravidelné zpracování úloh, zařazování nových úloh, aktualizace úloh a zpracování zásahu do úloh. Tyto stavy jsou uchovávány v proměnných a na základě jejich hodnot je prováděno blokování, zamezující možné kolize při provádění více současných akcí.

Inbound – v této sekci jsou implementovány metody přejímající zprávy přijaté od tenkých klientů (C2S) a umožňující jejich zpracování. Při přijetí odpovědi klienta na požadavek provedení úlohy je volána metoda `HandleJobProcessRes()`, která oznamuje logice třídy přijetí úlohy klientem a zároveň nastavuje čas, za který platnost úlohy vyprší, pokud by nebyla dříve dokončena. Při dokončení úlohy je hlášení klienta o jejím provedení zpracováno metodou `HandleJobCompletion()`, která upraví v tabulce úloh patřičné informace o stavu dokončení úlohy a případném chybovém hlášení při selhání. Sekce také obsahuje logiku pro přijetí potvrzení o uzamčení tenkého klienta a započetí reinstalace systému na cílovém zařízení.

Enqueue – sekce zpracování zpráv od uživatele konzole (M2S), jedná se primárně o logiku zařazování nových úloh a zásahů do průběhu existujících úloh. Hlavní metoda `EnqueueJob()` zajišťuje vložení zaslaných konfiguračních úloh do databáze a jejich zařazení ke zpracování. Skládá se z dílčích volání dalších metod, které vedou k postupnému vytvoření struktury úlohy až po konkrétní příkazy. Další důležitou metodou je `JobOperation()`, jejíž volání umožňuje provádění příkazů `ABORT` (zrušení úlohy) a `RESUME` (znovuspuštění, např. pro případ vypršení času) nad již existujícími úlohami. Tyto příkazy jsou propagovány v rámci struktury úloh metodou `JobOperationTree()`, umožňující rekurzivně aplikovat nastavení na nižší úrovně.

Každá úloha vytvořená uživatelem sestává ze tří, hierarchicky uspořádaných úrovní: `job master` (hlavní uzel, kontejner pro celou úlohu), `target master` (uzel specifický pro konkrétní zařízení, může jich být v úloze více) a `target command`

(nejnižší úroveň, již není kontejnerem, ale přímo příkazem pro dané zařízení; těchto příkazů může být pro každé zařízení v rámci úlohy více). V rámci databáze jsou prvky všech úrovní reprezentovány záznamy tabulky `jobs`, s hierarchickým propojením pomocí cizího klíče `parent_id`, přičemž pouze položky typu `target command` mají vyplněno pole s příkazy. Hierarchii úrovní záznamů konfiguračních úloh lze pro lepší představu demonstrovat pomocí jednoduché XML struktury:

```
<job_master>
  <target_master>
    <target_command>Task 1</target_command>
    <target_command>Task 2</target_command>
  </target_master>
</job_master>
```

Process – hlavní sekce třídy, zajišťující pravidelné zpracování úloh a aktualizaci databáze. Toto zpracování je pravidelně prováděno metodou `ProcessJobs()`, která nejprve volá metody aktualizace, kontroly a odstranění nepotřebných záznamů, a následně prochází záznamy tabulky úloh. Při nalezení úlohy splňující předpoklady k provedení (tj. zařazená, neprobíhající úloha, naplánovaná na aktuální či dřívější čas) je ověřeno, zda je cílové zařízení právě spuštěno a úloha odeslána ke zpracování. Řízení odesílání konfiguračních úloh, jejich priorita a interakce s cílovým zařízením jsou založeny na rozsáhlém souboru pravidel a podmínek, který zde v zájmu rozsahu nebude podrobněji popisován.

Update – v této sekci jsou definovány obecné metody pro aktualizaci tabulky konfiguračních úloh, souhrnně volané metodou `UpdateJobCompletions()`. Tímto voláním je zajištěno posouzení průběhu úloh v čase a nastavení expirace, propagace stavu úloh (např. dokončení, nebo selhání) v rámci hierarchie konfiguračních úloh a vyhodnocení dokončení speciálních úloh (např. vypnutí/zapnutí zařízení).

Cleanup – sekce obsahující jedinou metodu `Cleanup()` pro odstraňování nepotřebných záznamů z tabulky konfiguračních úloh. Tato metoda prochází záznamy a odstraňuje položky dokončených úloh starší 5 minut, prázdné kontejnery a případné položky odkazující na neexistující objekt (zařízení, úloha).

Třída Utility (pomocné metody)

Pomocné metody, využívané ostatními třídami služby serveru jsou definovány v rámci třídy `Utility`. Jedná se zejména o metody poskytující informaci o lokální IP adrese serveru a vnitřní verzi služby. Dále jsou poskytovány pomocné metody pro sestavování a odesílání XML zpráv typu S2M/S2C, generování univerzálních odpovědí na nepodporované dotazy a práci s poli IP adres v XML zprávě. V neposlední řadě je zde také definována metoda `SendWakeOnLAN()`, která umožňuje vygenerovat tzv. "magic packet" pro specifikovanou MAC adresu a odeslat jej jako broadcast prostřednictvím modifikovaného UDP klienta, založeného na třídě `System.Net.Sockets.UdpClient`.

Třída Service (systémové metody)

`Service` je základní systémová třída služby, rozšiřující základní třídu `ServiceBase` a definující překryté metody `OnStart()` pro spuštění serveru, `OnStop()` pro zastavení a metodu `OnShutdown()` pro korektní ukončení serveru při vypínání systému.

5.4 Management Agent

Začlenění zařízení do správy pomocí zde navrhované aplikace bude na straně tenkého klienta zajištěno službou Management Agent. V této části bude popsán návrh specifických knihoven a vlastní aplikační služby. Popis služby agenta bude vzhledem ke značné podobnosti s výše popisovanou službou serveru a použití stejných mechanizmů a principů k řešení základních problémů zkrácen a zaměřen převážně na návrh metod pro realizaci vlastních konfiguračních úloh agentem.

5.4.1 Přehled zajišťovaných funkcí

Management Agent zahrne funkcionalitu akcí na straně tenkého klienta, souvisejících s aplikací správy. Jedná se zejména o poskytnutí následujících funkcí:

- Pravidelné informování serveru o stavu zařízení
- Provádění konfiguračních úloh a generování detailních reportů na vyžádání
- Monitorování zatížení zařízení v čase
- Inicializace reinstalace operačního systému ze síťového zdroje

5.4.2 Knihovny agenta

Kromě základních knihoven, společných pro všechny komponenty aplikace, bude Management Agent využívat specifické knihovny, poskytující zejména metody pro podporu provádění konfiguračních úloh a získávání informací o hardware, záměrně oddělené od vlastní služby agenta pro snadnější úpravy.

Knihovna ClientCommand

Knihovna `ClientCommand` bude tvořena souborem metod, určených pro podporu konfigurace místních systémových prostředků a aplikací tenkého klienta. Tyto metody využijí zejména v rámci knihovny definovaná volání funkcí aplikačního rozhraní Windows API pro práci s operačním systémem a předdefinované konstanty pro práci s registry a lokálními soubory nainstalovaných aplikací. Důležitým prvkem knihovny bude také řešení pro interakci s alternativním uživatelským rozhraním tenkého klienta, aplikací Shell32[9], prostřednictvím tzv. pojmenované roury (named pipe).

Hlavním prostředkem poskytovaným knihovnou samotné aplikaci bude třída `ClientCmd`, která bude určitým "funkčním obalem" všech systémových a pomocných metod definovaných v rámci knihovny. Poskytne zejména metody pro zobrazování informací na alternativním uživatelském rozhraní klienta a jeho uzamčení, zadávání příkazů místnímu filtru zápisu EWF, vypnutí či restartování systému, místní konfiguraci TCP/IP, změnu hesla účtu "Administrator" a další. Tyto metody budou podrobněji popsány v rámci podkapitoly zabývající se řešením samotných konfiguračních úloh.

Knihovna HWInfo

Knihovna `HWInfo` zajistí službě agenta přístup k informacím o hardwaru, které nejsou dostupné z běžných zdrojů, jako jsou metody knihovny tříd .NET, volání Windows API či databáze WMI. Poskytovaná funkcionalita bude založena na metodách knihovny Open Hardware Monitor, dostupné zdarma pod licencí Mozilla Public License 2.0. Samotná knihovna pak definuje metody pro zjištění modelového označení základní desky zařízení (důležité pro identifikaci modelu tenkého klienta) a údaj o teplotě procesoru pro účely monitorování a zobrazení s ostatními stavovými informacemi.

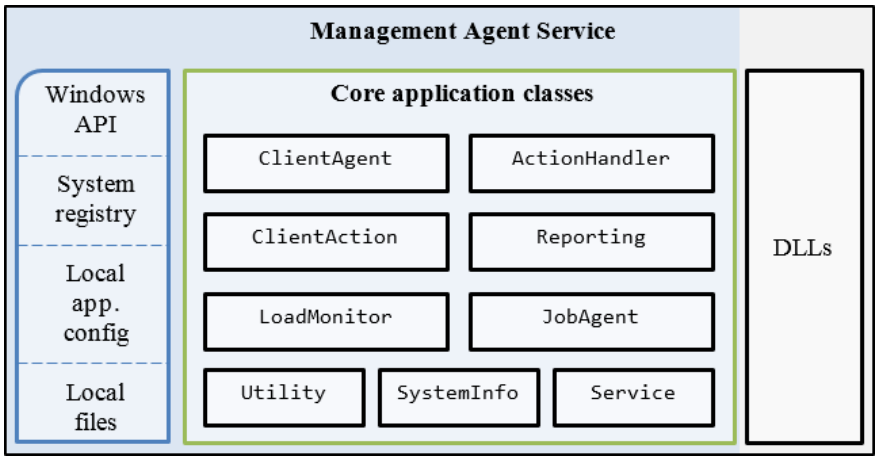
[9] Návrh a popis aplikace Shell32 je podrobněji zpracován v rámci bakalářské práce (2).

5.4.3 Služba agenta

Vlastní služba agenta bude složením komponent i metodami řešení v základu velmi podobná službě serveru – stejně jako u serveru, i zde je vyžadováno naslouchání na portu, přijímání zpráv, jejich zpracování/provedení akce a následná odpověď odesílateli (Management Serveru).

Schéma komponent a tříd služby agenta

Na následujícím obrázku je zjednodušeně schematicky shrnuto navrhované složení komponent a tříd služby agenta, sestávající z vlastní aplikační služby, připojených dynamických knihoven a systémových prostředků, aplikací a souborů, které bude služba při své činnosti využívat[10].



Obrázek 13 - Schéma komponent a tříd služby agenta (vlastní zpracování)

Funkční základ služby

Obdobně jako je tomu u služby serveru, i služba agenta je vystavěna nad základní systémovou třídou `ServiceBase`, definující metody pro řízení běhu služby, její instalaci a základní vlastnosti. Pro běh služby bude rovněž použit účet `LocalSystem`, z důvodu nutnosti přístupu k systémovým funkcím a běhu služby nezávisle na uživateli.

Základní řídící třídou služby je třída **`ClientAgent`**, zastávající stejnou funkcionalitu jako třída `Controller` u serveru, tedy inicializaci a ukončování služby (tj. načtení konfigurace, aktivace protokolování, ověření běhu vyžadovaných služeb [zde

[10] Úplný diagram tříd a metod služby agenta je uveden v příloze č. 3.

služby WMI], spuštění naslouchání na portu, aktivace monitorování zatížení, spuštění vlákna periodických notifikací pří startu a ukončení všech podprocesů při vypínání). Dále třída provozuje frontu pro příchozí komunikaci a příchozí zprávy předává dále ke zpracování třídě **`ActionHandler`**, zodpovědné za rozpoznání zpráv, větvení a volání příslušné metody pro zpracování a odpověď. Tyto metody jsou definovány v rámci třídy **`ClientAction`**. Pomocné metody pro provádění základních úkonů (práce s poli IP adres, XML, vytváření a odesílání zpráv, apod.) jsou aplikaci poskytovány třídou **`Utility`**.

Specifické třídy a funkce služby Management Agent

Jádrem funkcionality služby agenta jsou čtyři specifické třídy, zaměřené na práci s místními prostředky tenkého klienta a zpracování konfiguračních úloh.

Třída **`SystemInfo`** poskytuje aplikaci soubor metod pro získávání systémových informací o daném zařízení. Většina těchto metod je založena na dotazování interní systémové databáze WMI, funkcích z knihovny tříd .NET Framework (jmenné prostory `System.Diagnostics`, `System.IO` a `System.Net`) a volání metod z výše popsané knihovny `HWInfo`. V aplikaci je interakce s touto třídou zajištěna voláním veřejných metod `Refresh()`, zajišťující aktualizaci údajů a `GetSystemInfo()`, vracející soubor získaných informací formou v rámci třídy definované struktury `SysInfo`.

Třída **`LoadMonitor`** je nástrojem monitorování zatížení zařízení. Po spuštění služby běží na samostatném vlákně a periodicky, v intervalu 60s zaznamenává aktuální stav vytížení do datového souboru (formát CSV) v pracovním adresáři. Každý záznam obsahuje následující informace: datum a čas měření, procentuální zatížení procesoru, teplotu procesoru, množství aktuálně využité paměti a obsazené místo na systémové jednotce. V souboru je udržováno posledních 1440 záznamů, tedy data za posledních 24 hodin běhu zařízení. Tato data jsou následně použita při generování kompletních reportů zařízení (na vyžádání v rámci samostatné úlohy).

Třída **`Reporting`** zodpovídá za periodické zasílání notifikací o stavu zapnutého tenkého klienta na Management Server, kterým je spravován, pro účely zobrazení aktuálního stavu zařízení uživateli. Systém notifikací pracuje na vlastním vlákně a každé 3 minuty volá metodu `SendClientNotification()` třídy `ClientAction`, která zajistí obnovení a získání aktuálních stavových informací od třídy `SystemInfo`, data

vloží do tabulky a odešle v XML formátu na server, kde je následně aktualizován záznam daného tenkého klienta v databázi.

Třída `JobAgent` obsahuje implementaci metod pro realizaci vlastních konfiguračních úloh. Tyto metody jsou již velmi specifické, využívají funkcionalitu obsaženou v knihovně `ClientCommand` a budou podrobněji popsány v další části práce. Dále jsou v rámci třídy definovány konstanty cest k registru povinného místního uživatelského profilu, pracovnímu adresáři uživatelského rozhraní Shell32 a proměnné pro řízení vykonávání samotné úlohy na separátním vlákně včetně blokace (třída umožňuje provádění pouze jedné konfigurační úlohy v čase, případné další požadavky jsou zamítány, dokud není aktuální úloha dokončena, nebo nevyprší).

5.4.4 Konfigurační úlohy tenkého klienta

Zpracování konfiguračních úloh je primárním účelem služby Management Agent. Provádění úloh je řízeno systémem správy úloh na serveru, který aktivním klientům dle plánu zasílá zprávy s požadavkem `JOB_PROCESS_REQ` (viz příloha č. 1), jejíž datová oblast obsahuje strukturu s názvem a parametry úlohy, a jejíž provedení je vyžadováno. Dále bude popsán návrh základních úloh, které bude aplikace schopna nad tenkými klienty provádět a způsob jejich technické realizace službou agenta.

Změna hesla místního účtu Administrator

Úloha `ADMIN_PASS` umožní změnit na tenkém klientovi heslo zabudovaného místního účtu "Administrator". Nastavení lze provést spuštěním příkazu net.exe prostřednictvím metod třídy `System.Diagnostics.Process` ve skrytém okně s následujícími parametry:

```
net.exe user administrator nove_heslo
```

Spuštění dávky příkazů jako správce

Úloha `BATCH_CMD` spustí zaslanou dávku příkazů lokálně na zařízení s právy správce. Parametry úlohy: dávka příkazů (řetězec – jeden či více řádků), časový limit na zpracování příkazů a informace o požadavku na propsání změn filtrem EWF. Z přijatého řetězce příkazů bude vytvořen dávkový *.bat soubor v dočasném úložišti a následně vytvořen skrytý proces příkazového řádku s oprávněním správce:

```
cmd.exe /c cesta_docasneho_bat_souboru
```

Proces bude sledován a po jeho dokončení zaslána odpověď serveru. V případě vypršení časového limitu bude proces ukončen a podáno hlášení o nedokončení úlohy.

Konfigurace připojení k relaci Citrix HDX

Úloha `CITRIX_CONFIG` zajistí konfiguraci tenkého klienta pro připojení k relaci vzdálené plochy Citrix HDX. Pro realizaci připojení prostřednictvím klienta Citrix Online Plugin je vyžadováno pouze otevření adresy serveru webovým prohlížečem, realizace úlohy tedy bude spočívat ve vytvoření (resp. odstranění) *.url souboru se zástupcem webové adresy VDI serveru v adresáři aplikací uživatelského rozhraní Shell32, odkud je již prezentován uživateli jako jedna z dostupných aplikací. Webová adresa VDI serveru musí být dále vložena do registru uživatelského profilu jako důvěryhodné umístění:

```
SOFTWARE\Microsoft\Windows\CurrentVersion\InternetSettings\ZoneMap\Ranges\*
```

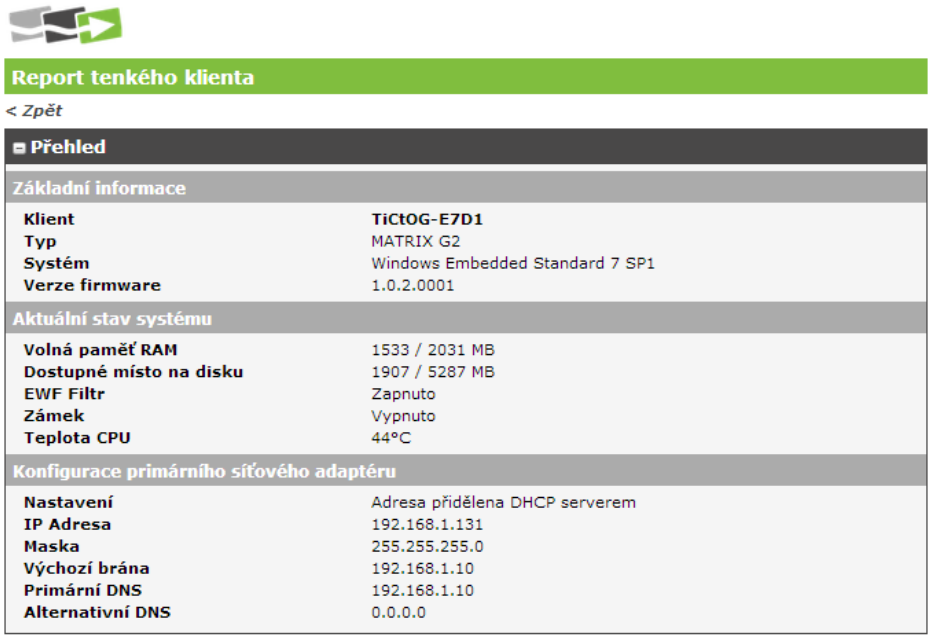
Přejmenování počítače (změna host name)

Úloha `IDENTIFICATION` změní hostitelský název počítače na požadovanou hodnotu pomocí volání metody `SetComputerNameEx()` rozhraní Windows API. Pro dokončení operace je vyžadován restart operačního systému.

Vygenerování reportu o zařízení

Úloha `COMPLEX_REPORTING` umožní na cílovém zařízení vygenerovat souhrnný report o jeho stavu a předat jej Management Serveru. Report je reprezentován XML strukturou obsahující stavová data tenkého klienta, získaná pomocí metod třídy `SystemInfo` a prostředků knihovny `ClientCommand.dll`. Obsah reportu sestává ze základních stavových údajů o zařízení (typ zařízení, verze systému, stav systémových prostředků a sítě), historie prováděných úloh, seznamu aplikací dostupných uživateli, dat o vytížení systému v posledních 24 hodinách a výpisu protokolů Management Agenta i systému Windows. Reporty budou následně na serveru uloženy do repositáře HTTP serveru a přístupné prostřednictvím vestavěného prohlížeče konzole. Vzhled a struktura výsledného dokumentu bude definován pomocí XSLT transformační šablony,

pro vykreslení grafů navrhuji použití JavaScript knihovny *jQuery Flot* (19). Vzhled šablony reportu je demonstrován na následujícím obrázku[11].



Obrázek 14 - Část reportu tenkého klienta (vlastní zpracování)

Konfigurace prohlížeče Internet Explorer

Úloha `IE_CONFIG` zajistí zpřístupnění prohlížeče Internet Explorer uživateli a nastavení domovské stránky. Zpřístupnění aplikace spočívá ve vytvoření zástupce (souboru *.lnk) v adresáři uživatelských aplikací a následném nastavení domovské stránky prostřednictvím hodnoty `Start Page` v klíči registru uživatelského profilu:

```
SOFTWARE\Microsoft\Internet Explorer\Main
```

Změna nastavení sítě

Úloha `NET_CONFIG` umožní změnit IPv4 konfiguraci primárního síťového adaptéru tenkého klienta na automatické nastavení, nebo statickou konfiguraci. Tato operace je proveditelná invokováním metod databáze WMI (`EnableDHCP` pro nastavení automatické konfigurace, případně metody `EnableStatic`, `SetGateways` a `SetDNSServerSearchOrder` pro vlastní statickou konfiguraci). Práce s metodami a položkami databáze WMI je v jazyce C# možná použitím třídy `System.Management`.

[11] Plná podoba reportu je uvedena v příloze č. 5.

Ovládání napájení

Úloha `POWER_OPERATION` provede na základě poskytnutých parametrů restart či vypnutí systému tenkého klienta – tato akce vyžaduje volání metody `ExitWindowsEx()` rozhraní Windows API (s požadovanými parametry), kterému však musí předcházet volání metody `AdjustTokenPrivileges()` a získání privilegia k vypnutí systému Windows procesem.

Konfigurace Microsoft Remote Desktop Services

Úloha `RDS_CONFIG` umožní zpřístupnit klienta MS Remote Desktop Services uživateli (s přednastavenou konfigurací připojení). Realizace spočívá ve vytvoření textového souboru *.rdp s konfigurací připojení (adresa serveru, název domény, uživatelské jméno a volitelně parametr přesměrování místních prostředků) v adresáři uživatelských aplikací. Server vzdálené plochy je následně přidán mezi důvěryhodné terminálové servery v registru uživatelského profilu pod následujícím klíčem:

```
SOFTWARE\Microsoft\Terminal Server Client\LocalDevices
```

Odhlášení tenkého klienta ze správy

Úloha `UNSUBSCRIBE` odhlásí tenkého klienta ze správy aktuálního Management Serveru. Odhlášení sestává z odstranění adresy Management Serveru z konfiguračního souboru agenta a zastavení zasílání periodických notifikací agentem na server. Po obdržení odpovědi o dokončení této úlohy Management Server odstraní položku tenkého klienta z databáze a zruší všechny naplánované úlohy pro toto zařízení.

Spuštění aktualizace operačního systému

Úloha `UPDATE_FW` iniciuje reinstalaci operačního systému z úložiště HTTP serveru. V prvním kroku bude vytvořen na systémové jednotce soubor *.shl s URL adresou instalačního souboru pro aplikaci *TiCtOG Updater*[12], která provede vlastní stažení a rozbalení obrazu systému. Druhým krokem je editace souboru `menu.lst` zavaděče GRUB a nastavení spuštění systému *WindowsPE* s aplikací *TiCtOG Updater*

[12] Aplikace *TiCtOG Updater* není součástí řešení správy a nebude zde podrobněji popisována.

ze separátního diskového oddílu pro provedení instalace. Následně je serveru podáno hlášení o provedení úlohy, systém restartován a započata automatická aktualizace.

Konfigurace připojení k relaci VMware View

Úloha `VIEW_CONFIG` zpřístupní uživateli předkonfigurovaného klienta aplikace VMware View pro připojení k relaci VDI. Na základě přijatých informací budou do registru uživatelského profilu zapsány hodnoty výchozího brokera (server VDI), uživatelského jména a domény. Cesta k upravovanému klíči je následující:

```
SOFTWARE\VMware, Inc.\VMware VDM\Client
```

Po uložení výchozí konfigurace bude na základě systémového registru vyhledán spustitelný soubor klienta VMware View a vytvořen zástupce *.lnk v adresáři uživatelských aplikací, čímž je uživateli zajištěn přístup k VDI aplikaci.

5.5 Management Console

Aplikace Management Console bude hlavním ovládacím prvkem celého řešení správy s účelem poskytnout uživateli (správci) přehledné grafické rozhraní pro přístup k datům, interakci s Management Serverem a zprostředkovaně i spravovanými tenkými klienty. Vzhledem k opětovnému využití již výše rozebraných technických řešení pro základní úlohy (jako je komunikace, práce s XML zprávami, konfigurace, protokolování apod.) bude tato kapitola zaměřena výhradně na popis navrženého grafického rozhraní a vlastní uživatelské funkcionality aplikace Management Console.

5.5.1 Přehled zajišťovaných funkcí

Aplikace Management Console poskytne na straně uživatele následující funkce:

- Přihlášení do aplikace pomocí uživatelského jména a hesla
- Zobrazení a interpretace dat získaných z Management Serveru
- Generování a zasílání příkazů na server pomocí prvků uživatelského rozhraní
- Prohlížení obsahu HTTP repositáře serveru a reportů
- Podpora více jazyků uživatelského rozhraní

5.5.2 Funkční základ aplikace

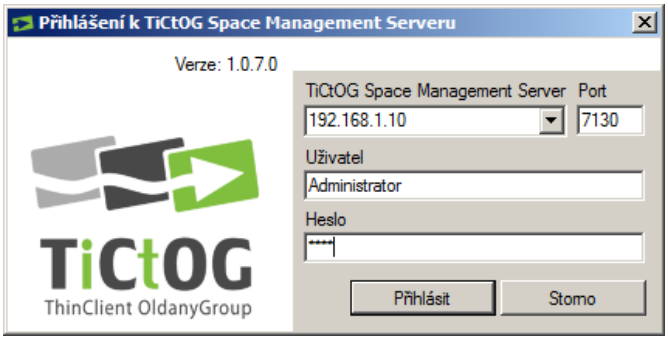
Aplikace správcovské konzole bude formulářovou aplikací systému Windows, založenou na architektuře MVC (Model-View-Controller). Pro řízení běhu podporcesů aplikace, základní úlohy a práci s daty budou využity stejné principy a metody jako u výše popsaných komponent (agenta a serveru), přičemž zásadní rozdíl spočívá v koncepci aplikace: již se nejedná o službu systému Windows, ale standardní aplikaci s možností přímé interakce uživatele. Pohled, implementaci formulářů a grafické uživatelské rozhraní (GUI) navrhuji realizovat pomocí grafického aplikačního rozhraní Windows Forms, poskytovaného platformou Microsoft .NET Framework.

5.5.3 Grafické rozhraní aplikace

Grafické rozhraní aplikace Management Console bude tvořit soubor účelově navržených formulářů pro prezentaci dat uživateli a dialogů, umožňujících zpracování jeho vstupů. Jednotlivé formuláře a jejich funkcionalita budou dále popsány.

Formulář přihlášení do aplikace

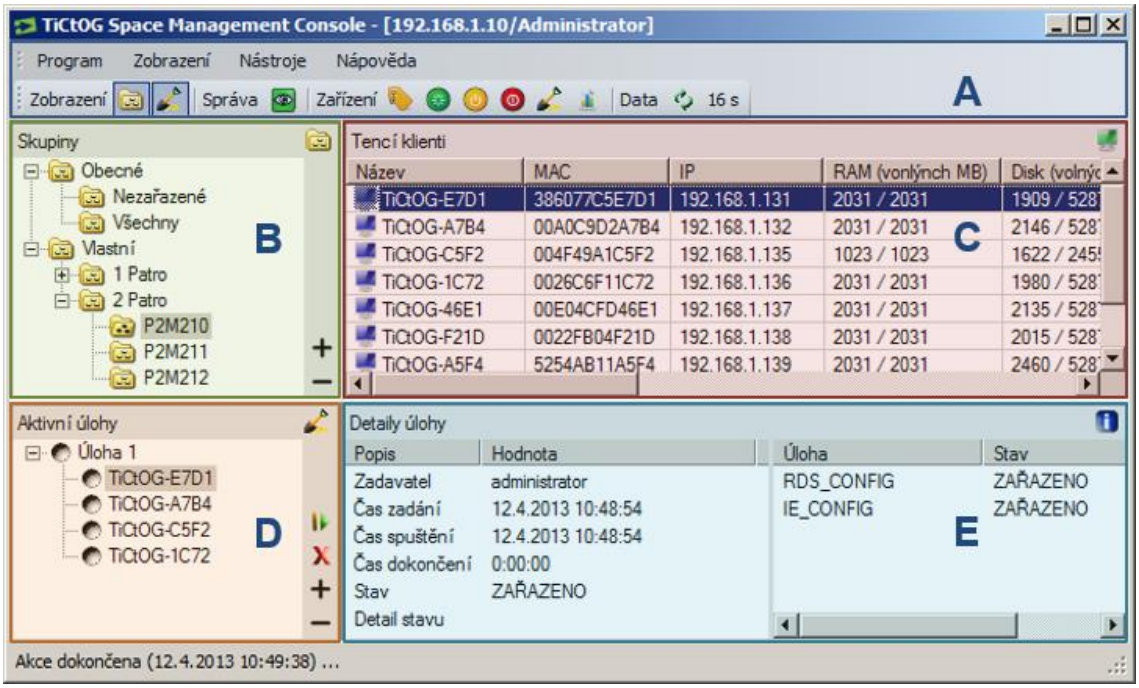
Vstupní formulář aplikace vyžaduje od uživatele vyplnění adresy serveru (podporováno je zadání IP adresy i DNS jména) a portu, na kterém služba běží, přihlašovacího jména a hesla. Tyto údaje jsou po vyplnění uchovávány v konfiguračním souboru (s výjimkou hesla). Při potvrzení jsou údaje zaslány na uvedený server a zahájen proces autentizace uživatele (popsán v kapitole Management Server), na jehož základě je uživateli umožněn (případně odepřen) přístup k hlavnímu formuláři aplikace.



Obrázek 15 - Přihlašovací formulář (vlastní zpracování)

Hlavní formulář aplikace

Hlavní formulář aplikace sestává ze souhrnného přehledu informací o položkách správy a ovládacích prvků. Pro názornost je v následujícím obrázku rozdělen do oblastí.



Obrázek 16 - Hlavní formulář aplikace (vlastní zpracování)

Oblast A obsahuje programová menu a hlavní ovládací prvky aplikace, zleva jsou to tlačítka pro přepínání zobrazení oblastí B a D/E, tlačítko pro vzdálený přístup (VNC), přiřazení popisku k zařízení, tlačítka vzdáleného spuštění/restartu/vypnutí, vyvolání editoru konfiguračních úloh a vyžádání reportu od vybraných zařízení. Posledním tlačítkem je manuální obnovení zobrazených dat (aplikace však aktualizuje zobrazená data i automaticky v intervalu 20 vteřin).

Oblast B umožňuje uživateli spravovat hierarchii skupin, rozřazovat tenké klienty, a na základě volby skupiny filtrovat zařízení zobrazená v oblasti C.

Oblast C je hlavním zdrojem informací: obsahuje stavové informace o tenkých klientech ve správě: jejich spuštění, údaje o prostředcích systému a konfiguraci, uživatelský popisek zařízení a další. Tato oblast funguje jako multi-výběrová tabulka s možností zvolit jedno zařízení, nebo množinu zařízení, na kterou se budou aplikovat příkazy správy z oblasti A, nebo příkazy kontextového menu.

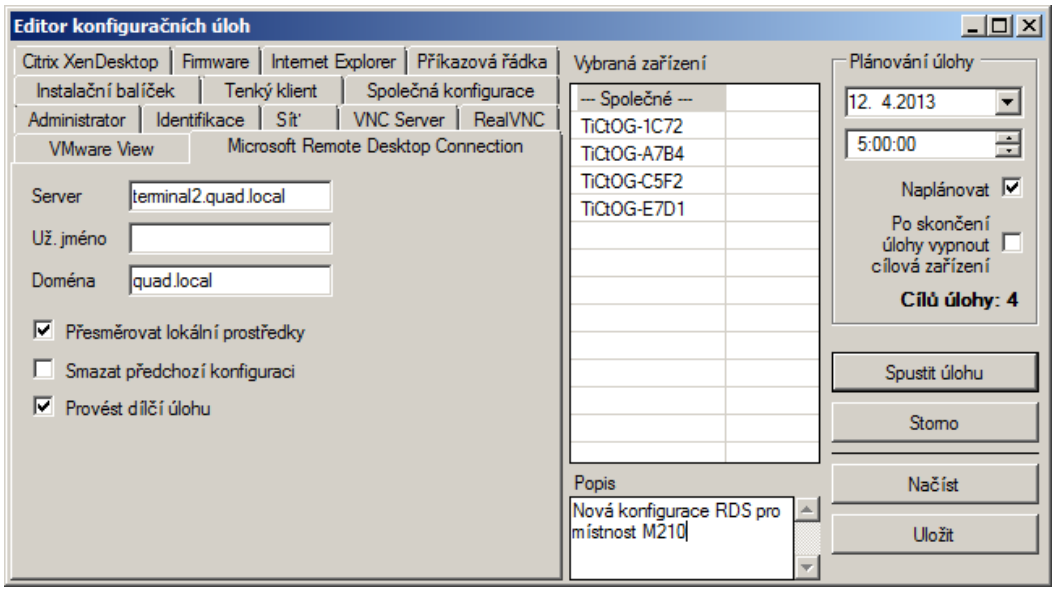
Oblast D je hierarchickým výpisem probíhajících a naplánovaných konfiguračních úloh, uložených na serveru. Umožňuje filtrovat detaily o úlohách zobrazené v oblasti E, dále obsahuje tlačítka pro zrušení úlohy a obnovení úlohy. Tlačítka "+" a "-" slouží k rozbalení/srolování hierarchie položek (využito i v oblasti B).

Oblast E zobrazuje detaily o úlohách zařízení, vybraného v oblasti D. Jsou zde uvedeny informace o průběhu zpracování dílčích příkazů konfigurační úlohy na cílovém zařízení, uživatelské jméno zadavatele úlohy, čas zadání, plánovaný čas spuštění, čas dokončení úlohy a případné detaily o stavu úlohy či chybová hlášení.

Programová menu a kontextové menu individuálních zařízení obsahují zejména příkazy již popsané v rámci oblasti A, nabízí tedy alternativní přístup k ovládání aplikace. Kontextové menu každého zařízení navíc nabízí možnost odhlášení zařízení ze správy a smazání záznamu z databáze, všechny ostatní položky tohoto menu jsou však dostupné i formou tlačítek na hlavním formuláři aplikace.

Editor konfiguračních úloh

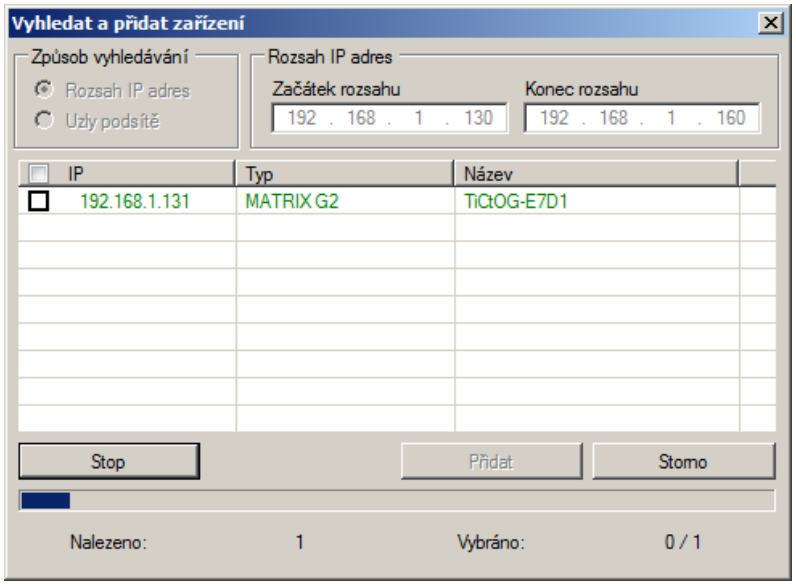
Formulář konfiguračních úloh je nástrojem pro vytvoření struktury konfigurační úlohy a její odeslání na server. V levé polovině formuláře jsou formou záložek zobrazeny dostupné konfigurační úlohy, které může uživatel pomocí zátržítka aktivovat a přidat tak do navrhovaného souboru úloh, včetně specifikace parametrů. Úlohy mohou být definovány nad společným kontejnerem a aplikovány na všechna vybraná zařízení, anebo parametrizovány individuálně, pro každé zařízení zvlášť. Úlohy mohou být pomocí položek v pravé části formuláře naplánovány na konkrétní čas, rozšířeny o příkazy vypnutí zařízení (po jejich dokončení) a dále odeslány na server ke zpracování, případně uloženy jako šablona pro budoucí použití (XML soubor s daty formuláře), resp. ze souboru opět načteny. Úlohy lze také opatřit popisem pro upřesnění – tento pak bude zobrazen v detailech úlohy v rámci hlavního formuláře.



Obrázek 17 - Editor konfiguračních úloh (vlastní zpracování)

Formulář vyhledávání zařízení

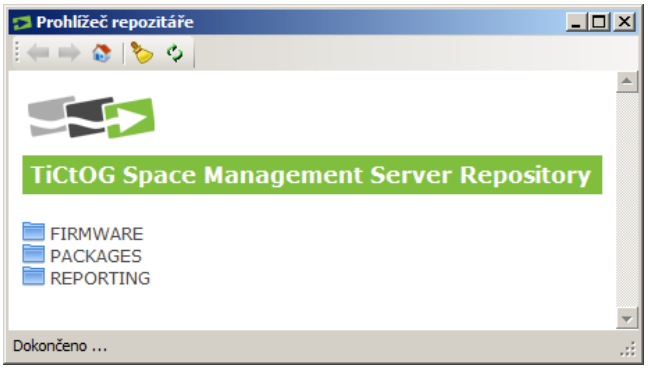
Ovládání vyhledávání zařízení na síti Management Serverem a přidání do správy je zajištěno pomocí formuláře, ve kterém lze definovat rozsah IP adres pro skenování, případně IP adresu a masku sítě, jejíž uzly mají být prohledány. Formulář je v průběhu vyhledávání pravidelně aktualizován a průběh indikován uživateli. Po vyhledání zařízení lze rozeslat požadavek na přihlášení tenkých klientů do správy serverem.



Obrázek 18 - Formulář vyhledávání zařízení (vlastní zpracování)

Prohlížeč repositáře HTTP serveru

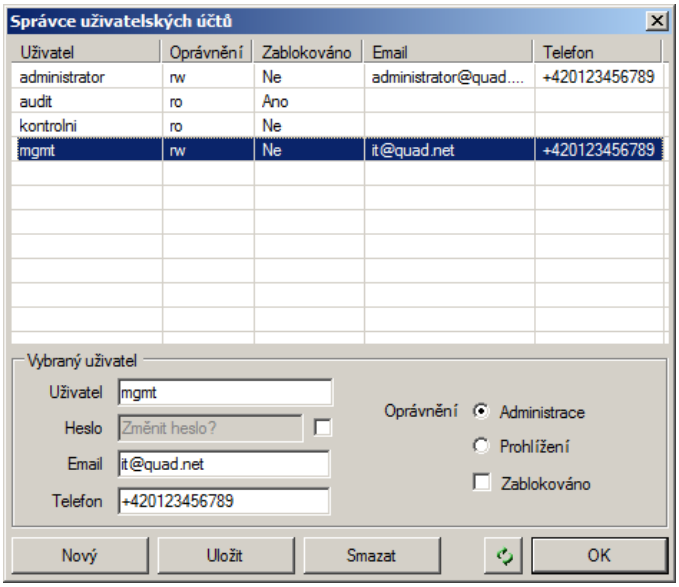
Prohlížeč repositáře je jednoduchý, účelově modifikovaný webový prohlížeč rozhraní Windows Forms. Jeho primárním účelem je umožnit uživateli prohlížení uložených reportů tenkých klientů a umožnit procházení cesty k souborům v repositáři pro některé úlohy (např. cesta k souboru s firmware pro úlohu aktualizace).



Obrázek 19 - Prohlížeč repositáře (vlastní zpracování)

Správa uživatelských účtů aplikace

Ve formuláři správy uživatelských účtů lze zobrazit a editovat uživatelské účty aplikace správy (za předpokladu dostatečných oprávnění). Pomocí poskytnutých ovládacích prvků je možno vytvořit nový účet, smazat, upravit údaje či zablokovat účet.



Obrázek 20 - Správa uživatelských účtů (vlastní zpracování)

5.6 Instalace aplikace

Jedním z požadavků na návrh aplikace je také vytvoření uživatelsky jednoduchého způsobu nasazení aplikace a instalace dílčích komponent – Management Agenta, Serveru a Console. Řešení bude popsáno individuálně pro každou komponentu.

Management Agent

Služba Management Agent bude součástí firmware tenkého klienta, její instalační logika bude proto zakomponována do řešení automatizovaného nasazení customizovaného obrazu systému Windows Embedded. Realizace instalace služby agenta spočívá v nakopírování souborů aplikace do programového adresáře cílového zařízení (velikost cca 1MB) a následném spuštění příkazu InstallUtil (.NET Framework Installation Utility) s parametrem cesty k spustitelnému souboru služby. Po dokončení příkazu je služba vytvořena a při dalším startu systému již bude automaticky spuštěna.

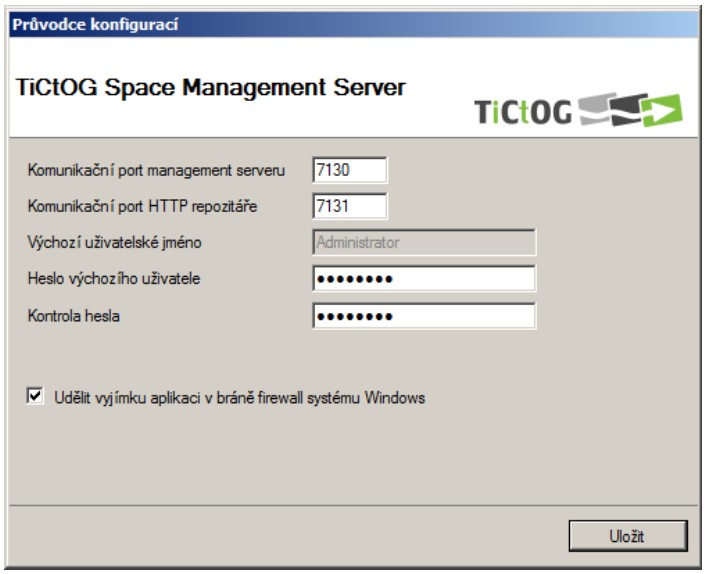
Management Console

Aplikace Management Console bude instalována uživatelem, proto musí existovat možnost manuální konfigurace a instalace s průvodcem. S ohledem na požadavky navrhuji použít instalační program *Microsoft Windows Installer*, jehož instalační balíky (*.msi) lze snadno vygenerovat v rámci instalačního projektu nástroje Visual Studio. Průvodce instalace bude sestaven z následujících dialogů: uvítací obrazovka, licenční smlouva, volba cílového adresáře, potvrzení instalace a ukončení. Velikost aplikace je cca 4MB.

Management Server

Instalace služby serveru využije stejného řešení, jako instalace aplikace Management Console. Navíc je zde však nutno vytvořit systémovou službu, předkonfigurovat parametry serveru a udělit aplikaci výjimku pro příchozí spojení v rámci brány firewall. K provedení těchto akcí bude sloužit formulářová aplikace ***ConfigWizard***, která bude spuštěna po dokončení kopírování souborů služby serveru na disk (velikost cca 3.5MB). Tato aplikace vygeneruje na základě vyplněných údajů (s použitím dříve vytvořených knihoven a metod) konfigurační soubor, upraví záznam výchozího uživatelského účtu v databázi a následně přidá pravidlo brány firewall

systému Windows pomocí funkcí systémové COM knihovny `NetFwTypeLib.dll`. Posledním krokem je vytvoření a spuštění služby, zde navrhuji použití metod automatizované instalace služeb, obsažené ve třídě `ManagedInstallerClass` platformy Microsoft .NET Framework.



Obrázek 21 - Konfigurační dialog instalace serveru (vlastní zpracování)

5.7 Ekonomické zhodnocení projektu

V této části diplomové práce bude nastíněn způsob realizace projektu návrhu a vlastního vývoje výše popsané aplikace. Projekt bude dále zhodnocen z pohledu časové náročnosti, souvisejících nákladů a v závěru posouzeny přínosy pro společnost vyplývající z jeho realizace.

5.7.1 Realizace projektu, metodika vývoje a časový harmonogram

Vzhledem k malému rozsahu projektu, stanoveným požadavkům na funkcionalitu výsledné aplikace a časovým možnostem byl pro jeho realizaci vytvořen malý team, sestávající ze dvou členů, zodpovídající zároveň za jeho průběh a řízení. Časový rámec aktivit projektu byl stanoven na šest měsíců a dále rozdělen na dvě celistvé, oddělené etapy o délce tří měsíců, jejichž účelem bylo dosáhnout stanovených milníků (základní funkční verze řešení již v polovině předpokládaného času), a zároveň

přizpůsobit projekt časovým možnostem členů projektového teamu. Časový harmonogram realizace a milníky projektu jsou dále podrobněji popsány formou tabulek.

Aktivita\Období	Měsíc 1				Měsíc 2				Měsíc 3				Měsíc 4				Měsíc 5				Měsíc 6			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Analýza a návrh základního konceptu aplikace																								
Implementace společných komponent aplikace																								
Implementace základní funkcionality služby serveru																								
Implementace základní funkcionality služby agenta																								
Implementace zákaldní funkcionality GUI aplikace																								
Testování a ladění																								
Revize návrhu a plánování dalšího vývoje																								
Implementace rozšiřující funkcionality aplikace																								
Návrh a implementace instalačních procesů																								
Testování a ladění																								
Vydání aplikace a nasazení v produkčním prostředí																								

Tabulka 10 – Časový harmonogram projektu (vlastní zpracování)

Návrh aplikace byl obecně formulován na začátku projektu a postupně rozpracováván, upřesňován a upravován v průběhu vývoje jednotlivých komponent a funkcí celého řešení, jejichž implementace na tento návrh bezprostředně navazovala. Vzhledem k předpokládané potřebě průběžných změn v návrhu, participace zákazníka (společnosti OldanyGroup) na projektu a zaměření na cílovou funkcionalitu, byla pro vývoj aplikace zvolena ***agilní metodika FDD – Feature Driven Development*** (20). Tato metodika zavádí iterativní, inkrementální proces vývoje a umožňuje dynamicky řídit projekt vývoje na základě cílových funkcionalit a vlastností produktu. Metodika je na rozdíl od jiných vývojových metodologií dobře škálovatelná a lze ji úspěšně aplikovat i na velmi malých projektech a týmech, je tedy vhodná i pro zde popisovaný scénář projektu návrhu a vývoje aplikace.

Milník	Termín splnění
Zahájení projektu	1. měsíc, týden 1
Základní návrh a koncept aplikace	1. měsíc, týden 4
Funkční základ aplikace a vytvoření dokumentace	3. měsíc, týden 1
Ověření funkčnosti a předání řešení se základní funkcionalitou	3. měsíc, týden 4
Zhodnocení současného stavu, revize a plán dalšího vývoje	4. měsíc, týden 2
Vývoj funkcionality aplikace (splnění definovaných požadavků)	5. měsíc, týden 4
Vytvoření instalačních procesů a testování finálního řešení	6. měsíc, týden 3
Předání hotového řešení pro nasazení v produkci a ukončení projektu	6. měsíc, týden 4

Tabulka 11 - Milníky projektu (vlastní zpracování)

5.7.2 Náklady projektu

Uskutečnění výše popsaného projektu je spojeno se vznikem určitých nákladů, které musí firma během jeho realizace vynaložit. Největší položkou jsou zde náklady související se mzdovým ohodnocením práce na vlastním vývoji aplikace – v přehledu dále uváděná hodnota vyplývá z odhadovaných 2x480 hodin při průměrné hodinové sazbě 120 Kč, vyplácených pracovníkům průběžně formou dohody o provedení práce[13]. Další položkou rozpočtu jsou vývojářské nástroje, tyto však již společnost vlastní, v přehledu jsou pro informaci zahrnuty, avšak ohodnoceny částkou 0. Ostatní náklady sestávají z hardware (3x tenký klient) a software (3x licence operačního systému) pro účely vytvoření testovací infrastruktury, za předpokladu využití společností již vlastněných aktivních prvků a sítě. Testovací HW a SW (tencí klienti) může být po skončení projektu opět prodán. Do budoucna lze dále očekávat vznik dodatečných nákladů, souvisejících s budoucími aktualizacemi a opravami aplikace, tyto však zatím nelze posoudit, pro úplnost jsou však v přehledu uvedeny. Přehled nákladů je zobrazen v následující tabulce.

[13] Vzhledem k výslednému počtu v průměru 80 hodin/měsíc na jednoho pracovníka a stanovené hodinové sazbě spadají jednotlivé dohody o provedení práce do kategorie <10.000 Kč, nepodléhají tedy odvodu sociálního a zdravotního pojištění. Rozdělením projektu do dvou časově oddělených etap zároveň dochází k vyplacení druhé poloviny celkového počtu hodin v rámci dalšího kalendářního roku, je tedy dodržen limit nejvýše 300 odpracovaných hodin v kal. roce.

Položka	Cena
Práce na vývoji aplikace	115 200 Kč
Vývojářské nástroje*[14]	0 Kč
Hardware pro účely testování	15 000 Kč
Software pro účely testování	6 000 Kč
Budoucí náklady na aktualizace a opravy*[15]	0 Kč
Celkem	**136 200 Kč**

Tabulka 12 - Přehled nákladů projektu (vlastní zpracování)

5.7.3 Přínosy pro firmu

Přínosy realizace tohoto projektu nelze vzhledem k povaze výstupu přímo kvantifikovat – jedná se o podpůrnou aplikaci k již existujícímu nabízenému produktu, jejíž poskytování nebude zpoplatněno, nelze tedy přímo posuzovat očekávané výnosy z této investice. Pro firmu však ze zavedení této aplikace vyplývají zejména následující přínosy a výhody:

- **Zlepšení konkurenceschopnosti produktu tenkého klienta** – poskytnutím vlastního bezplatného řešení centrální správy se výrazně zvýší úroveň a konkurenceschopnost celého řešení tenkého klienta *TiCtOG*.
- **Zvýšení tržeb** – projekt přispěje ke zvýšení tržeb z prodeje tenkých klientů: bude splněn požadavek některých zákazníků, kteří by v případě absence možnosti centrální správy pravděpodobně volili konkurenční řešení tenkého klienta. Jednoduchá správa a bezproblémový provoz zařízení může zároveň podpořit rozhodnutí stávajícího zákazníka k nákupu dalších jednotek.
- **Usnadnění práce na vlastních projektech –** aplikace usnadní zaměstnancům společnosti práci na vlastních projektech implementace VDI u zákazníka – díky možnosti hromadné konfigurace a instalace operačního systému mohou ušetřit řádově až desítky minut času na každé instalované zařízení.
- **Vyšší hodnota produktu pro zákazníka** – zákazník získává nákupem tenkých klientů *TiCtOG* navíc plnohodnotný nástroj pro jejich správu a konfiguraci.

[14] Vývojářské nástroje společnost již vlastní.
[15] Náklady budoucích aktualizací zatím nelze posoudit.

6 Zhodnocení a závěr

Cílem této diplomové práce bylo navrhnout a posléze vytvořit na základě zvolených technologií a stanovených požadavků aplikaci pro centralizovanou správu tenkých klientů *TiCtOG*, nabízených společností OldanyGroup s.r.o.

Primárním účelem této aplikace je poskytnout nástroje pro sledování stavu a konfiguraci zařízení, nasazených ve firemní síti zákazníka. Další motivací pro realizaci tohoto projektu společností pak byla potřeba zvýšení konkurenceschopnosti celého řešení tenkého klienta, vzhledem k rozšiřujícímu se trendu poskytování bezplatných řešení centrální správy konkurenčními výrobci tenkých klientů. V neposlední řadě má zavedení této aplikace pro firmu potenciál zvýšení tržeb z prodejů tenkých klientů a zjednodušení konfiguračních úkonů při instalaci těchto zařízení u zákazníka.

S ohledem na definované požadavky byl vytvořen návrh a následně vyvíjena a implementována požadovaná funkcionalita aplikace, sestávající z komponent serveru správy, agenta instalovaného na tenkých klientech a aplikace ovládací konzole. Projekt vývoje byl realizován dvoučlenným týmem v rámci půlročního časového období, na jehož závěru byla společnosti předána hotová, plně funkční aplikace.

V rámci diplomové práce navržená aplikace centrální správy tenkých klientů splňuje všechny požadavky definované společností, umožňuje efektivně spravovat, monitorovat a řídit konfigurace nasazených tenkých klientů *TiCtOG*, a to s minimálními nároky na hardwarové a softwarové vybavení, při zachování jednoduchého a intuitivního ovládání. Popsané softwarové řešení je plně funkční a dostačující pro aktuální situaci, v budoucnu však lze předpokládat další rozvoj této aplikace, rozšíření o nové funkce, opravy chyb a případné další aktualizace.

Seznam použité literatury

(1) OLDANYGROUP. *Virtualizace, konzultace a správa pro řešení na VMware vSphere, Veeam, Linux a Windows | OldanyGroup* [online]. 2012 [cit. 2012-12-27]. Dostupné z: http://www.oldanygroup.cz

(2) JUHAŇÁK, F. *Návrh tenkého klienta*. Brno, 2011. Bakalářská práce. Vysoké učení technické v Brně, Fakulta podnikatelská.

(3) Dell Wyse Device Manager. WYSE TECHNOLOGY INC. *Dell Wyse | The Global Leader in Cloud Client Computing* [online]. 2012 [cit. 2012-12-28]. Dostupné z: http://www.wyse.com/products/software/management/WDM

(4) HP Device Manager - Overview. HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. *HP Business Support Center* [online]. 2012 [cit. 2012-12-28]. Dostupné z: http://h20000.www2.hp.com/bizsupport/TechSupport/Document.jsp?objectID=c01490790

(5) Universal Management Suite. IGEL TECHNOLOGY. *Thin Clients Software & Hardware: IGEL Technology* [online]. 2012 [cit. 2012-12-28]. Dostupné z: http://www.igel.com/products/software/universal-management-suite-ums.html

(6) VSpace Management Center. NCOMPUTING. *NComputing | Desktop Virtualization for Enterprise, Education, SMB, Government* [online]. 2012 [cit. 2012-12-28]. Dostupné z: http://www.ncomputing.com/products/vSpace-management-center-for-Nseries

(7) ARGAWAL, Neeraj et al. *Practical Handbook Of Thin-Client Implementation*. New Delhi: New Age International, 2005. ISBN 81-224-1685-3.

(8) MICROSOFT. *Windows Embedded | Evaluate & Develop Microsoft | Intelligent Systems* [online]. 2012 [cit. 2012-12-31]. Dostupné z: http://www.microsoft.com/windowsembedded/en-us/windows-embedded.aspx

(9) REYNDERS, Deon a Edwin WRIGHT. *Practical TCP/IP and Ethernet networking*. London: Elsevier, 2003. ISBN 07-506-5806-1.

(10) ŠMRHA, Pavel. *Internetworking pomocí TCP/IP*. České Budějovice: KOPP, 1994. ISBN 80-858-2809-X.

(11) ELMASRI, Ramez a Sham NAVATHE. *Fundamentals of database systems*. 6th ed. Boston: Addison-Wesley, 2011. ISBN 01-360-8620-9.

(12) KREIBICH, Jay A. *Using SQLite*. 1st ed. Sebastopol, CA: O'Reilly, 2010. ISBN 05-965-2118-9.

(13) ECKSTEIN, Robert a Michel CASABIANCA. *XML pocket reference*. 2nd ed. Sebastopol, CA: O'Reilly, 2001. ISBN 05-960-0133-9.

(14) *JSON* [online]. 2012 [cit. 2013-03-05]. Dostupné z: http://www.json.org/json-cz.html

(15) MICROSOFT. *MSDN – the Microsoft Developer Network* [online]. 2012 [cit. 2012-12-27]. Dostupné z: http://msdn.microsoft.com/en-US

(16) RICHTER, Jeffrey. *CLR via C#*. 3rd ed. Sebastopol: Microsoft Press, 2010. ISBN 07-356-4281-8.

(17) VERMA, Dinesh C. *Principles of computer systems and network management*. London: Springer, 2009. ISBN 03-878-9008-4.

(18) ORTEGA, R. Sample HTTP Server Skeleton in C#. *CodeProject - For those who code* [online]. 2007 [cit. 2013-03-01]. Dostupné z: http://www.codeproject.com/Articles/17071/Sample-HTTP-Server-Skeleton-in-C

(19) *Flot: Attractive JavaScript plotting for jQuery* [online]. 2012 [cit. 2013-04-16]. Dostupné z: http://www.flotcharts.org

(20) NEBULON PTY. LTD. *Feature Driven Development | The portal for all things FDD* [online]. 2012 [cit. 2013-05-04]. Dostupné z: http://www.featuredrivendevelopment.com

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

ACID – Atomic, Consistent, Isolated, Durable
ADO – ActiveX Data Objects
API – Application Programming Interface
ARP – Address Resolution Protocol
BCL – Base Class Library
C2S – Client-to-Server
CBC – Cipher-block Chaining
CIDR – Class Inter-domain Routing
CIL – Common Intermediate Language
CLI – Common Language Infrastructure
CLR – Common Language Runtime
CMS – Content Management System
CSS – Cascading Style Sheets
CSV – Comma-separated Values
DB – Database
DBMS – Database Management System
DCL – Data Control Language
DDL – Data Definition Language
DES – Data Encryption Standard
DHCP – Dynamic Host Configuration Protocol
DML – Data Manipulation Language
DNS – Domain Name System
DOM – Document Object Model
DTD – Document Type Definition
EWF – Enhanced Write Filter
FCL – Framework Class Library
FDD – Feature Driven Development
FK – Foreign Key
FTP – File Transfer Protocol
GUI – Graphical User Interface
HTML – HyperText Markup Language
HTTP – HyperText Transfer Protocol
HW – Hardware
ICMP – Internet Control Message Protocol
ICT – Information and Communications Technology
IP – Internet Protocol
ISO – International Organization for Standardization
IT – Information Technology
JSON – JavaScript Object Notation
M2S – Management-to-Server
MAC – Media Access Control
MS – Microsoft
MVC – Model-View-Controller
OEM – Original Equipment Manufacturer
OSI – Open Systems Interconnection
PC – Personal Computer
PDU – Protocol Data Unit
PK – Primary Key
RAM – Random Access Memory
RDBMS – Relational Database Management System
RPC – Remote Procedure Call
S2C – Server-to-Client
S2M – Server-to-Management
SAX – Simple API for XML
SCM – Service Control Manager
SGML – Standard Generalized Markup Language
SMTP – Simple Mail Transfer Protocol
SQL – Structured Query Language
SSD – Solid-state Drive
SW - Software
TCP – Transmission Control Protocol
UDP – User Datagram Protocol
URL – Uniform Resource Locator
VDI – Virtual Desktop Infrastructure
VNC – Virtual Network Computing
VS – Visual Studio
W3C – World Wide Web Consortium
WMI – Windows Management Instrumentation
XDR – External Data Representation
XML – Extensible Markup Language
XSLT – Extensible Stylesheet Language Transformations

Seznam příloh

Příloha č. 1: Přehled interakcí v XML komunikaci

Popis akcí pro XML zprávy typu M2S, zasílané z aplikace konzole na server:

Název akce (M2S)	Popis
AUTH_REQ	Požadavek uživatele konzole na autentizaci
MGMT_VIEW_REFRESH_REQ	Vyžádání aktuálních dat pro zobrazení v konzoli
DISCOVERY_START_REQ	Spuštění vyhledávání tenkých klientů na síti
DISCOVERY_STOP_REQ	Zastavení procesu vyhledávání
DISCOVERY_PULL_REQ	Žádost o průběžný výsledek vyhledávání
DISCOVERY_ADD_REQ	Žádost o přidání nalezeného zařízení do správy
JOB_ENQUEUE_REQ	Zadání/naplánování konfigurační úlohy
JOB_OPERATION_REQ	Pozastavení/obnovení konfigurační úlohy
REPORTS_CLEANUP_REQ	Odstranění reportů z HTTP repositáře
CLIENTS_REPORT_REQ	Vyžádání reportu od vybraných zařízení
CLIENTS_TABLE_REQ	Žádost o data z tabulky klientů
CLIENTS_TABLE_*	Metody pro práci s daty v tabulce
GROUPS_TABLE_REQ	Žádost o data z tabulky skupin
GROUPS_TABLE_*	Metody pro práci s daty v tabulce
JOBS_TABLE_REQ	Žádost o data z tabulky konfiguračních úloh
USERS_TABLE_REQ	Žádost o data z tabulky uživatelů
USERS_TABLE_*	Metody pro práci s daty v tabulce

Popis akcí pro XML zprávy typu S2M, zasílaných v odpovědi na výše uvedené požadavky:

Název akce (S2M)	Popis
AUTH_RES	Odpověď na požadavek autentizace uživatele
ACTION_REPORT	Univerzální odpověď o provedení akce (bez dat)
DISCOVERY_PULL_RES	Zaslání tabulky s průběžným výsledkem vyhledávání
MGMT_VIEW_REFRESH_RES	Zaslání aktuálních dat pro zobrazení v konzoli
CLIENTS_TABLE_RES	Zaslání dat z tabulky klientů
GROUPS_TABLE_RES	Zaslání dat z tabulky skupin
JOBS_TABLE_RES	Zaslání dat z tabulky konfiguračních úloh
USERS_TABLE_RES	Zaslání dat z tabulky uživatelů

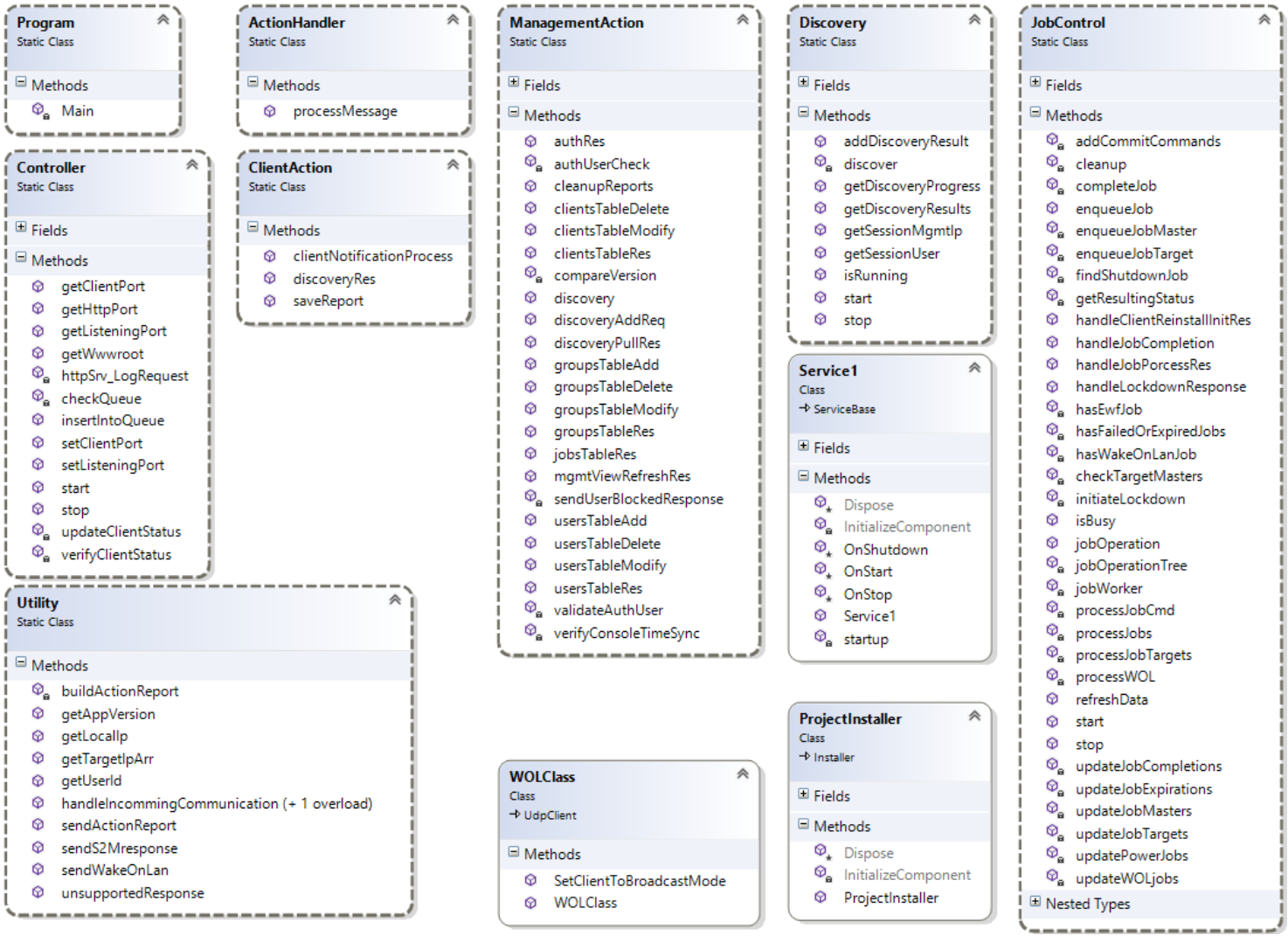
Popis akcí pro XML zprávy typu S2C, zasílaných serverem jako příkazy na tenké klienty:

Název akce (S2C)	Popis
DISCOVERY_REQ	Dotaz na základní informace o zařízení při vyhledávání
DISCOVERY_ADD_REQ	Požadavek na přihlášení zařízení ke správě tímto serverem
INIT_LOCKDOWN_REQ	Požadavek na uzamčení tenkého klienta
PRIORITY_REBOOT_REQ	Požadavek na provedení restartu s nejvyšší prioritou
JOB_PROCESS_REQ	Požadavek na zpracování zaslané konfigurační úlohy

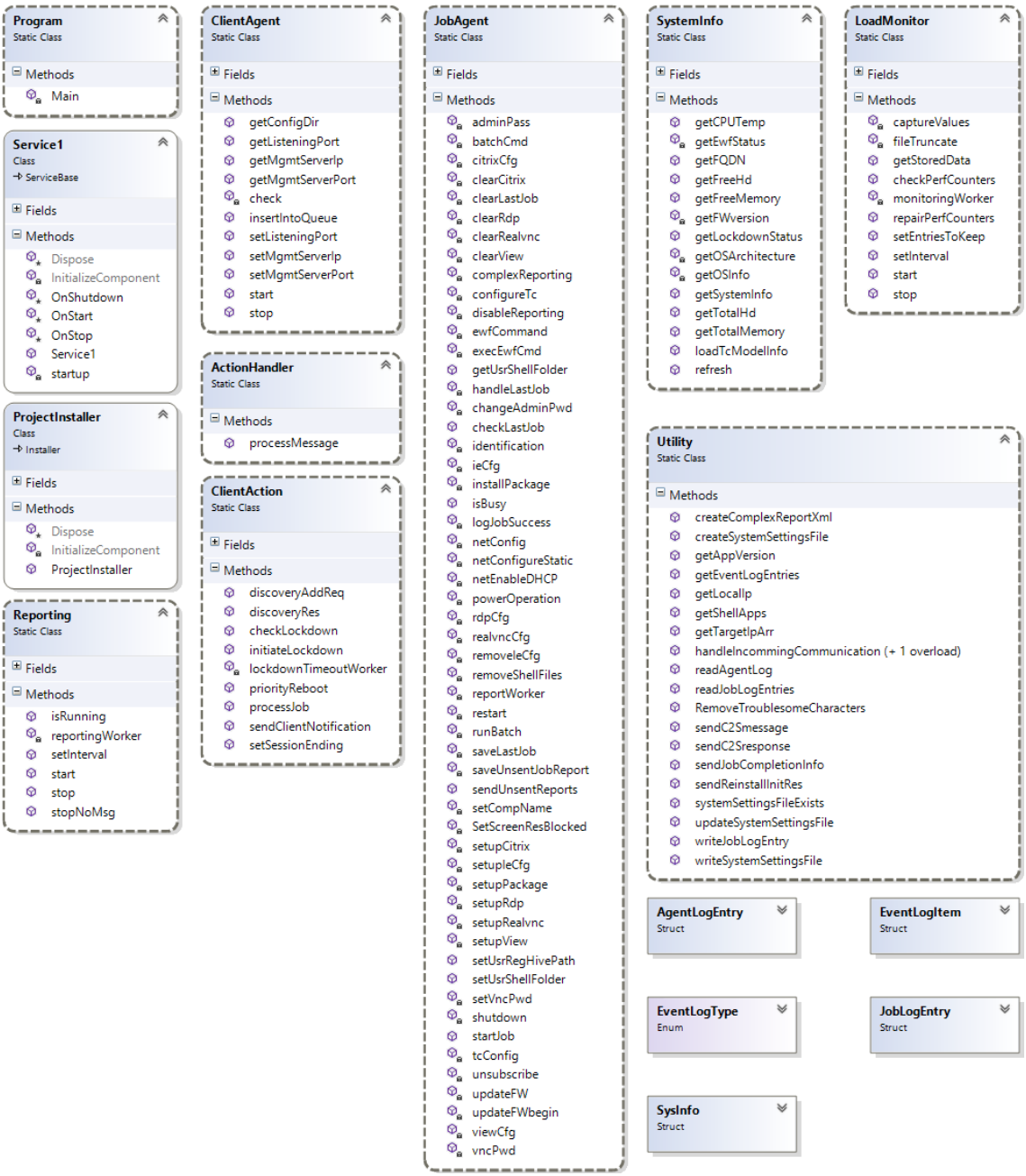
Popis akcí pro XML zprávy typu C2S, odpovědi tenkého klienta na požadavky zaslané serverem a výstupy konfiguračních úloh:

Název akce (C2S)	Popis
DISCOVERY_RES	Odpověď na požadavek při vyhledávání zařízení serverem
INIT_LOCKDOWN_RES	Potvrzení o uzamčení zařízení
REINSTALL_INIT_RES	Oznámení klienta o zahájení reinstalace systému
JOB_PROCESS_RES	Potvrzení požadavku na provedení konfigurační úlohy
JOB_COMPLETION_INFO	Oznámení klienta o průběhu zadané konfigurační úlohy
CLIENT_NOTIFICATION	Pravidelně zasílaná notifikace se stavovými údaji zařízení
COMPLEX_REPORT	Zaslání vyžádaného detailního reportu o stavu zařízení

Příloha č. 2: Diagram tříd a metod služby Management Server



Příloha č. 3: Diagram tříd a metod služby Management Agent



Příloha č. 4: Diagram formulářů, tříd a modulů aplikace Management Console

Příloha č. 5: Report tenkého klienta

Report tenkého klienta

< Zpět

Přehled

Základní informace

Klient	**TiCtOG-E7D1**
Typ	MATRIX G2
Systém	Windows Embedded Standard 7 SP1
Verze firmware	1.0.2.0001

Aktuální stav systému

Volná paměť RAM	1533 / 2031 MB
Dostupné místo na disku	1907 / 5287 MB
EWF Filtr	Zapnuto
Zámek	Vypnuto
Teplota CPU	44°C

Konfigurace primárního síťového adaptéru

Nastavení	Adresa přidělena DHCP serverem
IP Adresa	192.168.1.131
Maska	255.255.255.0
Výchozí brána	192.168.1.10
Primární DNS	192.168.1.10
Alternativní DNS	0.0.0.0

Vytížení systému

Historie úloh

RDS_CONFIG	COMPLETED	11.4.2013 11:34:45
VIEW_CONFIG	COMPLETED	11.4.2013 11:35:15
IE_CONFIG	COMPLETED	11.4.2013 11:36:16

Uživatelské aplikace

RDS trm1.quad.local
VMware View Client
Internet Explorer

Výpis protokolu služby TiCtOG Agent

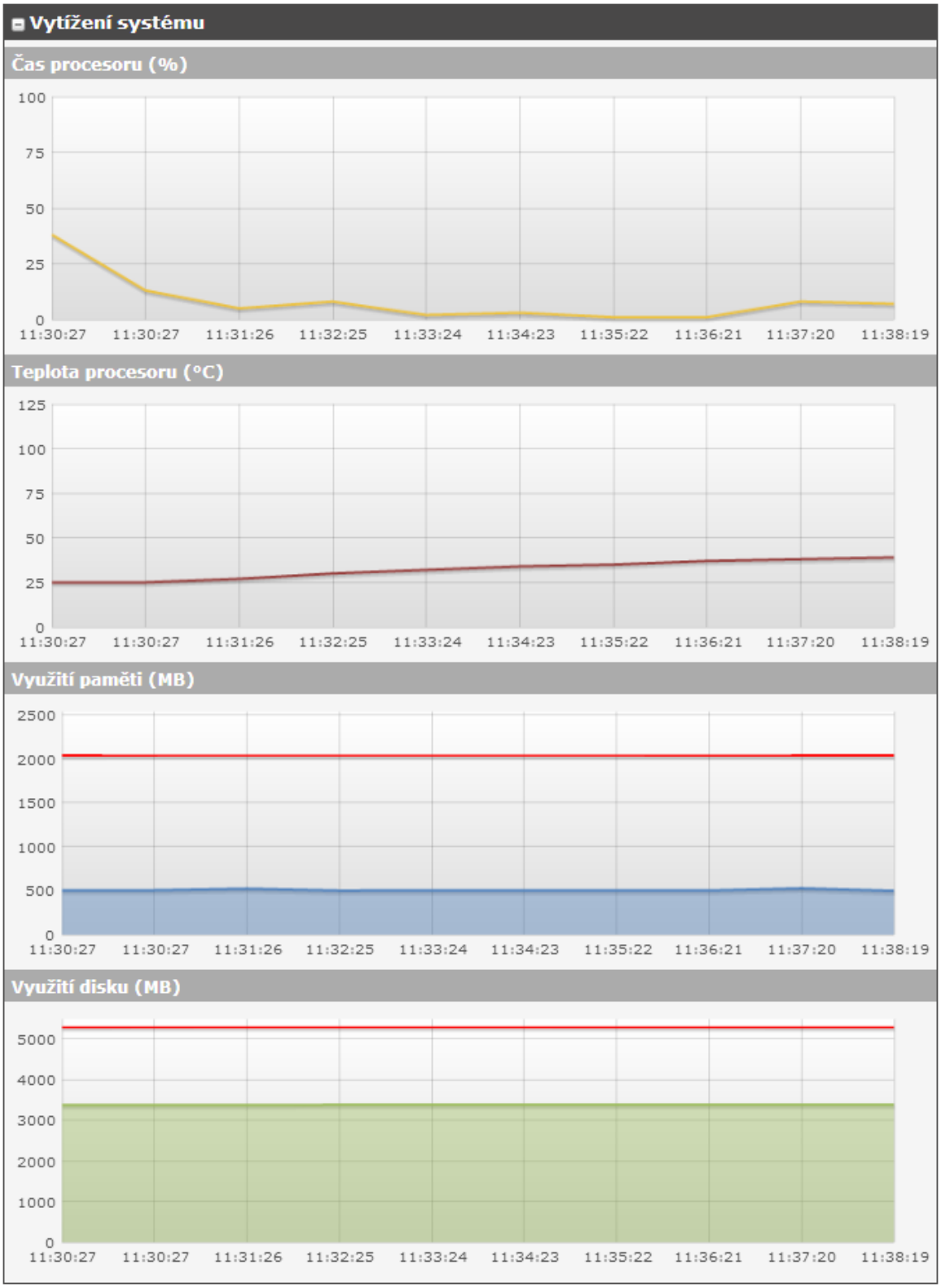
INFO	Creating VMware View configuration...	2013.04.11 11:35:14
INFO	Job processed successfully.	2013.04.11 11:35:15
INFO	Sending client notification...	2013.04.11 11:35:48
INFO	Setting user IE configuration...	2013.04.11 11:36:15
INFO	Job processed successfully.	2013.04.11 11:36:16
INFO	Generating complex report...	2013.04.11 11:38:47

Protokol systému Windows

Systémové události

Aplikační události

Vygenerováno 11.4.2013 11:38:48


Vytížení systému
Čas procesoru (%)
100
75
50
25
0
11:30:27
11:30:27
11:31:26
11:32:25
11:33:24
11:34:23
11:35:22
11:36:21
11:37:20
11:38:19
Teplota procesoru (°C)
125
100
75
50
25
0
11:30:27
11:30:27
11:31:26
11:32:25
11:33:24
11:34:23
11:35:22
11:36:21
11:37:20
11:38:19
Využití paměti (MB)
2500
2000
1500
1000
500
0
11:30:27
11:30:27
11:31:26
11:32:25
11:33:24
11:34:23
11:35:22
11:36:21
11:37:20
11:38:19
Využití disku (MB)
5000
4000
3000
2000
1000
0
11:30:27
11:30:27
11:31:26
11:32:25
11:33:24
11:34:23
11:35:22
11:36:21
11:37:20
11:38:19