VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VYTVOŘENÍ GUI NÁVRHÁŘE PRO ITP (INELS TOUCH PANELS)

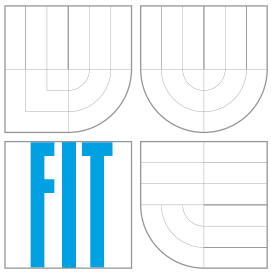
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ONDŘEJ KUČERA

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

VYTVOŘENÍ GUI NÁVRHÁŘE PRO ITP (INELS TOUCH PANELS)

CREATION OF GUI DESIGNER FOR ITP (INELS TOUCH PANELS)

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. ONDŘEJ KUČERA
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. MARTIN DRAHANSKÝ, Ph.D.
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce se věnuje problematice vytvoření grafického návrháře uživatelského rozhraní aplikace pro mobilní zařízení, umožňující ovládání systému inteligentní elektroinstalace. Popisuje konkrétní řešení inteligentní elektroinstalace poskytované společností ELKO EP, s.r.o. a obsahuje detailní návrh prototypu aplikace pro mobilní zařízení a aplikace grafického návrháře.

Abstract

This thesis is concerned with creation of designer for graphical user interface used in mobile devices. The application for these devices will be used for controlling system of intelligent electroinstallation. The thesis describes an existing solution of an intelligent electroinstallation provided by ELKO EP, s.r.o. It also contains detailed design of a prototype of an application to be used on mobile device and a desktop application used for graphical design.

Klíčová slova

inteligentní elektroinstalace, mobilní zařízení, Android, grafické uživatelské rozhraní, Java, NetBeans Platform, Eclipse

Keywords

intelligent electroinstallation, mobile devices, Android, graphical user interface, Java, NetBeans Platform, Eclipse

Citace

Ondřej Kučera: Vytvoření GUI návrháře pro iTP
(iNELS Touch Panels), diplomová práce, Brno, FIT VUT v Brně, 2011

Vytvoření GUI návrháře pro iTP (iNELS Touch Panels)

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Martina Drahanského, Ph.D.
Další informace mi poskytl pan Jiří Konečný, jednatel společnosti ELKO EP, s.r.o.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Kučera
24. května 2011

Poděkování

Tímto bych rád poděkoval panu Jiřímu Konečnému, jednateli společnosti ELKO EP, s.r.o za poskytnutí potřebných prostředků pro řešení diplomové práce. Děkuji panu docentovi Martinu Drahanskému za jeho vedení a projevenou vstřícnost při zpracovávání této práce. Dále bych chtěl poděkovat Jakubovi Šálovi za cenné připomínky k mobilní aplikaci a zvláště pak také Ladislavu Némethovi za spolupráci a velkou ochotu při poskytování konzultací.
V neposlední řadě patří velké díky mé rodině za její obrovskou podporu a také všem mým přátelům.

Obsah

Kapitola 1

Úvod

V dnešní době se dostávají stále více do popředí většího zájmu pojmy jako inteligentní domy a inteligentní elektroinstalace. Tyto pojmy byly dříve označovány souhrnným názvem domácí automatizace, se kterým se lze často setkat i dnes. V podstatě můžeme říct, že se jedná o zjednodušené a modifikované systémy průmyslové automatizace, uzpůsobené potřebám nasazení v obytných prostorech. Tyto systémy nám umožňují snadnější ovládání běžných úkonů, se kterými se jako uživatelé bytu, domu, či jiného prostoru setkáváme každý den.

Tyto systémy často využívají centralizovaného ovládání prostřednictvím jedné či více řídicích jednotek, kterými může být jak embedded zařízení, osobní počítač, tak i kombinace obou možností.

V poslední době, kdy je kladen čím dál tím větší důraz na přenositelnost zařízení, se dostávají ke slovu také mobilní zařízení, mezi které patří např. chytré telefony či tablety. Tato zařízení umožňují uživateli velkou mobilitu, které je dosaženo možností připojení k síti pomocí více bezdrátových technologií. Proto roste význam jejich využití pro vzdálené ovládání systémů, mezi které samozřejmě patří i inteligentní domy.

Cílem této práce je v první řadě seznámit čtenáře s konkrétním řešením inteligentního domu. Dále také popsat a navrhnout vlastní řešení mobilního ovládání již existujícího systému spolu s možností vytvoření aplikace grafického návrháře usnadňujícího konfiguraci.

Kapitola 2

Inteligentní domy

Než přistoupíme k detailnímu popisu celé problematiky, seznámíme nejprve čtenáře této práce se základními pojmy, které budou v následujících kapitolách dále rozšiřovány, a proto je dobré zmínit jejich význam hned zpočátku.

Pod pojmem inteligentní dům si můžeme v podstatě představit komplexní systém pro měření řízení a regulaci. Systém se zpravidla skládá z více komponent propojených sběrnicí. Komponenty mohou být různých typů, od jednoduchých spínacích či vstupních jednotek, přes termostaty, stmívače, hladinové snímače apod., až po zabezpečovací, bezdrátové a multimediální systémy.

Pole záběru problematiky inteligentních domů je opravdu široké, což reflekuje také fakt, že na ČVUT vznikl v roce 2009 nový studijní obor Inteligentní budovy.

Existuje celá řada jak komerčních, tak i nekomerčních produktů. V poslední době stále narůstá množství firem, které poskytují vlastní řešení. Ve většině případů se jedná o modulární systémy, jejichž architektura si je navzájem velmi podobná a často se odlišuje pouze jinými marketingovými názvy.

V této kapitole se budeme nejprve detailněji věnovat komerčnímu řešení od společnosti ELKO EP, s.r.o. Provedeme také porovnání s nekomerčními alternativami a jejich zástupci vycházejícími z řad open source komunity. Tito zástupci se ve většině případů řadí do kategorie tzv. multimediálních center, která bývají podporována více platformami. Často však jsou vystavena nad jádrem operačního systému GNU/Linux.

Mezi nejznámější a nejdiskutovanější zástupce patří LinuxMCE (kapitola 2.4), jenž je od základu navržen tak, aby ho bylo možné propojit se systémem domácí automatizace. Dále pak XBMC (kapitola 2.5), který spolu s MythTV patří do kategorie multimediálních center.

S dalšími volně dostupnými alternativami se u nás můžeme setkat i v elektrotechnických časopisech jako jsou Praktická elektronika a Amatérské rádio, které na toto téma již publikovaly bezpočet článků a nadále je tato tematika v popředí jejich zájmu.

2.1 iNELS - Inteligentní elektroinstalace

Systém byl vyvinut společností ELKO EP, s.r.o. Jeho škála nasazení je opravdu široká, a to od bytů a rodinných domů, až po velké komplexní budovy.

Základním stavebním prvkem inteligentní elektroinstalace je centrální jednotka (dále jen CPU), ke které se pomocí dvoudrátové sběrnice *CIB* připojují snímače a aktory (dále jednotky). Sběrnice je sdílena všemi zařízeními, jež je možné kdykoliv připojit i odpojit, a

to bez potřeby vypnutí celého systému. Instalace je tedy velmi jednoduchá.

2.1.1 Centrální jednotka

Systém iNELS v současné době podporuje dva typy CPU. Prvním typem je vlastní jednotka CU2-01M, vyvinutá přímo společností ELKO EP, s.r.o. a druhým je jednotka TECOMAT FOXTROT od společnosti Teco, a.s. Obě CPU poskytují kompatibilitu s průmyslovým komunikačním síťovým protokolem EPSNET viz [11].

Jednotka CU2-01M

Tato CPU (viz obr. 2.1) zastává v prvé řadě funkci řídicího centra, rovněž je také prostředníkem mezi jednotkami připojenými na sběrnici a programovým prostředím, umožňující programování a ovládání s vyšší mírou abstrakce.

Komunikace mezi jednotkami a CPU zabezpečuje sběrnice *CIB (Common Installation Bus)*, která také slouží pro jejich napájení. S okolním světem CPU komunikuje prostřednictvím rozhraní Ethernet a UDP protokolu EPSNET, jenž zároveň slouží pro její ovládání a konfiguraci.

Konfigurace a programování se provádí v prostředí aplikace *INELS Designer and Manager* 2.2 (dále *IDM*), která je primárně určena pro operační systémy MS Windows, XP a vyšší. CPU také disponuje vestavěným webovým serverem, s jehož pomocí lze vzdáleně ovládat její uživatelské funkce.

Obrázek 2.1: Centrální jednotka CU2-01M.

Přehled vlastností [14]**:**

- K CU2-01M je možné přímo připojit až dvě sběrnice CIB, přičemž na každou sběrnici lze připojit maximálně 32 jednotek iNELS libovolného typu.
- Konektor RJ45 Ethernet portu se nachází na čelním panelu jednotky, rychlost přenosu je 10, nebo 100 Mbps.
- CU2-01M disponuje displejem, který zobrazuje stav jednotky a funkční tlačítko MODE; při stisknutí a přidržení řídicí jednotka zobrazí nastavení komunikace - IP adresa, maska, brána.
- CU2-01M lze vzdáleně konfigurovat a ovládat i přes internet (pokud je jednotka prostřednictvím sítě LAN na internet napojena).
- Při výpadku napájecího napětí jsou všechna data a čas zálohovány min. 72 hodin.

2.2 IDM - INELS Designer and Manager

Jedná se o proprietární softwarové řešení, které slouží pro kompletní správu systému iNELS, včetně CPU viz [14].

Samotný popis celého systému je v IDM reprezentován formou projektu, který obsahuje podklady pro konkrétní budovu (např. půdorysy podlaží). Uživatelé mohou tyto půdorysy dále dělit na místnosti, vkládat do nich jednoduché grafické komponenty a párovat je s fyzickými jednotkami systému iNELS. Seznam všech dostupných modulů je možné získat připojením CPU k počítači s nainstalovanou aplikací IDM.

Pro konfiguraci jsou podporovány tři úrovně přístupových práv [14]:

1. **administrátor**, umožňuje kompletní správu systému a projektu. Pro tuto úroveň není nutné připojení na CPU a lze pracovat off-line nad projektem na disku.

2. **konfigurátor**, umožňuje základní uživatelskou konfiguraci systému. Vyžaduje připojení na CPU.

3. **uživatel**, slouží jen pro základní zobrazení stavu systému a jeho ovládání. Vyžaduje připojení na CPU.

Jednotky mají v rámci celého systému unikátní fyzickou adresu uváděnou v hexadecimálním tvaru. Uživatel si sám může v IDM zvolit vlastní název dané jednotky a tím si ji jednoznačně a srozumitelně pojmenovat, např. `vypinac_horni_schodiste`.

Pokud uživatel při konfiguraci zvolí nesrozumitelné názvy, nevyhne se tak v budoucnu rekonfiguraci celého systému. Při rozsáhlém řešení se jedná o velmi náročnou akci, která bude mít za následek odstavení celého systému.

Rovněž je možné definovat také akce, které jsou vykonány buď na základě interakce uživatele s grafickými komponentami/jednotkami, či akce reagující na interní stavy systému získané prostřednictvím senzorů. Pro názornost můžeme uvést jednoduchý příklad, který zastává funkci termostatu. Při poklesu teploty v místnosti si přejeme aktivovat vytápění až do doby, než bude opět teplota dosahovat požadované úrovně.

2.2.1 Export projektu

Velmi důležitým rysem aplikace IDM je export uživatelem vytvořeného projektu. Jeho výstupem je běžný textový soubor s příponou *pub* (výpis 2.1). Soubor obsahuje seznam jednotek, včetně jejich parametrů, přičemž musí platit, že na každém řádku smí být uvedena pouze jedna jednotka viz [11].

Jednotkami rozumíme v tomto případě následující aktory a snímače:

- Vstupy a Výstupy
- Časový program
- Časová událost
- Čítače a Časovače
- Události systému určené pro vizualizaci

Každý řádek souboru musí odpovídat požadovanému formátu (tabulka 2.1) a předpokládá se, že jednotlivá pole jsou oddělena mezerou.

Pro potřeby této práce jsou důležitá zejména pole `inels_item` a `TYPE`. Detailní význam jednotlivých polí lze nalézt v [11]. Nutné je však podotknout, že obsah exportovaného souboru, zejména prvního pole `inels_item`, slouží k propojení CPU a tedy i všech jednotek

inels_item	REG	CF	ADDR	[.B]	TYPE	PUB_INOUT

Tabulka 2.1: Formát exportovaného souboru [11].

s okolním světem. Bez existence tohoto souboru a „znalostí“, jež obsahuje, by nebylo možné k jednotkám vůbec přistupovat.

Pole `TYPE` může nabýt jedné z hodnot `REAL`, `BOOL`, nebo `BYTE` podle toho, jakou hodnotu lze z jednotky číst, nebo do ní zapsat.

Tohoto faktu se využívá při napojení CPU na multimediální systém *iNELS Multimedia* (dále iMM) viz kapitola 2.3 a důležitý je také pro implementaci GUI návrháře (viz kapitola 4.2), který je součástí této práce.

Výpis 2.1: Ukázka exportovaného souboru

```
Lampa_ON R B 18129 .0 BOOL PUB_INOUT
Lampa_OFF R B 18129 .1 BOOL PUB_INOUT
Lampa_TRIG R B 18129 .2 BOOL PUB_INOUT
Lampa R B 18139 .0 BOOL PUB_INOUT
state_Svetlo_na_zdi_nalevo Y B 16 .1 BOOL PUB_OUT
Svetlo_na_zdi_nalevo_ON R B 18156 .0 BOOL PUB_INOUT
```

2.3 iMM - iNELS Multimedia

Jedná se o systém, který bývá provozován buď zcela samostatně, nebo jako další rozšíření systému iNELS. Uživateli tak přináší nejen podporu ovládání multimédií napříč několika místnostmi, ale díky možnosti propojení s CPU také přehledné ovládání a vizualizaci všech jednotek. iMM také podporuje rozmístění jednotlivých komponent do místností podobně, jako je tomu u jednotek. Pojem místnost zde nahrazuje zóna. Zóny slouží v podstatě pro jednoznačnou identifikaci iMM klientů a speciálních zařízení, určených pro streamování hudby. V místnosti můžeme mít několik zón, avšak nejčastěji je vztah zóny a místnosti *1:1*. Zóny byly do stávajícího systému přidány až zpětně a jejich vztah k místnostem nebyl explicitně definován. Pro uchování relace místností (zón) a jednotek slouží konfigurační soubor ve formátu XML s názvem `rooms.cfg` (výpis 2.2). Tento soubor má silnou vazbu na projekt exportovaný z aplikace *IDM* (kapitola 2.2.1). XML formát tohoto souboru obaluje obsah první pole `inels_item` výše zmíněného *pub* souboru a rozšiřuje ho o další sémantiku. Hodnoty pole `inels_item` a tím i konkrétní akce fyzicky existujících jednotek lze dále řadit do kategorií podle jejich účelu (světla, lampy, rolety, kamery atd.), a nakonec je přiřadit požadovaným místnostem.

Výpis 2.2: Ukázka konfiguračního souboru `rooms.cfg`

```
<?xml version="1.0" encoding="UTF-8"?>
<rooms>
  <room name="obyvak">
    <lamps>
      <item inels="da22_rs_stmivana_zasuvka_lampa">Lampa 1</item>
    </lamps>
    <lights>
      <item inels="sa02_rs_led_osvetleni">Stena</item>
    </lights>
    <shutters>
      <item
```

```
        up="evnt_wsb80_stena_zaluzie_nahoru"
        down="evnt_wsb80_stena_zaluzie_dolu">Žaluzie</item>
    </shutters>
    <cameras>
      <item loc="~/Cameras/parkoviste.cam">Parkoviště</item>
    </cameras>
  </room>
</rooms>
```

Během návrhu a implementace řešení byly zvoleny právě soubory *pub* a `rooms.cfg` jako prostředky poskytující pořebné informace jak pro mobilní aplikaci, tak i pro samotný GUI návrhář.

2.3.1 Architektura

Stavebním prvkem systému iMM je distribuovaně orientovaná síťová aplikace určená pro běh na operačním systém GNU/Linux. Teoreticky je možné aplikaci nasadit na platformy podporované tímto operačním systémem. V současné době se využívá hlavně osobních počítačů, jejich miniaturizovaných a designových modifikací, notebooků a dotykových panelů. Aplikaci je možné provozovat v rámci jednoho či několika osobních počítačů, jejich počet není teoreticky nijak omezen.

Systém iMM můžeme rozdělit na následující části:

- EPSNET server
- iMM Klient
- iMM Touch
- iMM Tablet

EPSNET server

Tato aplikace jako jediná umožňuje komunikaci s CPU prostřednictvím UDP protokolu EPSNET. Pokud uživatel vyžaduje propojení systému iMM a iNELS, musí být v každé konfiguraci tato aplikace přítomna. Provozovat ji můžeme jak na odděleném počítači, který často slouží také jako úložiště dat, tak v rámci samotného klienta.

iMM Klient

Klientem v tomto případě rozumíme osobní počítač, ke kterému je připojen širokoúhlý monitor či televize. Ovládání celého systému probíhá prostřednictvím grafické aplikace, kterou uživatel kontroluje zejména gyroskopickou myší, případně i klávesnicí. Klávesnice bývá využívána pouze v případě, že si prohlížíme internet a potřebujeme tedy cestu pro zadávání jednotilvých znaků. iMM klient umožňuje přístup k následujícím akcím [11]:

- **prohlížení fotografií** – uživatel si může prohlížet jednotlivé fotografie uložené v centrální databázi či si spustit jejich prezentaci.
- **přehrávání hudby** – uživateli je pro každou zónu poskytnuta možnost ovládat přehrávání hudby z centrální databáze.
- **přehrávání videa** – stejné jako předchozí bod s tím rozdílem, že jsou přehrávány videozáznamy.
- **sledování televizního vysílání** – uživatel si může ze seznamu zvolit požadovanou stanici a místnost, kde ji chce sledovat. Tato akce předpokládá přítomnost zařízení pro streaming televizního vysílání v systému.

- **prohlížení internetu** – uživateli se otevře okno webového prohlížeče
- **osvětlení** – uživatel si může volit mezi několika půdorysy místností a ovládat tak příslušné jednotky svázané s osvětlovací technikou.
- **kamery** – přenos obrazu a ovládání přítomných IP kamer.

2.3.2 iMM Touch

iMM Touch je jen jiný název pro touchpanel, jenž je speciálním typem iMM klienta. Lze ho vestavět, např. do zdi na několika místech v obytném prostoru. K iMM se připojuje ethernetovým kabelem. Grafické rozhraní je navrženo s ohledem na zobrazení na touchpanelu a dokáže se přizpůsobit velikosti displeje. Provázání se systémem iNELS a ostatními iMM klienty je umožněno správným nastavením konfiguračních souborů. Je to také jediná možnost jak na touchpanelu zobrazit odpovídající grafické komponenty reprezentující jednotky. Stejně jako iMM klient podporuje iMM Touch rozdělení jednotek do zón.

Ovládání CPU je možné pouze skrze EPSNET server a není tedy podporována přímá komunikace.

2.3.3 iMM Tablet

Jedná se o mobilní řešení panelu iMM Touch, které je předmětem této práce a jeho návrh je detailněji popsán v následující kapitole.

2.3.4 Implementační detaily

Řešení iMM je implementováno v dynamickém objektově orientovaném interpretovaném jazyce Python. Společně se zvoleným GUI frameworkem wxPython, který je zapouzdřením multiplatformí knihovny wxWidgets implementované v jazyce C++, umožňuje rapidní vývoj této aplikace.

Komunikace

Komunikace v iMM je založena na distribuované objektově orientované technologii *PYRO* neboli *PYthon Remote Objects*. Jak již její název napovídá, jedná se o technologii implementovanou v jazyce Python, což plně vyhovuje požadavkům systému. *PYRO* je open source řešení založené na technologii RPC (Remote procedure call), která je rozšířena o vzdálené volání objektů jazyka Python. Projekt *PYRO* lze přirovnat k technologii Java Remote Method Invocation (RMI), kterou je také do značné míry inspirován.

Pro potřeby realizace iMM Tabletu se ukázalo, že stávající provedení není vhodné pro komunikaci s mobilními zařízeními. Důležitým faktem je, že implementace jazyka Python pro ně není často dostupná, a pokud ano, tak není možné její nativní napojení na GUI.

Bylo tedy nutné navrhnout řešení (viz kapitola 4.1.1), které lze jednoduše začlenit do stávajícího systému, a odstínit tak komunikaci od silné vazby na jazyk Python.

2.4 LinuxMCE

Mezi známé zástupce volně dostupných řešení patří projekt LinuxMCE, který se řadí mezi komplexní open source řešení multimediálního centra v kombinaci s domácí automatizací.

Jedná se o distribuovanou síťovou aplikaci typu klient-server, založenou na linuxové distribuci Kubuntu. Aplikace je tvořena stovkami skriptů a nástrojů převzatých z distribuce Pluto Linux. Jádro aplikace je implementováno v jazyce C/C++. Pro komunikaci se všemi komponentami systému slouží tzv. *DCERouter*, který umožňuje výměnu zpráv prostřednictvím socketů [17].

LinuxMCE poskytuje tyto základní vlastnosti [17]:

- Správa a přehrávání médií, včentě záznamu televizního vysílání ze set-top boxů či jiných zařízení podporujících streaming videosignálu.
- Ovládání systému domácí automatizace.
- Podpora VoIP.
- Bezpečnostní systém.

2.4.1 Architektura

Jádro

Za jádro je považován počítač, který zastává funkci serveru pro ostatní komponenty systému. Naslouchá na síti, a pokud nalezne dostupná síťová zařízení (např. IP telefony a kamery), které podporuje, provede jejich konfiguraci a zpřístupní je uživatelům systému [17].

Speciálním klientům (multimediální stanice) poskytuje obraz operačního systému, který mohou spustit přímo ze sítě [17].

Multimediální stanice

Dedikovaný počítač, který je schopný přijímat multimediální data přímo z hlavního serveru [17]. Díky možnosti zavedení operačního systému přímo ze sítě bývá často nazýván jako tzv. *thin-client*. GUI aplikace pro multimediální stanici rozhodně nepatří mezi intuitivní a uživatelsky přívětivé.

Podobně jako u iMM jsou podporovány zóny. Každá místnost může tedy obsahovat jednu či více těchto stanic.

Pozorovatel

Pozorovatel je zařízení, které slouží pro vzdálené ovládání celého systému. Do této kategorie spadají mobilní zařízení (Symbian, Microsoft Mobile), PDA, bezdrátové tablety, notebooky a další viz [17]. Pro tato zařízení musí být dostupný webový prohlížeč a tedy i možnost připojení k síti LAN [17].

Pozorovatel se připojí k hlavnímu serveru, který mu poskytne webové rozhraní pro ovládání celého systému.

Domácí automatizace

LinuxMCE může také řídit funkce domácí automatizace. Systém podporuje napojení na následující komerční řešení [16]:

- Z-Wave
- EIB/KNX
- X10
- Insteon
- 1-Wire
- PLCBUS
- EnOcean

Bezpečnostní systém

Komplexní řešení pro zabezpečení celého objektu. Připojením IP kamer a pohybových čidel PIR (Passive Infrared Sensor) lze monitorovat daný prostor a vyvolat příslušné akce při bezpečnostním konfliktu [17]. Systém umožňuje definovat rozvrhy a různé scénáře, např. při opuštění domu jsou zamknuty dveře a aktivován bezpečnostní systém v předem definovaných místnostech.

2.5 XBMC

XBMC (z původním názvu „XBox Media Center“) je volně dostupné softwarové řešení multimediálního centra s celoobrazovkovým GUI. Centrum lze provozovat na platformách Linux (OpenGL), Mac OS X (OpenGL), Apple TV, Windows XP/Vista/7 (OpenGL) a Xbox. Implementačním jazykem je C/C++.

XBMC je určeno pro přehrávání videa, hudby a prohlížení obrázků, které umožňuje načítat jak z pevných či optických disků, tak i ze síťových zdrojů. Díky tomu je možné streamovat multimédia z lokální sítě nebo přímo z internetu (Samba, FTP, HTTP a další).

Pro přehrávání a zobrazování je podporována většina majoritních formátů. Kompletní seznam viz [24]. Aplikaci lze dále snadno rozšiřovat pomocí pluginů/skriptů napsaných v jazyce Python a měnit vzhled pomocí uživatelsky definovaných skinů.

XBMC zastává podobnou funkci výše uvedeného řešení iMM s tím rozdílem, že není v základní verzi umožněno napojení na inteligentní elektroinstalaci. To lze však do aplikace dodatečně implementovat díky skriptům, které přidávají další funkčnost (přehrávání internetového rádia, streaming televizního vysílání Dreambox apod.). Co se však týče propracovanosti GUI, podpory pro multimediální formáty a rozšiřitelnosti, je tato aplikace daleko robustnější.

Kapitola 3

Analýza požadavků a dostupných prostředků

V této kapitole jsou diskutována možná řešení a prostředky vedoucí ke splnění požadavků vycházejících z předběžné analýzy.

Požadavky lze rozdělit v zásadě do dvou kategorií. Prvním požadavkem bylo vytvoření mobilního řešení iMM Touch nesoucí nové označení iMM Tablet, které spadá do kategorie doplňkových dotykových panelů pro iMM. Nejprve se tedy budeme v této kapitole věnovat popisu mobilní platformy Android, která byla zvolena jako primární pro implementaci aplikace iMM Tablet, a to hlavně díky své otevřenosti a dostupnosti vývojového kitu SDK.

V budoucnu se předpokládá rozšíření aplikace iMM Tablet i na další platformy jako je např. operační systém iOS od společnosti Apple Inc.

Druhým požadavkem bylo vytvoření desktopové aplikace primárně určené pro operační systém Microsoft Windows XP a novější, která by sloužila jako grafický nástroj pro konfiguraci iMM Tabletu a v budoucnu by bylo umožněno ji rozšířit i na další platformy. Dále budeme v této kapitole rozebírat možné přístupy k implementaci desktopové aplikace, udržitelnost jejího vývoje a v neposlední řadě také její škálovatelnost.

3.1 Android

Android je softwarová platforma založená na modifikovaném jádře operačního systému Linux. Primárně je určena pro mobilní zařízení jako jsou chytré mobilní telefony (smartphones), PDA, GPS navigace a další.

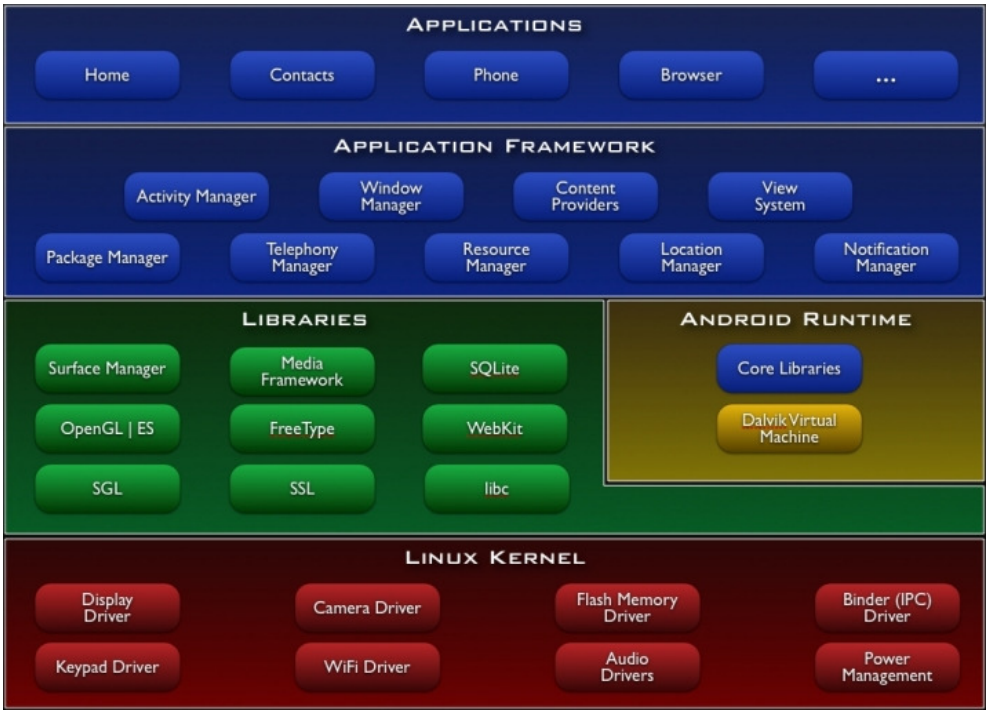
Původně byla tato platforma vyvinuta společností Android Inc., kterou v roce 2005 koupila společnost Google a následně ji i se zdrojovými kódy předala konsorciu Open Handset Aliance, jejímž je také členem. Členy konsorcia jsou i některé další známé společnosti. Mezi významné zástupce patří Intel, Samsung, HTC, Nvidia a T-Mobile. V roce 2008 iniciovala společnost Google vydání všech zdrojových kódů pod licencí Apache, která je umožňuje začlenit jednak do proprietárních, tak i open source aplikací.

Formou, jakou je tato platforma vyvíjena, lze předpokládat její velkou budoucnost, a proto byla zvolena pro potřeby této práce. Již dnes ji lze zařadit jako velkého konkurenta řešení od společnosti Apple Inc., která si dlouhou dobu udržovala prvenství na poli chytrých mobilních zařízení.

Architektura

Architektura platformy Android je rozdělena do více vrstev, přičemž vyšší vrstva využívá vždy služeb nižší vrstvy.

Pro znázornění architektury slouží nejlépe následující diagram (obr. 3.1).



Obrázek 3.1: Architektura platformy Android [2].

Vrstva aplikací – v hierarchii stojí tato vrstva na nejvyšší úrovni. Je tvořena sadou hlavních aplikací. Zahrnujíce aplikaci pro rozesílání SMS zpráv, adresář kontaktů, e-mailový klient, prohlížeč internetu a mapových podkladů a další.

Všechny aplikace této vrstvy jsou implementovány v programovacím jazyce Java [2].

Aplikační rámec – v hierarchii jako druhý poskytuje otevřenou platformu pro vývoj. Vývojářům umožňuje přístup ke stejnému aplikačnímu rozhraní, které využívají hlavní aplikace. Přidává tak podporu pro opětovné využití jednotlivých komponent.

Vrstva knihoven – tato vrstva je spolu s běhovým prostředím nejblíže samotnému jádru operačního systému. Její součástí je sada knihoven napsaná v jazyce C/C++, která je zpřístupněna vývojářům prostřednictvím aplikačního rámce.

Běhové prostředí – platforma obsahuje také sadu knihoven, které implementují většinu funkcionality knihovny programovacího jazyka Java.

Pro běh samotných aplikací slouží *Dalvik Virtual Machine* (dále jen *Dalvik VM*). Každá aplikace je spuštěna ve svém vlastním procesu spolu s instancí *VM*. Na rozdíl od standardizovaných běhových prostředí jazyka Java (JVM), která jsou založena na zásobníkových automatech, využívá *Dalvik VM* architektury založené na registrech.

Jádro – celá platforma Android je vystavěna na jádře operačního systému Linux verze 2.6.

Android SDK

Vývojový kit je na internetu volně dostupný pro platformy Microsoft Windows, Mac OS X (architektura Intel) a Linux (i386). Oficiláním podporovaným prostředím pro vývoj je Eclipse IDE. Konkrétně je doporučováno od verze 3.5 používat sestavení „Eclipse Classic“. Vývoj aplikací je umožněn díky pluginu ADT (Android Development Tools), který je taktéž volně dostupný.

GUI pro aplikace lze vytvářet pomocí specifických XML souborů, nebo programově. Vývojové prostředí integruje grafický návrhář pro první zmíněnou možnost. Vývojáři je tak poskytnut prostředek pro rychlý návrh aplikace. V současné době ovšem nejsou pokryty všechny možnosti, kterých lze docílit samotnou modifikací zmíněných XML souborů.

Oproti vývojovému prostředí a SDK, které poskytuje společnost Apple Inc. pro své mobilní zařízení, má otevřenost vývojové platformy Android velkou řadu výhod. Díky otevřenosti lze očekávat stále větší počet přispěvatelů jak do samotné platformy, tak i při tvorbě aplikací.

Struktura aplikací

Aplikace pro platformu Android jsou vyvíjeny v programovacím jazyce Java. Nástroje Android SDK se starají o kompilaci zdrojových kódů a vytvoření příslušného balíčku určeného pro distribuci. Balíček aplikace má příponu `.apk` a lze ho snadno instalovat na jakékoliv mobilní zařízení s operačním systémem Android.

Po instalaci existuje aplikace ve svém vlastním prostoru nazývaném *security sandbox* [3]:

- Operační systém Android je víceuživatelským linuxovým systémem, každá jeho aplikace je zastoupena vlastním uživatelem.
- Systém přiřadí aplikaci unikátní identifikační uživatelské číslo, které je využíváno pouze systémem, samotná aplikace o jeho existenci neví, a následně nastaví práva všem souborům v aplikaci tak, aby byl přístup umožněn pouze uživateli s tímto identifikačním číslem.
- Každý proces obsahuje vlastní instanci *VM*, a proto je vykonávání kódu jednotlivých aplikací navzájem izolováno.

Tento přístup je nazýván *principem minimálních privilegií*, kde každá aplikace může přistupovat jen k prostředkům, které nezbytně potřebuje pro svoji funkci. Díky tomu nabízí platforma Android bezpečné prostředí pro běh aplikací a prostředek k omezení neoprávněného přístupu do částí systému, ke kterým nemají aplikace přidělena příslušná práva [3].

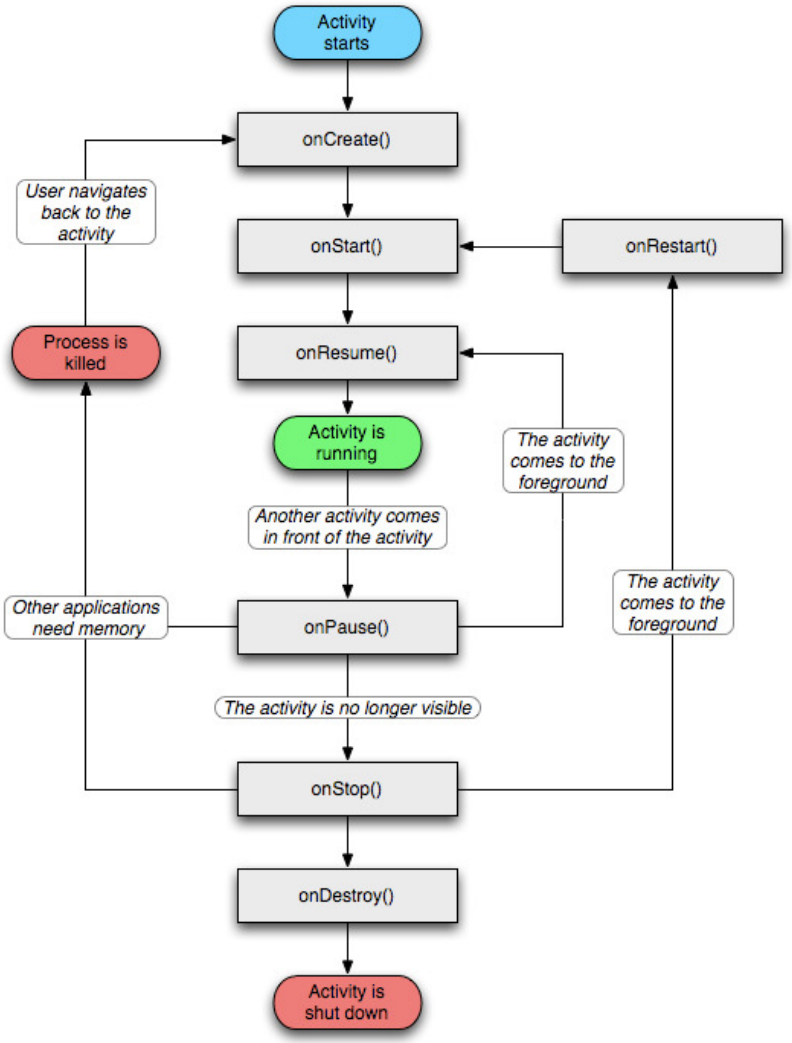
Platforma Android poskytuje pro vývoj aplikací čtyři základní stavební bloky — Activity, Services, Content providers a Broadcast receivers.

Activity

Activity jsou součástí prezentační vrstvy aplikace. Jedná se o základní a nejčastěji používanou komponentu. Každá tato komponenta rozšiřující třídu `Activity` reprezentuje jednu obrazovku. Activity jsou tvořeny tzv. *View*, které zobrazují informace v rámci GUI a reagují na příslušné uživatelské akce [10].

Aplikace typicky obsahuje sadu několika komponent Activity[1]. Pokud to aplikace dovoluje, je možné aktivity spouštět i z jiných aplikací.

Pro vytvoření vlastní aktivity musí být nejdříve vytvořena třída dědící od třídy `Activity` nebo její podtřídy. Následuje implementace tzv. *callback* metod, které jsou vyvolány na základě přechodů mezi stavy životního cyklu aktivit (viz obr. 3.2), v němž jsou jednotlivé aktivity vytvořeny, pozastaveny, obnoveny, či zrušeny [1].



Obrázek 3.2: Životní cyklus aktivity [2].

Důležitá je implementace, zejména metod `onCreate()` a `onPause()`. Metodu `onCreate()` systém volá při vytváření aktivity, je to tedy místo, kde bychom měli provést potřebnou inicializaci komponent, včetně definice rozvržení uživatelského rozhraní aktivity zavoláním metody `setContentView()` [1].

Metoda `onPause()` je vyvolána těsně před tím, než opustí aktivitu. Na tomto místě se typicky provádí uložení změn stavu aktivity, aby tyto změny mohly být použity při jejím opětovném spuštění.

[1] V následujícím textu bude pro pojem activity užíváno českého synonyma aktivita.

Další metody je možné využít v případech, kdy nám z nějakého důvodu nepostačují základní metody `onCreate()` a `onPause()`.

Services

Service neboli služba, se využívá k vykonávání časově náročných operací na pozadí tak, aby neblokovaly uživatele při další interakci s uživatelským rozhraním [3].

Content providers

Content provider je nástrojem pro správu a sdílení dat mezi aplikacemi. Skrze content provider mohou aplikace tato společná data číst, případně je i modifikovat, pokud je to povoleno. Jako úložiště může být zvolen lokální systém souborů, SQLite databáze a další viz [3].

Broadcast receivers

Broadcast receiver slouží pro vyvolání příslušné akce, která je iniciována systémovou událostí, např. příchozí volání či zpráva SMS.

3.2 Prostředky pro tvorbu GUI aplikace

Jak již bylo dříve zmíněno, aplikace GUI návrháře má být určena primárně pro provoz na platformách Microsoft Windows. Zároveň je kladen požadavek na možnost budoucího rozšíření na ostatní platformy, což musí být zohledněno jednak při analýze aplikace, tak i při volbě programovacího jazyka, případně i aplikačního rámce.

Pro implementaci aplikce se nabízí několik přístupů. Některé z nich budou rozebrány v následujícím textu.

3.2.1 Výstavba aplikace „na zelené louce“

Vždy je k dispozici možnost vytvořit aplikaci úplně od začátku a přenechat tak vývojáři či vývojářům zcela volné ruce. Nevýhodou této volby je doba vývoje, která je nepoměrně vyšší než při implementaci využívající dílčích existujících řešení nebo aplikačních rámců.

Pokud nejsou respektovány uznávané zásady tvorby aplikace, bývá častým problémem udržitelnost při jejím vývoji a rozšiřování. Mezi tyto zásady určitě patří využití návrhových vzorů, které dodává implementaci na přehlednosti a výrazně zjednodušuje následnou refaktorizaci zdrojových kódů.

Aplikační rámce nebo platformy, které je sdružují, jsou při implementaci rozsáhlejších aplikací víceméně samozřejmostí. Avšak i u nich je nutné tyto zásady respektovat.

3.2.2 Aplikační rámce

Aplikační rámec neboli framework slouží ke zkvalitnění procesu návrhu a vývoje aplikace. Často poskytuje implementaci funkcionality, která je běžně při tvorbě aplikace vyžadována. Programátorovi tak umožňuje se soustředit na samotné zadání a neřešit dílčí problémy, mezi které patří např. lokalizace a internacionalizace. Díky abstrakci je také umožněno rozšiřovat či přetěžovat stávající implementaci tak, aby bylo dosaženo požadovaných výsledků.

Aplikační rámce jsou zvláštním případem softwarových knihoven, které díky abstrakci umožňují opětovné užití kódu, jenž se ukrývá za přesně definovaným programovým aplikačním rozhraním (API) [20]. Skrývání implementace za rozhraní patří mezi zásady objektově orientovaného programování, ve kterém se dává často přednost kompozici před dědičností.

Některé rámce poskytují rovněž nástroje pro generování kódu a jsou navrženy tak, aby vývojář následoval běžně uznávané postupy při implementaci, které v sobě skýtají návrhové vzory. Příkladem může být framework pro tvorbu webových aplikací *Ruby on Rails*, který využívá návrhového vzoru MVC a poskytuje tzv. *scaffolding*, který slouží jako generátor kostry aplikace nad zvoleným datovým modelem. Dále pak například odlehčený kontejner pro vývoj J2EE aplikací Spring Framework, jenž je postaven na návrhových vzorech *Inversion of Controll* a *Dependency Injection*.

Kvalita výsledné aplikace záleží mimo jiné i na kvalitě zvoleného aplikačního rámce.

V současné době se do popředí zájmu dostává termín *Rich Client Platform* neboli RCP. Jedná se o speciální případ aplikačního rámce, kerý můžeme v pomyslné hierarchii softwarové knihovny a aplikačního rámce zařadit na nejvyšší úroveň díky tomu, že pro vývoj nabízí celou existující platformu.

3.2.3 Rich Client Platform

Z hlediska architektury typu klient-server se pojem „rich client“ užívá pro označení klientů, u kterých probíhá zpracování dat převážně na klientské straně. Klientské aplikace poskytují také GUI a často jsou rozšiřitelné prostřednictvím zásuvných modulů [5].

Nejdůležitějším aspektem *Rich Client Platform* (dále RCP) je její architektura. Aplikace založené na této platformě jsou tvořené moduly umožňujícími izolovat logicky koherentní části aplikace. Moduly jsou popsány deklarativně a automaticky načítány platformou. Ve výsledku tedy není potřebná explicitní vazba mezi zdrojovým kódem a aplikací. Díky tomuto přístupu je umožněno ustavit relativně volné vazby (tzv. *loose coupling*) mezi moduly a tím umožnit jednoduchou výměnu základních částí, které moduly poskytují [5].

Platforma RCP je ve většině případů složena z následujících komponent [19]:

- Jádro (mikrojádro) a manažer životního cyklu platformy.
- Standardizovaný aplikační rámec.
- Knihovna grafických uživatelských prvků.
- Knihovny zapouzdřující přístup k systému souborů.
- Workbench, který sdružuje pohledy, editory, perspektivy a pomocníky (wizards) do logických celků.
- Data bindig, umožňující provázání několika vlastností a vzájemné reflektování změn jejich stavů.
- Manažer aktualizací.

Na rozdíl od přístupu výstavby aplikace „na zelené louce“, mohou vývojáři těžit z osvědčených a předem otestovaných vlastností aplikačního rámce, který platforma poskytuje. Přístup na bázi RCP umožňuje rapidní vývoj aplikace, na druhou stranu však vývojář musí být před implementací seznámen s danou platformou. Ta bývá často velmi rozsáhlá a komplikovaná. Kvůli tomuto faktu je pro začínajícího vývojáře nutný čas, který by mohl být

věnován v přístupu „na zelené louce“ samotné implementaci, strávit studiem platformy. Avšak z hlediska budoucího vývoje aplikace by měla být tato časová investice výhodnou.

Mezi významné open source zástupce RCP patří zejména platformy, nad kterými jsou implementována dvě konkurenčí vývojová prostředí Eclipse a NetBeans.

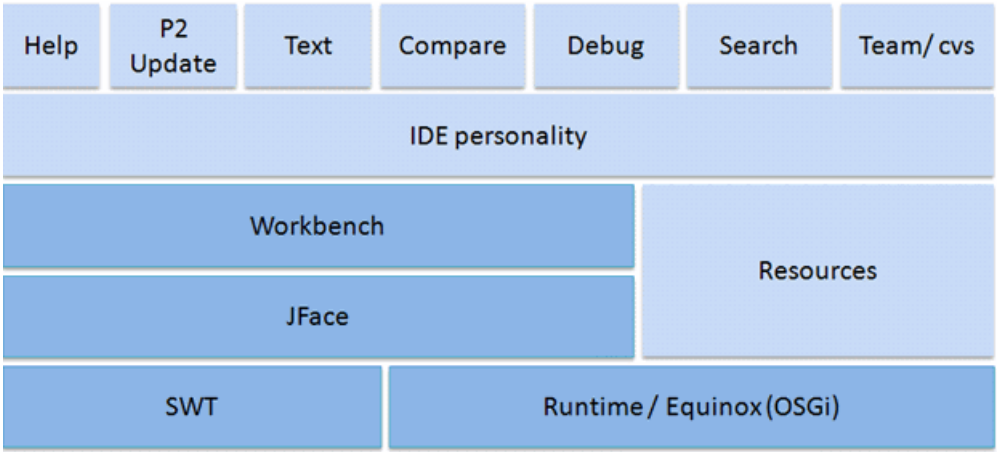
3.3 Eclipse

Eclipse je open source platforma, primárně využívaná jako vývojové prostředí (dále IDE) pro velkou škálu programovacích jazyků. Jejich podpora je do prostředí přidávána prostřednictvím pluginů.

Platforma je implementována v programovacím jazyce Java a v současné době patří mezi nejvyužívanější IDE pro tento jazyk. Prostředí Eclipse je dostupné pro platformy Linux, Windows a Mac OS X jak v 32bitové, tak 64bitové verzi.

Kompletní diagram architektury Eclipse IDE je na obr. 3.3.

Vzhledem k otevřenému zdrojovému kódu platformy je možné vytvořit vlastní derivát tohoto prostředí.



Obrázek 3.3: Architektura platformy Eclipse [18].

Platforma Eclipse je postavena nad rámcem OSGI (Equinox). Pluginy (v OSGI nazývané bundle) mohou být nahrávány dynamicky. Díky těmto pluginům lze rozšířit Eclipse IDE nebo aplikaci postavenou na Eclipse RCP [21]. GUI je implementováno prostřednictvím rámce *SWT (Standard Widget Toolkit)*, který využívá k vykreslování grafických prvků nativní knihovny operačních systémů prostřednictvím *JNI (Java Native Interface)* obdobným způsobem jako programy psané s použitím nativního API operačního systému. To je velký rozdíl oproti konkurenčnímu prostředí NetBeans IDE, používající rámec *Swing*, implemetnovaný čistě v jazyce Java.

JFace je abstraktní vrstvou nad rámcem *SWT*. Umožňuje jednoduše přistupovat k prostředkům tohoto rámce s ohledem na MVC (Model View Controller) architekturu, kterou *JFace* implementuje viz [13].

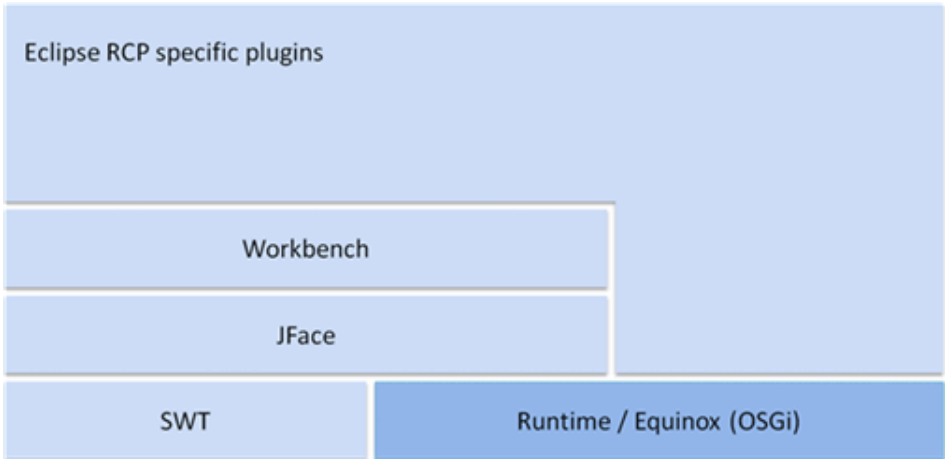
Workbench poskytuje aplikační rámec, ve kterém jsou všechny grafické komponenty zobrazeny.

Výše popsané komponenty jsou základem platformy Eclipse. Další komponenty z diagramu architektury jsou specifické pro vývojová prostředí.

3.3.1 Eclipse RCP

Eclipse RCP je platforma pro vývoj grafických aplikací založená na architektuře Eclipse. Vývojáři poskytuje kvalitní a stabilní platformu pro vývoj aplikace, přičemž základní stavební prvky jsou již součástí této platformy, a může se tedy věnovat hlavně implementaci požadované funkcionality.

Od vývojového prostředí Eclipse IDE se odlišuje pouze sadou základních komponent (viz obr. 3.4).



Obrázek 3.4: Architektura platformy Eclipse RCP [18].

Vývojáři jsou poskytnuty všechny základní prostředky pro vývoj GUI aplikace. Hlavní komponenty Eclipse RCP aplikace:

- **Hlavní program** - hlavní aplikační třída implementující rozhraní `IApplication`.
- **Perspektiva** - kontejner, který v sobě seskupuje view a editory. Mezi perspektivami lze přepínat a komponenty se mohou v rámci perspektivy uspořádat podle potřeb. Eclipse si toto uspořádání pamatuje. Perspektiva by měla sdružovat komponenty vztahující se k jedné problematice [21].
- **Workbench Advisor** - skrytá komponenta, která kontroluje vzhled aplikace (komponenty, menu, perspektivy apod.).

Zpočátku byla pro požadovanou implementaci návrháře zvažována pro svoji robustní architekturu právě tato platforma. Přestože získané zkušenosti s vývojem aplikací v Eclipse IDE byly převážně pozitivní, nakonec bylo od této volby upuštěno, a to hlavně pro nedostupnost potřebné literatury a návodů umožňujících snadnou orientaci v platformě bez její předchozí znalosti.

V současné době je na tom konkurenční řešení NetBeans Platform, co se týče dokumentace a návodů pro začínající vývojáře, daleko lépe. Samotné stránky tohoto projektu poskytují velmi užitečné a přehledně organizované informace.

3.4 NetBeans IDE a NetBeans Platform

Pro implementaci návrháře, o kterém pojednává tato práce, byla nakonec zvolena právě platforma NetBeans. V následujících několika odstavcích bude tedy čtenář postupně seznámen s jejími vlastnostmi.

Vznik NetBeans IDE se datuje do roku 1996 a jeho původní název byl Xelfi. Jednalo se o projekt studentů Matematicko-fyzikální fakulty Univerzity Karlovy v Praze. Původním záměrem bylo vytvoření IDE pro jazyk Java, které by bylo obdobou prostředí Delphi pro jazyk Object Pascal [12].

Postupem času byla zakladateli vytvořena společnost, která si kladla za cíl nabízet projekt Xelfi jako komerční produkt. Prostředí bylo přejmenováno na NetBeans, které následně odkoupila společnost Sun Microsystems, která uvolnila jeho zdrojové kódy pod open source licencí viz [12].

Platforma NetBeans jako taková zpočátku neexistovala. Její vznik byl reflektován programátory, kteří začali vytvářet vlastní aplikace postavené nad běhovým jádrem NetBeans doplněné vlastními moduly a nejednalo se o nástroje pro vývoj.

Během let 2000 a 2001 byla platforma postupně izolována od samotného IDE [12].

Na rozdíl od Eclipse IDE, obsahuje základní verze kvalitní nástroj RAD (Rapid Application Development) pro tvorbu grafického rozhraní. Díky tomu je vývoj desktopových aplikací daleko rychlejší a pohodlnější.

3.4.1 Přehled základních charakteristik

Platforma nabízí výhody, jež poskytuje koncept RCP, četné množství aplikačních rámců a specifikací usnadňujících návrh a vývoj aplikací [5]. Koncept platforem Eclipse a NetBeans je velmi podobný. Často se liší jen v pojmenování termínů. V Eclipse jsou stavebními prvky pluginy a v NetBeans moduly, avšak v podstatě se jedná o označení téhož.

Mezi hlavní charakteristiky můžeme začlenit [5]:

- Aplikační rámec GUI
- Editor
- Aplikační rámec pro tvorbu a přizpůsobení zobrazení komponent
- Aplikační rámec pro tvoru pomocníků (wizards)
- Systém pro práci s daty
- Internacionalizace
- Systém pro práci s nápovědou

Aplikační rámec GUI

V rámci platformy jsou k dispozici okna, nabídky, nástrojové lišty a další komponenty, jež jsou založeny na technologii *AWT/Swing* [5]. Mohou tak být jednoduše implementovány vlastní komponenty, jež rozšiřují vlastnosti a akce těch stávajících.

Editor

Výkonný editor, který je součástí IDE, je možné rychle a jednoduše začlenit do vlastní aplikace a přizpůsobit ho jejím potřebám [5].

Aplikační rámec pro tvorbu a přizpůsobení zobrazení komponent

Aplikace a uživatelé často požadují zobrazení specifických komponent přizpůsobených jejich potřebám [5]. Jedná se například o různé dialogy, které jsou zpřístupněny přes aplikační rámec *Dialogs API*. Jsou navrženy tak, aby splňovaly požadavek na integraci do celé platformy a nabízely základní znovu využitelnou funkcionalitu, kterou lze snadno upravit dle potřeb aplikace [5].

Aplikační rámec pro tvoru pomocníků

Platforma NetBeans poskytuje jednoduchý nástroj pro tvorbu uživatelsky přívětivých pomocníků, kteří postupně provádějí uživatele aplikace možnými volbami [5].

Systém pro práci s daty

Z hlediska platformy mohou být data přítomna lokálně, vzdáleně skrze databáze, FTP a CVS, nebo ve formě XML souboru. Abstrakce umožňuje jednotlivým modulům přistupovat k datům stejným způsobem [5].

Internacionalizace

Platforma také obsahuje třídy a metody zpřístupňující internacionalizaci JavaHelp a dalších zdrojů. Textové konstanty jsou ukládány do tzv. *properties* souborů, které jsou v případě potřeby dynamicky načítány s ohledem na zvolené nastavení země a jejího jazyka [5].

Systém pro práci s nápovědou

Prostřednictvím standardního systému JavaHelp nabízí platforma uživateli systém pro centrální správu, integraci a zobrazení témat nápovědy. Každý modul může obsahovat nápovědu, která je pak zpřístupněna celé platformě [5].

3.4.2 Architektura

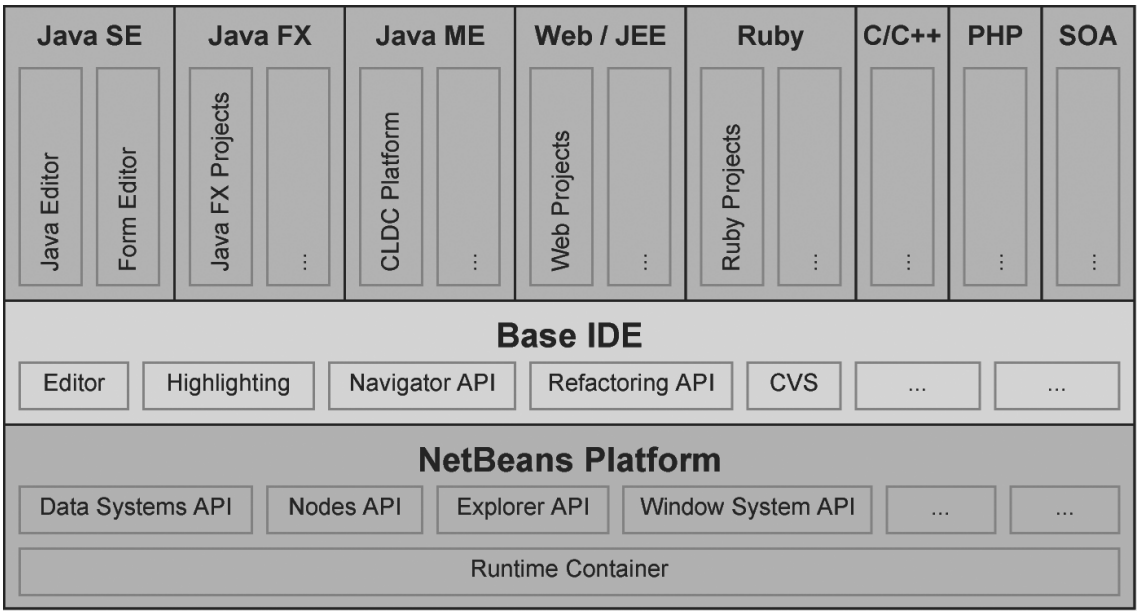
Rozsah a komplexnost moderních aplikací se neustále zvětšuje. Současně je požadováno, aby byly profesionální aplikace co nejvíce flexibilní a mohly být tak rychle a co nejjednodušeji rozšířeny o další funkčnost. Tento požadavek často vede k rozdělení aplikace do několika samostatných částí, tedy k vystavění modulární architektury. Moduly musí být navzájem nezávislé a díky definici přesně stanovených rozhraní zpřístupňovat ostatním částem aplikace konkrétní implemetaci [5].

Na rozdíl od monolitické architektury je modulární přístup flexibilnější a význačně zjednodušuje údržbu aplikace [5].

Jak již bylo v textu zmíněno, je základním stavebním prvkem platformy NetBeans modul [5]. Modul obsahuje navzájem související třídy, jejichž funkcionalita ja zpřístupněna prostřednictím veřejných metod deklarovaných v jednotlivých rozhraních či abstraktních třídách.

Platforma NetBeans, stejně jako aplikace vystavěná na jejím základě, je členěna do modulů, které umožňuje načíst tzv. *NetBeans Runtime Container*, jenž je součástí jádra platformy. Moduly jsou načítány běhovým kontejnerem plně dynamicky. Tento kontejner současně zajišťuje chod celé aplikace [5].

Příkladem *rich client* aplikace je samotné NetBeans IDE, jehož struktura je na obr. 3.5.

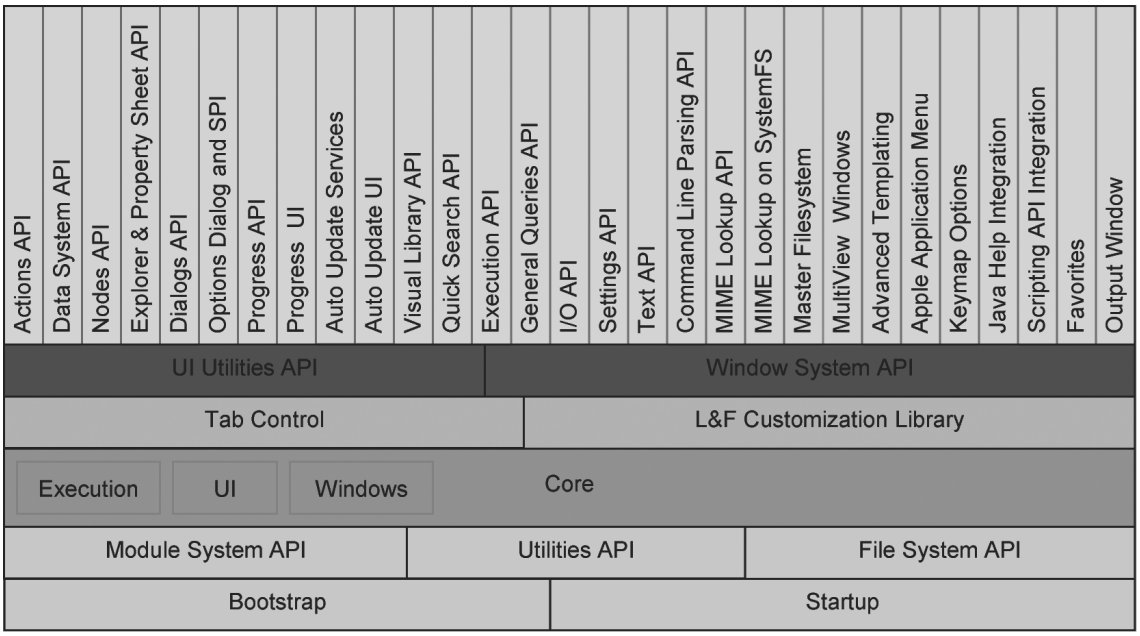


Obrázek 3.5: Konceptuální struktura NetBeans IDE [5].

Platforma NetBeans nabízí nejen mechanismus rozšiřitelnosti funkcionality určitého modulu jiným modulem, ale také prostředek pro komunikaci mezi moduly bez toho, aniž by na sobě byly závislé. V rámci celé aplikace jsou tak mezi moduly podporovány volné vazby — *loose coupling*. Pro optimalizaci zapouzdření kódu uvnitř modulu vyžadovaného modulární architekturou poskytuje platforma svůj vlastní prostředek pro správu a dynamické načítání tříd, tzv. *class loader* neboli zkráceně zavaděč. Platforma obsahuje tři typy zavaděčů. Každý modul obsahuje svůj vlastní, který se stará o načítání zdrojů z JAR archivu, do kterého je modul zabalen. Dále je k dispozici zavaděč aplikace, *application class loader*, který je implicitním předkem každého zavaděče modulu, a nakonec systémový zavaděč, *system class loader* [5].

Modul může samozřejmě využívat funkcionalitu implementovanou v jiných modulech. To umožňuje deklarovat vzájemné závislosti mezi moduly. Deklarace se provádí v rámci specifického souboru nazývaném *manifest*, který musí být obsažen v každém modulu. S existencí tohoto souboru počítá běhový kontejner, který zajišťuje, že aplikace bude vždy po spuštění v konzistentním stavu. Užití volných vazeb (*loose coupling*) hraje hlavní roli v deklarativním konceptu platformy NetBeans [5]. Každý modul je kromě *manifest* souboru popsán specifickými XML soubory, díky kterým platforma ví o jejich dostupnosti a umístění.

Platforma NetBeans je tvořena sadou základních modulů (obr. 3.6) potřebných pro spuštění vlastní aplikace a definici uživatelského rozhraní. Za tímto účelem je zpřístupněno velké množství aplikačních rozhraní (API) a tzv. poskytovatelů služeb značně zjednodušujících proces vývoje. Součástí této skupiny (obr. 3.6) je například často využívané aplikační rozhraní *Actions API* poskytující potřebnou funkcionalitu pro obsluhu uživatelských akcí, *Nodes API* zapouzdřující data různých typů a další viz [5].



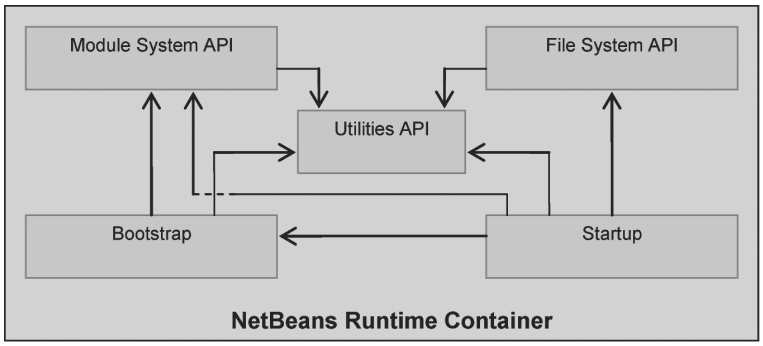
Obrázek 3.6: Architektura NetBeans Platform [5].

3.4.3 NetBeans Runtime Container

Základem platformy NetBeans a její modulární architektury je kontejner určený pro běh aplikací nazývaný NetBeans Runtime Container. Je složen z následujících pěti modulů[2], jejichž závislosti jsou zachyceny na obr. 3.7 viz [5].

- `Bootstrap`: Tento modul je spuštěn vždy jako první a zajišťuje registraci obsluhy příkazů a zavedení následujícího `Startup` modulu. Jinými slovy se tedy jedná o vstupní bod aplikace.
- `Startup`: Modul zabezpečující inicializaci systému souborů a modulů.
- `Module System`: Tento modul je zodpovědný za definici ostatních modulů, včetně jejich závislostí a specifického nastavení.
- `File System`: Modul zpřístupňující abstraktní virtuální datový systém, který umožňuje platformě nezávislý přístup k systému souborů daného operačního systému. Primárně se využívá pro načítání prostředků obsažených v modulech.
- `Utilities`: Tento modul poskytuje základní komponenty, které jsou vyžadovány pro intermodulární komunikaci.

[2]Jednotlivé termíny jsou záměrně ponechány v anglickém jazyce tak, aby korespondovaly s přiloženým obrázkem.



Obrázek 3.7: NetBeans runtime container [5].

Běhový kontejner představuje minimální podmnožinu modulů, které jsou nutné pro běh aplikace. Po spuštění této minimální konfigurace je však aplikace ihned ukončena, protože nejsou definovány žádné další akce a nastavení. Jakmile je spuštěň běhový kontejner, provede vyhledání dostupných modulů a následně se modul `Startup` postará o vytvoření interního registru složeného z těchto modulů. Každý modul je načten, jakmile je jednou potřeba, a to díky jeho registraci v modulu `Startup`. Během načítání umožňuje běhový kontejner modulům provádět určité typy úloh souvisejících s jejich životním cyklem, které jsou definovány v tzv. instalátoru [5].

Mimo jiné běhový kontejner poskytuje funkcionalitu pro dynamické načítání, uvolňování a odinstalování modulů, které byly načteny po dobu běhu aplikace. Této vlastnosti se využívá pro deaktivaci nepotřebných modulů a jejich aktualizaci [5].

3.4.4 NetBeans Classloader System

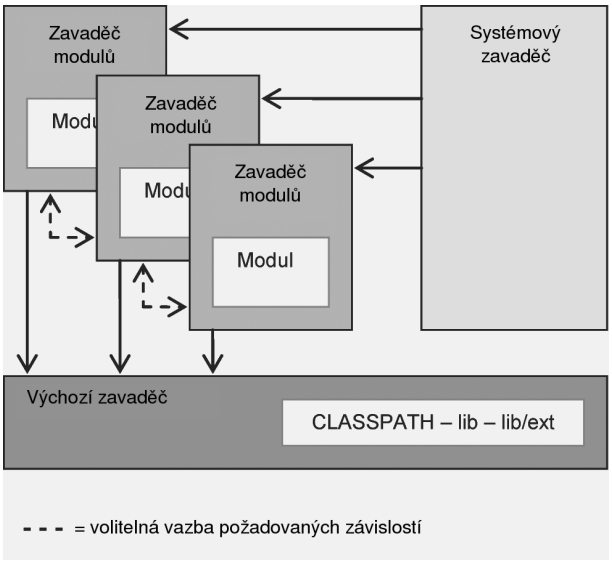
Nezbytnou částí běhového kontejneru je vlastní systém pro správu a dynamické načítání tříd neboli NetBeans Classloader System. Jak již bylo zmíněno dříve, jsou podporovány tři typy zavaděčů tříd. Nyní se jim budeme věnovat detailněji.

Většina tříd je načítána zavaděčem modulů, pouze v některých případech, kdy je třeba umožnit přístup k prostředkům modulů zvenku, je využito systémového zavaděče. Výchozí zavaděč (zavaděč aplikace) zajišťuje načítání zdrojů zpřístupněných systémovou proměnnou `CLASSPATH`. Tato proměnná je definována ve spouštěči aplikace, který je součástí platformy a zodpovídá za rozpoznání dostupného *Java Runtime Environment* [5]. Spouštěč je distribuován ve formě spustitelného souboru pro operační systémy podporované platformou NetBeans.

Zavaděč modulů a systémový zavaděč využívá typicky předem neurčený počet zavaděčů jako jejich předků. Vztahy mezi jednodlivými zavaděči jsou uvedeny na obrázku 3.8.

Zavaděč modulů

Každému modulu zaregistrovanému v systému náleží právě jedna instance tohto zavaděče. Primárně jsou načítány třídy z archivu JAR. Ve speciálních případech je možné načítat třídy i z několika archivů [5]. Jedná se zejména o moduly, které obalují existující knihovny tak, aby je bylo možné začlenit do platformy NetBeans.



Obrázek 3.8: Systém zavaděčů tříd platformy NetBeans [5].

Systémový zavaděč

Systémový zavaděč je vlastníkem všech instancí zavaděčů modulů. Teoreticky je tedy možné načítat všechny prostředky, které tyto moduly poskytují [5]. Lze ho využít v případech, kdy při vývoji aplikace narazíme na omezení platformy daná zapouzdřením jednodlivých částí.

Výchozí zavaděč

Výchozí zavaděč, často nazývaný také jako zavaděč aplikací, je volán prostřednictvím spouštěče (launcher) aplikací. Načítá třídy a ostatní zdroje definované ve výchozí systémové proměnné `CLASSPATH`, dále z adresářů knihoven `lib` a jejich podadresářů `ext` viz [5]. Pokud není archiv JAR rozeznán jako modul, typicky díky neplatnému souboru *manifest*, není začleněn do systému modulů. Tyto archivy jsou prohledány jako první v počáteční fázi startu aplikace a využívají se pro tzv. *branding*, případně také pro distribuci jazykových mutací modulů.

3.4.5 Systém modulů

Tento systém nese zodpovědnost za správu všech modulů v aplikaci. Zastává funkci vytváření zavaděčů modulů, jejich načítání, aktivaci a deaktivaci. Koncept tohoto systému je postaven na standardních Java technologiích, konkrétně vychází z technologie *Java Extension Mechanism*, jehož architektura je blíže specifikována v [15].

Modul je reprezentován archivem JAR. Vlastnosti modulu a závislosti na jiných modulech jsou popsány v souboru zvaném *manifest*. Formát tohoto souboru odpovídá standardu, který využívají archivy JAR s tím rozdílem, že platforma NetBeans tento formát rozšiřuje o další specifické atributy [5].

Kromě atributů obsažených v *manifest* souboru nepotřebuje většina modulů žádný další kód nutný k jejich instalaci a jsou tedy do platformy začleněny deklarativně. Další důležitým souborem poskytujícím specifické informace o aplikaci, jenž zároveň definuje způsob jakým je modul do platformy integrován, je XML soubor `layer.xml`. Vše, co modul do platformy přidává je specifikováno v tomto souboru [5].

Struktura modulu

Každý modul je prostým JAR archivem skládajícím se z následujících částí viz [5]:

- Soubor `MANIFEST.MF`. Pouze dostupnost tohoto souboru je povinná, protože slouží k identifikaci modulu.
- Soubor `layer.xml`[3]. Pokud modul obaluje knihovnu třetí strany, je tento soubor přebytečný a lze ho tedy vynechat.
- Soubory tříd.
- Další zdroje jako jsou ikony, soubory `Bundle.properties`, které se využívají k definici konstant a další.

Struktura tohoto archivu je graficky znázorněna diagramem na obr. 3.9.

Obrázek 3.9: Struktura modulu [5].

Moduly rovněž vyžadují speciální konfigurační XML soubor umístěný mimo archiv JAR. Tímto souborem oznamujeme platformě existeci daného modulu [5]. V ukázkovém diagramu se jedná o soubor `com-galileo-netbeans-module.xml`.

Typy modulů

Jak již bylo v textu dříve popsáno, jsou všechny moduly deklarovány v konfiguračním XML souboru a jejich relativní umístění vůči kořenovému adresáři daného *clusteru* aplikace je `config/Modules`. Obsah zmíněného adresáře je při startu aplikace čten systémem modulů. Moduly jsou načítány v závislosti na informacích poskytnutých jejich konfiguračními soubory. Konfigurační soubor modulu uvádí jeho jméno, verzi, umístění a způsob, jakým je načten [5]. Jeho struktura je uvedena ve výpisu 3.1.

Atribut `enabled` definuje, zda-li je modul načten, či nikoliv. Existují tři možnosti určující, jakým způsobem bude modul načten. Pokud je hodnota atributů `autoload` a `eager` nastavena na `false`, jedná se o běžný typ modulu nazývaný `regular`.

[3]Pro označení souboru `layer.xml` budeme v následujícím textu používat jeho originální název pouze v nezbytně nutných případech, jinak bude pro přehlednost uváděn zkrácený termín *layer*.

Výpis 3.1: Ukázka struktury konfiguračního souboru modulu

```
<?xml version="1.0" encoding="UTF-8"?>
<module name="com.elkoep.itpdesigner.itpd.core">
    <param name="autoload">false</param>
    <param name="eager">false</param>
    <param name="enabled">true</param>
    <param name="jar">modules/com-elkoep-itpdesigner-itpd-core.jar</param>
    <param name="reloadable">false</param>
    <param name="specversion">1.0</param>
</module>
```

Nastavíme-li však jeden z uvedených atributů na hodnotu `true`, jedná se o modul typu `autoload`, případně `eager` viz [5].

Modul typu `regular` je běžným typem modulu načítaným při startu aplikace. Čas potřebný k plnému načtení aplikace se prodlužuje o čas potřebný k inicializaci modulu.

Moduly typu `autoload` jsou načítány pouze v případě, jsou-li požadovány jiným modulem. Tento přístup je známý rovněž pod pojmem *lazy-loading*. Často se např. využívá při načítání kolekcí z databáze, kdy v určitý okamžik požadujeme pouze podmnožinu záznamů.

Posledním typem modulu je modul typu `eager`. Ten je načten pouze v případě, jsou-li dostupné všechny požadované závislosti. Uvedeme jednoduchý příklad. Mějme modul `C`, který je závislý na modulech `A` a `B`, které nejsou dostupné. V tomto případě nemá cenu načítat ani modul `C` [5].

Soubor `MANIFEST.MF` (dále *manifest*) textovou formou popisuje modul a jeho prostředí. Při čtení tohoto souboru se v první řadě kontroluje existence povinného atributu `OpenIDE-Module`. Je-li nalezen archiv JAR, je rozpoznán jako modul. Hodnota atributu bývá typicky stejná jako název samotného modulu (z ukázky struktury konfiguračního souboru z výpisu 3.1 se jedná o hodnotu `com.elkoep.itpdesigner.itpd.core`). Soubor *manifest* podporuje velké množství různých atributů. Jejich kompletní popis lze nalézt např. v [5].

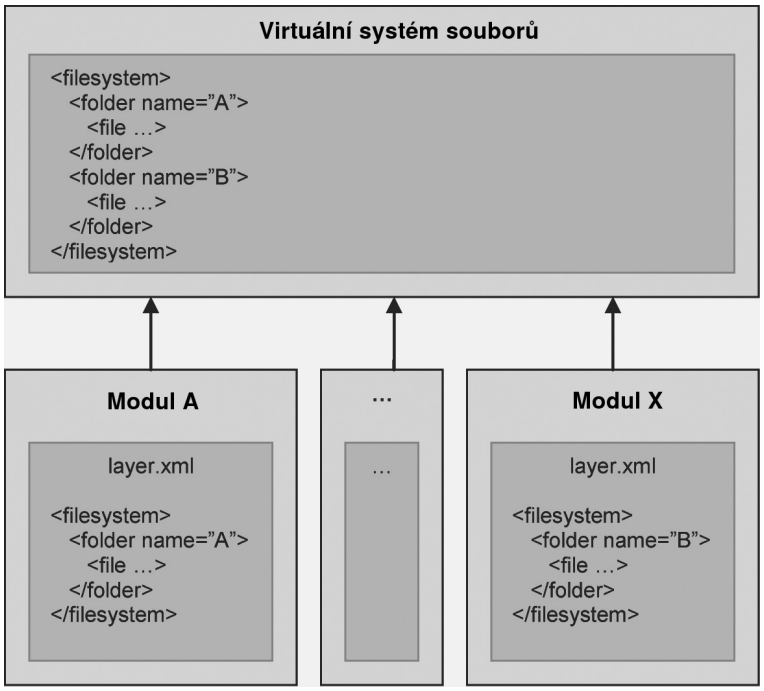
Dopňkem k *manifest* souboru je soubor `layer.xml` (dále jen *layer*), ve kterém je deklarativně popsáno, jakým způsobem je modul do platformy integrován. Částečně lze na tento soubor nahlížet jako na určitý typ rozhraní mezi modulem a platformou. O jeho existenci a umístění se platforma dozví právě z *manifest* souboru a atributu `OpenIDE-Module-Layer`. Soubor tvoří hierarchická struktura popisující adresáře, soubory a atributy. Během startu aplikace jsou všechny dostupné *layer* soubory spojeny do jednoho virtuálního systému souborů (viz obr. 3.10), který slouží ke konfiguraci běhu celé platformy [5].

Soubor *layer* obsahuje určité typy standardních složek nazývaných *extension points*, které mohou být definovány rozdílnými moduly [5]. Ukázka definice standardní složky, kde je do nástrojové lišty `Toolbar` přidána třída `SaveAction` implementující vlastní akci, je uvedena ve výpisu 3.2.

Výpis 3.2: Ukázka definice standarní složky

```
<folder name="Toolbars">
    <folder name="File">
        <file name="org-openide-actions-SaveAction.shadow">
            <attr name="originalFile" stringvalue="Actions/System/org-openide-actions-
                SaveAction.instance"/>
            <attr name="position" intvalue="444"/>
        </file>
</folder>
```

Jiné moduly mohou také do této nabídky přidávat vlastní položky. To je umožněno právě díky tomu, že jsou nakonec všechny soubory *layer* sloučeny do jednoho virtuálního



Obrázek 3.10: Struktura virtuálního systému souborů [5].

systému souborů. Okenní manažer zodpovědný za vytváření grafických prvků uživatelského rozhraní se na základě tohoto virtuálního systému souborů postará o vytvoření příslušné nabídky.

Pořadí, v jakém jsou záznamy *layer* souborů čteny, určuje hodnota atributu `position`, která v uvedeném príkladě (výpis 3.2) přímo ovlivní umístění tlačítka pro akci implementovanou třídou `SaveAction`.

Elementy `file` reprezentující objekty souborového systému mohou být v zásadě několika typů. V našem příkladě jsou uvedeny hned dva: `.instance` a `.shadow`.

Příponou `.instance` je oznámen požadavek na vytvoření instance daného objektu, která je vytvořena prostřednictvím standardního bezparametrického konstruktoru, nebo statické metody. Vytvoření těchto instancí zajišťuje systém souborů a aplikační rozhraní *Data System API*. Samotný kód pro vytváření instancí je následující [5]:

Výpis 3.3: Kód pro vytvoření instance objektu definované v souboru `layer.xml`

```
public static Object getInstance(String name) {
    FileSystem f = Repository.getDefault().getDefaultFileSystem();
    FileObject o = f.getRoot().getFileObject(name);
    DataObject d = DataObject.find(o);
    InstanceCookie c = d.getCookie(InstanceCookie.class);
    return c.instanceCreate();
}
```

Pokud třída neobsahuje standardní bezparametrický konstruktor, je možné její instanci vytvořit pomocí speciální statické metody definované atributem `instanceClass` viz [5].

Dalším typem jsou soubory s příponou `.shadow`, které slouží jako reference na již existující instance daných objektů. Ve většině případů se využívají pro referencování tříd umožňujících vytvoření jejich jediné instance, tedy o jedináčky (*singleton*).

Příkladem jsou různé akce, které lze jednoduše přiřadit k prvkům v nástrojových lištách, kontextových nabídkách atd.

Posledním typem jsou soubory s příponou `.settings`, které jsou rozšířením pro soubory `.instance`. Informace o typu třídy a způsobu, jakým je její instance vytvořena, definují v odděleném XML souboru. Na rozdíl od souborů `.instance` popisují kompletní hierarchii tříd, včetně implementovaných rozhraní viz [5].

Životní cyklus modulu

Pokud je nezbytně nutné nějakým způsobem ovlivnit standardní životní cyklus běhu modulu, je možné vytvořit pro každý modul tzv. instalátor.
Rozhraní *Module system API* k tomuto účelu zpřístupňuje třídu `ModuleInstall` a následující metody, které lze přetížit [5] :

- `validate()`: Metoda je vyvolána před tím, než je modul nainstalován a následně načten. Požadujeme-li například verifikaci licence modulu před jeho instalací, implementuje tato metoda příslušný kód. Pokud je vyvolána výjimka `IllegalStateException`, je zabráněno načtení a instalaci modulu.
- `restored()`: Tato metoda je vyvolána pokaždé, když je modul načten, a využívá se pro jeho inicializaci.
- `uninstalled()`: Metoda je vyvolána v okamžiku odstranění modulu z aplikace.
- `closing()`: Tato metoda je vyvolána těsně před tím, než je běh modulu ukončen. Například lze prostřednictvím této metody zobrazit dotazovací dialog pro potvrzení ukončení aplikace.
- `close()` Metoda je vyvolána právě tehdy, když je modul připravený na své ukončení.

3.4.6 Aplikační rozhraní platformy NetBeans

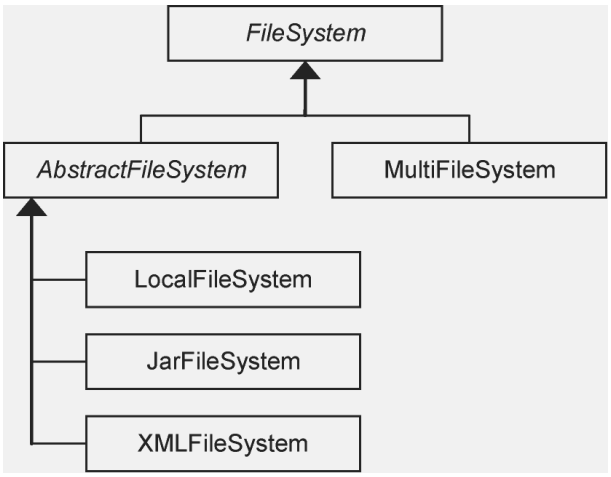
Kromě kvalitní modulární architektury nabízí také tato platforma sadu služeb zpřístupněných aplikačními rozhraními. Služby implementují funkcionalitu, která bývá často při vývoji aplikace požadována. V mnoha případech bychom se tak nevyhnuli její vlastní implementaci.

V následujících odstavcích budou tyto služby detailněji popsány. Některé z nich byly také využity při implementaci návrháře.

File Systems API

Každá aplikace postavená na platformě NetBeans má vlastní souborový systém nazývaný *System Filesystem* [9]. Tento systém je centrálním repozitářem, kde jsou uložena konfigurační data. Je dynamicky vytvářen po čas běhu aplikace sadou XML souborů *layer*.

Rozhraní *File System API* poskytuje transparentní přístup ke složkám a souborům. Umožňuje tím abstraktní přístup k souborům, které mohou být jak součástí virtuálního systému souborů (XML soubory *layer*), archivu JAR, tak i soubory a složky hostitelského operačního systému. Jak můžeme vidět na obr. 3.11, je tento fakt reflektován také hierarchií tříd a rozhraní.



Obrázek 3.11: Hierarchie tříd systému souborů [5].

Hlavní rozhraní systému souborů je specifikováno abstraktní třídou `FileSystem`. Další abstraktní třídou v hierarchii dané generalizací je třída `AbstractFileSystem`, která slouží jako nadtřída pro další specifické implementace systémů souborů (viz obr. 3.11).

Třída `MultiFileSystem` zastává funkci prostředníka (*proxy*) pro další implementace systému souborů [5].

FileObject

Data systému souborů, jako jsou složky a soubory, jsou reprezentována třídou `FileObject`, která obaluje standardní třídu `java.io.File`. Implementace této třídy poskytuje základní operace (čtení, zápis, vytváření, odstraňování, přejmenování a přesouvání) a možnost reakce na změny souborů či složek [5].

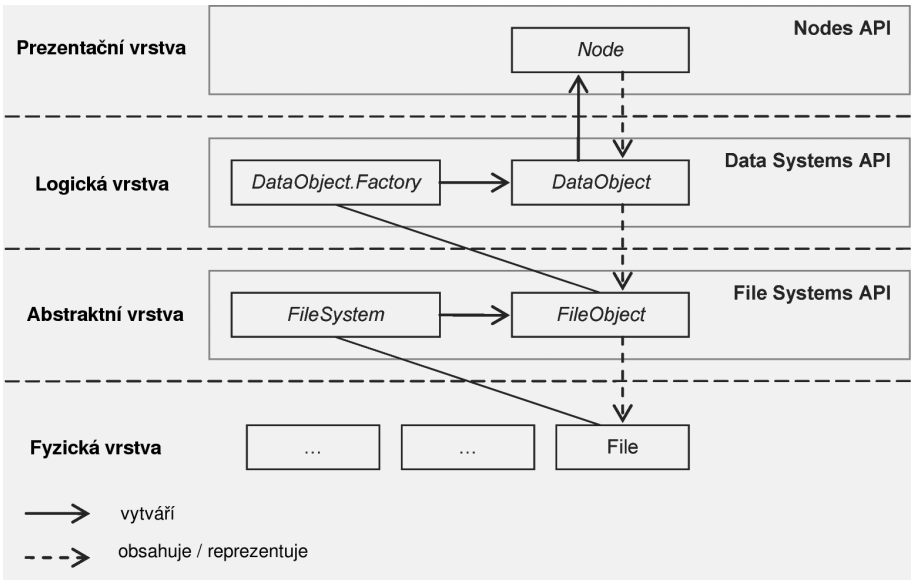
Data Systems API

Rozhraní *Data Systems API* poskytuje logickou vrstvu nad rozhraním *File Systems API* [5]. Vzájemný popis vztahů jednotilvých vrstev a objektů, které je reprezentují, popisuje diagram na obr. 3.12.

Zatímco třída `FileObject` pracuje s daty bez ohledu na jejich typu, třída `DataObject`, která ji obaluje, pracuje již se specifickým typem dat. Datové objekty (`DataObject`) umožňují přidat další typově specifickou funkcionalitu, která je zpřístupněna rozhraními či abstraktními třídami nazývanými *cookies* [5]. Jejich implementace je dostupná díky mechanismu *lookup*. Vlastnosti třídy `DataObject` jsou tedy dostupné zvenku a mohou být tak flexibilně rozšířeny.

Vytváření objektů `DataObject` zajišťují třídy `DataLoader`, `MultiFileLoader` a `UniFileLoader`, které implementují rozhraní `DataObject.Factory`. Koncept továrních metod sloužících pro vytváření instancí tříd `DataObject` je součástí platformy od verze 6.5 a může být deklarativně registrován v souboru *layer*. Blíže viz [5].

Díky tomu, že objekt `DataObject` pracuje již s konkrétním typem dat, je zodpovědný také za vytváření objektů `Node` reprezentujících tato data v GUI, tedy v prezentační vrstvě.



Obrázek 3.12: Vrstvy platformy NetBeans reprezentující data [5].

DataObject

V mnoha případech se pro objekt typu `DataObject` využívá jako předek třída `MultiDataObject`, která implementuje většinu abstraktních metod třídy `DataObject`, a jak její název napovídá, může také obsahovat více objektů typu `FileObject`, tzv. *secondary files* [5]. Této vlastnosti využívá např. editor pro grafický návrh a vytváření formulářů integrovaný do NetBeans IDE. Každý `DataObject` reprezentující formulář obsahuje hned trojici souborů (`FileObject`). Chceme-li, aby naše aplikace pracovala s konkrétními typy souborů, musíme pro ně vytvořit příslušné datové objekty. Ukázka jednoduchého datového objektu pro textový soubor *pub* (viz kapitola 2.2.1) je uvedena ve výpisu 3.4.

Výpis 3.4: Ukázka třídy reprezentující datový objekt pro soubor typu pub.

```
public class PubDataObject extends MultiDataObject {
    public PubDataObject(FileObject pf, MultiFileLoader loader)
        throws DataObjectExistsException, IOException {
        super(pf, loader);
        CookieSet cookies = getCookieSet();
    }
    @Override
    protected Node createNodeDelegate() {
        return new PubDataNode(this, getLookup());
    }
    @Override
    public Lookup getLookup() {
        return getCookieSet().getLookup();
    }
}
```

Povšimněme si nyní metody `createNodeDelegate()`. Tato metoda zajišťuje vytvoření příslušné instance třídy `Node` umožňující zobrazení dat v GUI. Třída `Node` náleží tedy do prezentační vrstvy a je součástí rozhraní *Nodes API*.

Jak již bylo zmíněno výše, využívají se k implementaci specifické funkcionality poskytované datovými objekty abstraktní třídy či rozhraní známé jako *cookies*. Instance těchto

rozhraní bývají typicky spravovány třídou `CookieSet` [5]. Díky tomuto faktu je možné funkcionalitu dynamicky spravovat, oddělit ji od datového objektu, a tak i zpřístupnit dalším podobným objektům. Prostřednictvím metody `getLookup()` poskytuje mechanismus *lookup* bezpečný přístup k rozhraním *cookies*.

Lookup API

Lookup API je implementováno stejnojmennou knihovnou. Jedná se o základní koncept celé platformy umožňující modulům jejich vzájemnou komunikaci a centralizovanou správu instancí objektů. Zjednodušeně lze tento koncept přirovnat ke kolekci typu `Map` známou ze standardní knihovny Java, ve které je klíč definován objektem `Class` a jeho instancí jako příslušnou hodnotou [5].

Hlavní myšlenkou konceptu je odstranění silných vazeb a vytvoření komunikačního kanálu mezi komponentami. Je umožněna také správa více instancí stejného typu, tedy instancí definovaných stejným klíčem.

Mechanismus *lookup* se také stará o nalezení tzv. poskytovatelů služeb (service providers), k jejichž instancím dodává podporu pro jejich deklarativní definici a *lazy-loading* (viz kapitola 3.4.5). Koncept navíc umožňuje předávání instancí objektů mezi moduly bez toho, aniž by tyto moduly o sobě znaly bližší informace [5].

Aplikace obsahuje typicky více než jeden *lookup*. Obvykle je nejvíce využívaný globální *lookup* poskytovaný platformou NetBeans. Navíc existují také komponenty poskytující svůj vlastní tzv. lokální *lookup*, např. komponenta GUI `TopComponent`. A nakonec je umožněno také vytvořit vlastní *lookup* [5].

Služby

Hlavní rolí konceptu *lookup* je funkce dynamického vyhledávače služeb, který dovoluje oddělit rozhraní služby od její implementace. Modulům je tak umožněno jednoduše využívat funkcionalitu bez toho, aniž by věděly cokoliv o její implementaci.

Pro dynamické poskytování a výměnu služeb je upřednostnována jejich deklarativní registrace k příslušnému *lookupu*. Registraci lze provést následujícími dvěma způsoby [5]:

- Přidáním speciálního konfiguračního souboru do adresáře `META-INF/services` modulu, který definuje implementaci dané služby.
- Přidáním definice do XML souboru *layer*.

Přístup ke službám poskytuje globální *lookup*. Ten lze jednoduše získat statickou metodou `Lookup.getDefault()`. Právě tento *lookup* se využívá k nalezení služeb registrovaných deklarativním způsobem.

Deklarativní přístup umožňuje vytvoření příslušné instance implementace služby až v okamžik, kdy je poprvé vyžadována (*lazy-loading*) a také registrovat více implementací pro jednu službu.

Problematika služeb a jejich poskytovatelů je velmi obsáhlá. Její bližší specifikace je popsána např. v [5]. Pro potřeby této práce by však měl postačovat pouze stručný nástin této problematiky.

Komunikace mezi moduly

Rozhraní *Lookup API* a *SPI (Service Provider Interface)* se využívá pro vytvoření komunikačního kanálu mezi komponentami různých modulů bez toho, aniž by se staly navzájem závislými [5].

Aplikační rozhraní *Lookup API* také poskytuje tovární metody, které umožňují vytvořit lokální *lookup* a současně také prostředek pro monitorování jeho změn.

Sadu nástrojů schopných vytvořit lokální *lookup* poskytuje třída `Lookups` díky svým statickým metodám.

Následující fragment zdrojového kódu ukazuje, jakým způsobem lze jednoduše vytvořit lokální *lookup* obsahující jediný objekt.

```
Room room = new Room();
Lookup lkp = Lookups.singleton(room);
```

Chceme-li vytvořit *lookup* obsahující několik objektů, provedeme to např. následovně:

```
Lookup lkp = Lookups.fixed(new ItemShutter(), new ItemCamera());
```

Požadujeme-li spravovat *lookup* dynamicky, je nutné vybrat implementaci, která to umožňuje. Nejčastěji se k tomuto účelu využívá tříd `InstanceContent` a `AbstractLookup`, viz následující ukázka zdrojového kódu.

```
InstanceContent = new InstanceContent();
Lookup lkp = new AbstractLookup(content);
content.add(new Room());
```

Aby bylo možné lokální *lookup* sdílet s dalšími objekty, je třeba ho zpřístupnit tak, že daná třída implementuje rozhraní `Lookup.Provider` obsahující jedinou metodu `getLookup()`.

Použitím speciální třídy `ProxyLookup` docílíme vytvoření prostředníka, který umožňuje sloučit více objektů typu *lookup* do jednoho.

Nodes API

Spolu s *Lookup API* patří toto rozhraní k nejvyužívanějším rysům platformy NetBeans. V hierarchii vrstev reprezentujících data je na nejvyšší úrovni (viz obr. 3.12) nazývané jako prezentační vrstva. S tímto aplikačním rozhraním je úzce spjato rozhraní *Explorer & Property Sheet API*, které zastává funkci správce uzlů[4] [5].

Uzly se nejčastěji využívají pro vizuální reprezentaci dat v GUI aplikace. Každému uzlu je možné přiřadit uživatelskou akci sloužící pro interakci s daty uloženými na nižší vrstvě.

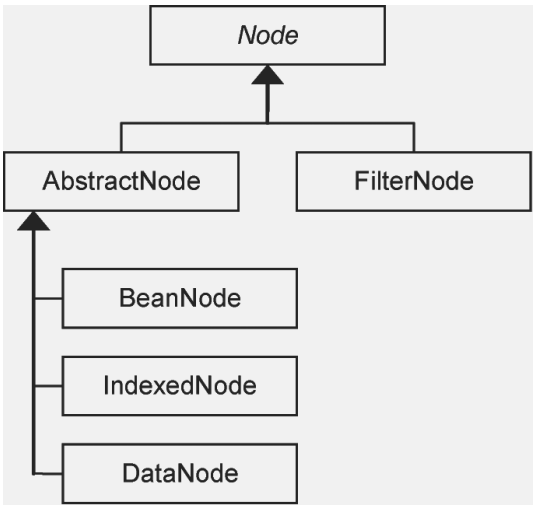
Jedním z nedostatků aplikačního rámce *Swing* je nutnost tvorby odlišných datových modelů pro různé komponenty GUI. Několikanásobné zobrazení jedné sady dat v různých vizuálních komponentách je tak poměrně komplikovaný problém. *Nodes API* tento problém řeší a umožňuje vytvořit jeden univerzální model [9].

Při implementaci je nutné respektovat fakt, že uzly jsou součástí prezentační vrstvy a měly by být tedy oproštěny od implementace tzv. business logiky. Tato logika by měla být součástí objektu `DataObject`, který k příslušnému uzlu náleží [5].

Předkem všech uzlů v hierarchii tříd (viz obr. 3.13), které rozhraní *Nodes API* poskytuje, je abstraktní třída `Node`, definující hlavní rozhraní a chování uzlu [5].

Třída `AbstractNode` umožňuje jednoduše vytvořit instanci třídy `Node`.

[4] V následujícím textu budeme pro anglický výraz node užívat české synonymum uzel.



Obrázek 3.13: Hierarchie tříd reprezentujících uzly [5].

Třída `FilterNode` vytváří pouze zástupce (prostředníka) delegujícího volání příslušných metod rodičovskému uzlu. Využívá se pro přizpůsobení prezentace rodičovského uzlu v různých komponentách aplikace.

Pro reprezentaci objektů `JavaBean` se používají uzly `BeanNode`. Dalším typem jsou uzly `IndexNode` umožňující použití indexu pro organizaci svých potomků. A nakonec uzly typu `DataNote` reprezentující objekty `DataObject` [5]. Právě vytvoření uzlů `DataNode` zajišťuje výše zmíněná metoda `createNodeDelegate()` (viz kapitola 3.4.6) a tvoří tak styčné místo mezi prezentační a logickou vrstvou reprezentace dat platformy NetBeans.

Kontejner Node

Každý objekt `Node` může obsahovat objekty `Children` a poskytovat jim také požadovaný kontejner [5]. Objekty `Children` zastupují poduzly objektu `Node`.

Použitá terminologie může vést k matoucí interpretaci pojmů uzlu `Node` a jeho potomků `Children`. Uvažujme příklad, kdy uzel reprezentuje objekt `DataObject` zapouzdřující soubor formátu XML. Pomineme-li nyní fakt, že `DataObject` může obsahovat více souborů různých typů, dojdeme úvahou k relaci typu 1:1 mezi objektem `DataObject` a uzlem `Node`. Z předchozího textu také víme, že `DataObject` je již seznámen s typem dat daného souboru a konkrétně tedy pro případ XML souboru zná také jeho strukturu, která může být využita pro tvorbu dalších poduzlů [5].

Naopak pro datové objekty zapouzdřující binární soubory pozbývá vytváření dalších poduzlů smyslu. Výjimkou mohou být vlastnosti tohoto souboru jako např. jeho velikost na disku atp.

Objekt `Children` je zodpovědný za vytváření a strukturování příslušných potomků. Součástí každého potomka je jeho reference na rodičovský uzel [5].

Tabulka 3.1 (viz následující strana) stručně popisuje dostupné třídy implementující různé typy kontejnerů, jejich vlastnosti a použití [5].

K asynchronnímu vytváření objektů `Children` se využívá generická abstraktní tovární třída `ChildFactory<T>` (třídy uvedené v tabulce 3.1 tento přístup nepodporují). Generickým typem `T` je třídě předán typ objektu, který využívá kolekce klíčů, na jejichž základě jsou vytvořeny příslušné uzly.

Třída	Použití
`Children.Array`	Rodičovská třída všech ostatních tříd `Children`. Kontejner, který tato třída reprezentuje, je typu `Array`. Jednotlivé uzly jsou tedy přidávány na konec tohoto pole, tím je také dáno jejich pořadí.
`Children.Keys<T>`	V implementaci se často využívá právě této rodičovské třídy. Uzlům je přiřazen určitý klíč, který zároveň slouží také pro jejich řazení.
`Children.Map<T>`	Uzly jsou uloženy do datové struktury `Map` a provázány s klíči, které také slouží k jejich odstranění.
`Children.SortedArray`	Tato třída rozšiřuje funkcionalitu třídy `Children.Array` a dodatečně tak přidává možnost řazení uzlů.
`Children.SortedMap<T>`	Tato třída je velmi podobná třídě `Children.SortedArray` s tím rozdílem, že rodičovskou třídou je zde `Children.Map<T>`.

Tabulka 3.1: Typy kontejnerů pro objekty `Children` [5].

Typicky je asynchronní přístup vyžadován pro časově náročné operace, jako je např. čtení dat z databáze, nebo další blokující vtupně/výstupní operace.

Chceme-li využít vlastností třídy `ChildFactory`, musíme implementovat vlastní dceřinou třídu a její instanci předat statické metodě `Children.create` jako argument. Signatura této metody je následující:

```
create (ChildFactory <T> factory, boolean asynchronous)
```

Je-li argument `asynchronous` nastaven na hodnotu `true`, je při požadavku na získání potomků objektu `Node` vytvořeno nové vlákno volající implementaci abstraktní metody `createKeys(List <T> toPopulate)` třídy `ChildFactory`. Tato metoda zajistí vytvoření klíčů generického typu `T`. Pro každý klíč v kolekci je následně zavolána metoda `createNodeForKey(T key)`, která vytvoří příslušný objekt `Node`.

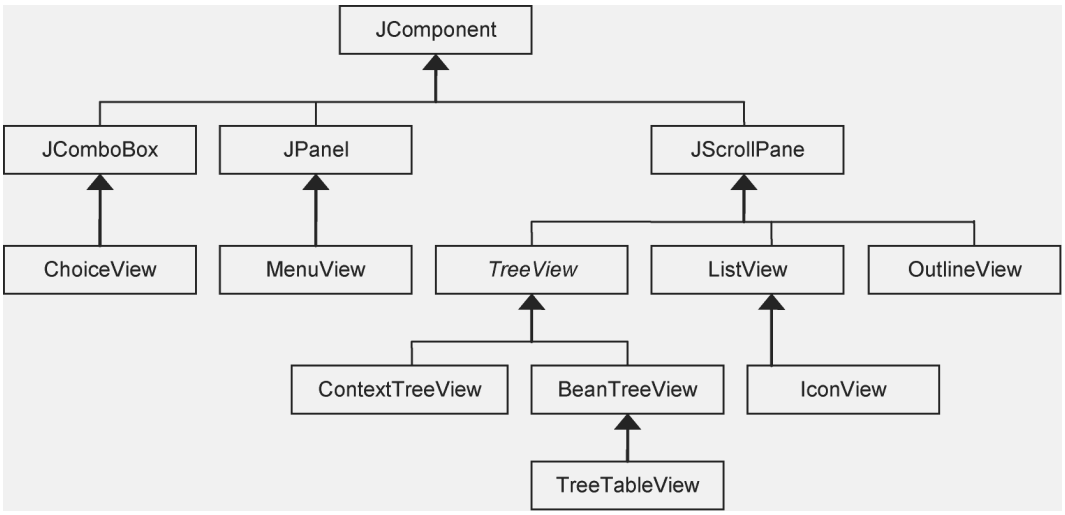
Každému uzlu lze prostřednictvím kontextové nabídky dynamicky přiřazovat uživatelské akce. Akce kontextové nabídky jsou pro uzly, které nereprezentují objekty `DataObject`, definovány metodou `getActions()`. V opačném případě jsou načteny ze souboru *layer* metodou `actionsContext()` třídy `DataLoader` [5].

Přetížením metody `getPreferredAction()` je možné definovat, která akce bude vyvolána dvojitým klikem na vybraný uzel. Pokud metoda navrátí hodnotu `null`, je použita první akce z pole, které je návratovou hodnotou metody `getActions()`.

Explorer & Property Sheet API

Aplikační rozhraní *Explorer & Property Sheet API* je nástrojem pro správu a zobrazení uzlů. Poskytuje sadu grafických komponent (*view*) umožňující prezentaci uzlů v některé z mnoha struktur [5]. Hierarchie tříd těchto komponent je znázorněna na obr. 3.14.

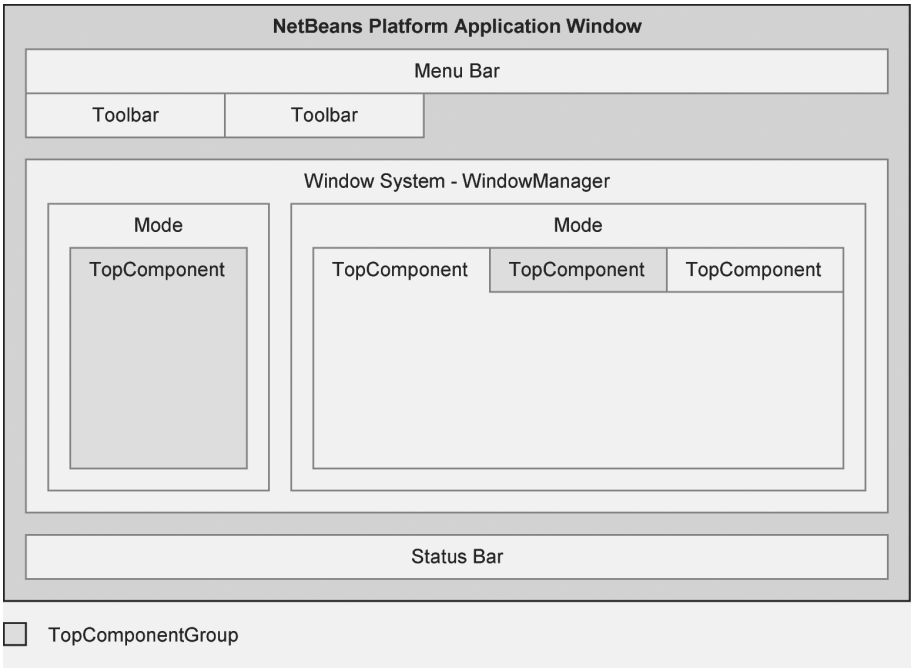
Za správu zobrazení uzlů nese zodpovědnost třída `ExplorerManager`. Detailní popis těchto API lze nalézt v [5].



Obrázek 3.14: Hierarchie tříd grafických komponent *Explorer API* [5].

3.4.7 GUI platformy NetBeans

Platforma NetBeans poskytuje vlastní systém pro správu oken. Zjednodušeně lze říct, že se jedná o kontejner zahrnující základní grafické prvky jako je panel nabídek (Menu bar), nástrojová lišta (Toolbar), okna (TopComponent), jejichž obsah utvářejí moduly, a stavový řádek (Status Bar). Samozřejmě platforma poskytuje modulům také prostředek pro snadný přístup ke všem těmto komponentám [5]. Rozložení jednotlivých komponent v rámci aplikace platformy NetBeans znázorňuje obr. 3.15.



Obrázek 3.15: Struktura komponent aplikace platformy NetBeans [5].

Problematika tvorby a konfigurace jednotlivých komponent je velmi obsáhlá a bude tedy v následujících odstavcích popsána jen stručně. Podrobný popis lze však nalézt např. v [5].

Panel nabídek

Hlavní panel nabídek (Menu bar) vytváří platforma skrze vlastní souborový systém nazývaný *System Filesystem* (viz kapitola 3.4.6). Každá nabídka, stejně jako její položky, je definována souborem *layer*. Díky tomu může každý modul do této nabídky deklarativně přidávat vlastní položky a implementovat jejich akce [5].

Lišty nástrojů

Aplikace založené na platformě NetBeans obsahují oblast, do které může uživatel přidávat své vlastní lišty nástrojů (Toolbars).

Vytváření lišt nástrojů a akcí příslušejících jejím položkám probíhá podobným způsobem jako u panelu nabídek, tedy definicí uvedenou v souboru *layer*. Konkrétně je nutné definovat virtuální složku `Toolbars` a další její podsložky zastupující jednotlivé lišty nabídek. Způsob, jakým je možné v souboru *layer* definovat vlastní lištu nástrojů viz [5], je uveden ve výpisu 3.5.

Výpis 3.5: Ukázka definice lišty nástrojů

```
<folder name="Toolbars">
    <folder name="File">
        <file name="org-openide-actions-SaveAction.shadow">
            <attr name="originalFile" stringvalue="Actions/System/org-openide-actions-
                SaveAction.instance"/>
        </file>
    </folder>
</folder>
```

Způsob a pořadí, v jakém budou lišty zobrazeny, je definován specifickým XML souborem. Implicitně poskytuje platforma standardní konfiguraci definovanou souborem `Standard.xml` [5]. Snadno lze však vytvořit vlastní konfigurační soubor a jeho referenci uvést na příslušném místě v souboru *layer*.

Systém pro správu oken

Platforma NetBeans poskytuje pro správu a zobrazení všech oken aplikace vlastní aplikační rámec umožňující přizpůsobit vzhled uživatelského rozhraní našim potřebám.

Systém pro správu oken, stejně jako i samotná aplikace postavená na platformě NetBeans, je primárně založen na tzv. *document-based* architektuře [5]. Mezi hlavní zástupce těchto aplikací patří bezesporu různé textové či tabulkové procesory.

Základní stavební jednotkou této architektury, jak již její název napovídá, je dokument. Dokument typicky reprezentuje množinu sémanticky podobných dat. Hlavní částí aplikace je tedy grafická komponenta umožňující zobrazení a editaci dokumentů, mezi kterými lze navzájem přepínat. Ostatní prvky uživatelského rozhraní jsou často svázány s aktivním dokumentem a jsou rozmístěny okolo této centrální grafické komponenty.

Systém pro správu oken platformy NetBeans je složen z kontejnerů obsahujících další okna, mezi kterými lze jednoduše přecházet. Kontejner je reprezentován třídou `Mode`. Pokud požadujeme, aby nově vytvořené okno bylo možné začlenit do příslušného kontejneru, musí třída, která ho implementuje, dědit od třídy `TopComponent` (viz obr. 3.15). Vykreslení a zobrazení každého okna zajišťuje `WindowManager` [5].

Třída `TopComponent` je poskytována aplikačním rozhraním *Window System API*. Tato třída slouží k vytváření oken, která jsou integrována do aplikace platformy NetBeans, a

může existovat vždy jen jako součást kontejneru `Mode` [5]. Vytváříme-li grafickou aplikaci, je tedy nutné, aby moduly implementovaly komponenty typu `TopComponent`.

Stavový řádek

Okno aplikace platformy NetBeans obsahuje integrovaný stavový řádek (Status Bar). Přístup k němu umožňuje globální servisní abstraktní třída `StatusDisplayer`. Díky jeho rozšiřitelnosti je možné do stavového řádku přidávat i své vlastní komponenty [5].

Standardní implementaci stavového řádku lze získat metodou `getDefault()` třídy `StatusDisplayer`. Pokud nebyla registrována vlastní implementace stavového řádku, navrátí tato metoda standardní implementaci poskytovanou platformou [5]. Zobrazení textu ve stavovém řádku je možné z jakékoliv části aplikace následovně:

```
StatusDisplayer.getDefault().setStatusText("This is my first status!");
```

Kapitola 4

Návrh a implementace

Součástí této práce byl požadavek na vytvoření mobilní aplikace iMM Tablet, jejíž účel byl stručně nastíněn v kapitole 3. Dalším požadavkem bylo vytvoření desktopové aplikace grafického návrháře (dále ITP Designer), která slouží jako nástroj zjednodušující konfiguraci mobilní aplikace iMM Tablet.

Primárně se tato kapitola zabývá návrhem, na který navazuje popis samotné realizace požadovaných aplikací a do jisté míry také navazuje na předchozí kapitolu, kde byly diskutovány a následně detailně analyzovány zvolené prostředky pro implementaci.

Pro dosažení požadovaných výsledků, tedy splnění požadavků na vytvoření příslušných aplikací, byly respektovány zásady softwarového inženýrství a jeho základní fáze životního cyklu aplikace, ke kterým náleží [7]:

- Analýza požadavků
- Návrh aplikace
- Implementace
- Integrace a nasazení
- Provoz a údržba

Text této práce se snaží tento koncept následovat. Analýze požadavků byla věnována předchozí kapitola, další dvě fáze jsou obsahem této kapitoly, a nakonec nasazení aplikací je diskutováno v kapitole následující. Jediná fáze, jež není touto prací pokryta je provoz a údržba, vyplývá to z charakteru práce a jejího časového horizontu.

Pro obě aplikace byl zvolen iterativní model životního cyklu. Pro iterativní životní cykly s přírůstky (označované někdy také jako evoluční nebo inkrementální) je charakteristické, že zahrnují iterace. Iterace znamená opakovaný průchod fázemi životního cyklu s cílem obohatit vytvářený systém v každé iteraci o nějaké rozšíření či vylepšení, které budeme nazývat přírůstkem [7].

4.1 iMM Tablet

Tímto pojmem budeme označovat mobilní zařízení s operačním systémem Android, které je schopno komunikovat prostřednictvím bezdrátové technologie Wi-Fi, a také samotnou aplikaci pro toto zařízení. Vzhledem ke zvolené platformě se může jednat jak o tablet, tak i chytrý telefon.

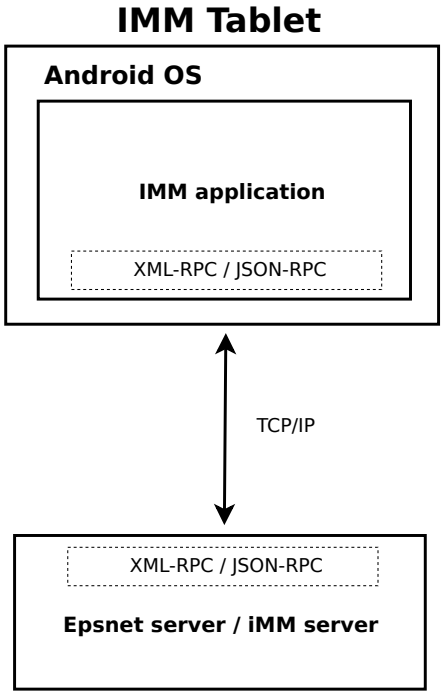
iMM Tablet je v podstatě mobilním řešením panelu iMM Touch a slouží pro ovládání systému iNELS a iMM. Jedná se však pouze o prototyp umožňující jen základní komunikaci se stávajícím řešením.

4.1.1 Komunikace

Během analýzy bylo odhaleno úskalí, které skýtá stávající řešení komunikace postavené na protokolu *PYRO*. Díky tomuto protokolu (viz kapitola 2.3.4) nebylo možné realizovat komunikaci se systémem iNELS a iMM transparentně. Toto styčné místo je však pro vytvoření požadované aplikace velmi zásadní. V následujících odstavcích bude tento problém blíže rozebrán a bude navrženo jeho možné řešení.

Společnosti ELKO EP, s.r.o bylo navrženo, aby do stávajícího řešení byla přidána podpora pro komunikaci založená na protokolu XML-RPC, nebo JSON-RPC. Oba tyto protokoly jsou podobné technologii *PYRO* s tím rozdílem, že jejich otevřený datový formát umožňuje komunikaci mezi různými platformami nezávislou na programovacím jazyce a může tak být na obou stranách komunikačního kanálu libovolný.

Při osobní návštěvě byl předložen obr. 4.1 popisující zjednodušený návrh řešení zmíněného problému.



Obrázek 4.1: Návrh řešení problému komunikace mobilní aplikace a stávajícícho systému.

Po vzájemné diskuzi bylo schváleno řešení postavené na technologii XML-RPC a přislíbeno, že vývoj serverové strany zabezpečí společnost ELKO EP, s.r.o.

Protokol XML-RPC

Protokol XML-RPC popisuje sekvenci a strukturu požadavků a odpovědí, které je možné zpracovat vzdáleným počítačovým systémem prostřednictvím protokolu HTTP, který je

součástí aplikační vrstvy architektury TCP/IP. Data požadavků a odpovědí jsou zapouzdřena značkovacím jazykem XML. Specifikace tohoto protokolu obsahuje popis dostupných datových typů běžně používaných v programovacích jazycích [6]. Protokol XML-RPC je postavený na standardizovaném protokolu vzdáleného volání procedur – RPC.

První implementace byla vytvořena již v roce 1998. V současné době není tento protokol dále vyvíjen, nicméně se stal předlohou pro protokol SOAP [23].

Přestože byl vývoj protokolu XML-RPC ukončen, je stále hojně využíván webovými službami jako jednodušší varianta protokolu SOAP. Bližší popis protokolu SOAP lze nalézt např. v [8].

Velkou výhodou tohoto protokolu je jeho nezávislost na operačním systému a dostupnost velkého množství existujících implementací pro různé programovací jazyky (Python, Perl, PHP, Ruby, Java, C/C++, Objective-C, C# a další).

Volání příslušné serverové metody je iniciováno klientským požadavkem, jehož tělo může vypadat např. následovně:

Výpis 4.1: Ukázka klientského požadavku

```
<?xml version='1.0'?>
<methodCall>
  <methodName>playVideo</methodName>
     <params>
     <param>
        <value><string>10.10.3.157</string></value>
     </param>
     <param>
        <value>
           <string>/home/imm/Video/Twilight/HD trailer Twilight 1080p.mov</string>
        </value>
     </param>
     </params>
</methodCall>
```

Tento požadavek server zpracuje a navrátí úspěšnou, či chybovou odpověď.

Android a XML-RPC

Pro komunikaci prostřednictvím protokolu XML-RPC byla zvolena knihovna *android-xmlrpc*, jejíž zdrojové kódy jsou dostupné pod licencí Apache verze 2.0.

Volání vzdálených procedur a čtení jejich návratových hodnot patří mezi blokující operace. Pro aplikace má vyvolání časově náročné operace nežádoucí vliv na uživatelské rozhraní.

Z tohoto důvodu byla implementována třída `XMLRPCMethod` (kompletní výpis jejího zdrojového kódu je součástí přílohy A této práce), která obaluje volání jednotlivých RPC metod do vláken, jež umožňují provedení asynchronní akce po obdržení požadované návratové hodnoty. Ukázka volání této metody je uvedena ve výpisu 4.2. Již při rané fázi návrhu aplikace iMM Tablet bylo rozhodnuto o jejím logickém členění do dvou částí pojmenovaných `immtablet-controller` a `immtablet`.

Část `immtablet-controller` je knihovnou poskytující jednoduché aplikační rozhraní (API) části `immtablet`, která implementuje uživatelské rozhraní aplikace a s ním svázanou funkcionalitu.

Výpis 4.2: Ukázka volání vzdálené RPC metody

```
public void play() {
    RPCMethod method = new RPCMethod(RPCMethodConstants.PLAY, client, handler, new
        RPCMethodCallBack() {
      @Override
        public void callFinished(Object result) {
            log.i(STATUS.isPlaying);
    }
  });
  method.call();
}
```

Aplikační rozhraní knihovny `immtablet-controller` abstrahují volání vzdálených procedur tak, že část aplikace `immtablet` je odstíněna od použité komunikační technologie. Všechny metody sloužící pro volání odpovídajících vzdálených procedur jsou deklarovány v sadě rozhraní nazývaných jako tzv. kontrolery. Knihovna také poskytuje výchozí implementaci těchto rozhraní umožňující volání vzdálených procedur prostřednictvím protokolu XML-RPC. Diagram tříd znázorňující jednotlivá rozhraní (kontrolery) a jejich implementaci je uveden na obr. 4.2 [1].

Přidání podpory pro jiný komunikační protokol

Pokud při dalším vývoji bude rozhodnuto o změně komunikačního protokolu, neměla by tato změna díky stávajícímu návrhu vést k výrazným změnám v části aplikace `immtablet`.

Knihovna `immtablet-controller` však bude muset být rozšířena o příslušnou funkcionalitu podporující zvolený komunikační protokol. V první řadě musí být implementován klient, který komunikuje se serverem tímto protokolem a obsahuje všechny implementační detaily — např. jakou formou jsou reprezentována data požadavků, odpovědí a jakým způsobem jsou transformována na jednoduché, či složené datové typy jazyka Java.

Pro známé a využívané protokoly lze na internetu často nalézt hned několik knihoven pro různé programovací jazyky. V takovém případě lze tuto knihovnu začlenit jako součást knihovny `immtablet-controller` a využít existující implementaci klienta tak, jak je tomu např. u výchozí implementace komunikace podporující protokol XML-RPC.

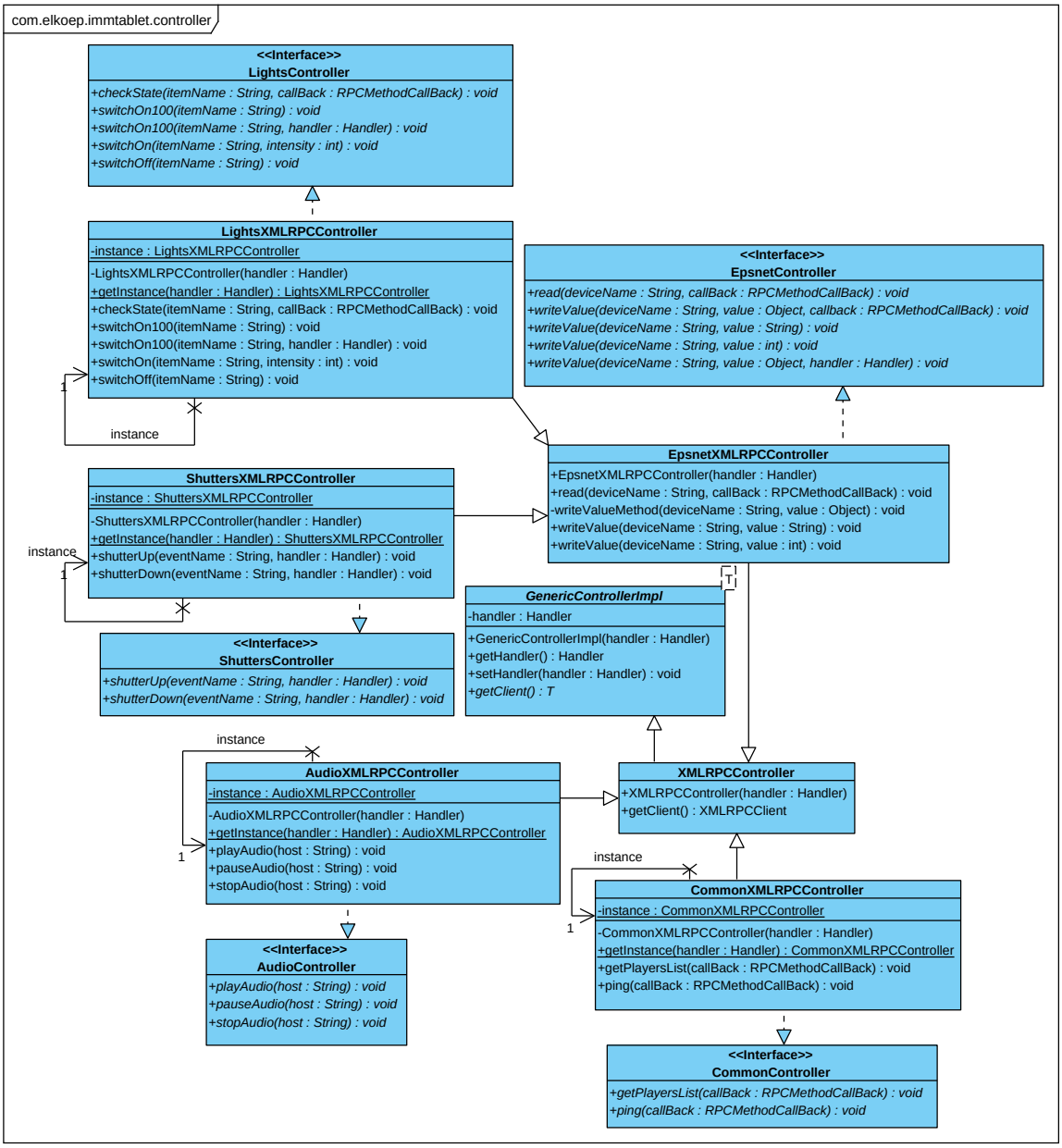
Dalším krokem, který musí bezprostředně následovat, je vytvoření rodičovské třídy pro všechny ostatní kontrolery (v diagramu na obr. 4.2 této třídě odpovídá `XMLRPCController`). Předkem této třídy musí být abstraktní generická třída `GenericControllerImpl<T>`. Generický typ `T` určuje, jakého klienta bude tento kontroler při komunikaci využívat.

Díky tomuto konceptu lze tedy jednoduše implementovat vlastní kontrolery, které budou podporovat požadovaný komunikační protokol, např. výše zmíněný protokol JSON-RPC.

Z diagramu mimo jiné také vyplývá, že vlastní implementace vzdáleného volání procedur, je ukryta za příslušnými rozhraními, která jsou zpřístupněna aplikaci `immtablet`. Pokud implementujeme vlastní kontroler využívající jiného komunikačního protokolu, stačí provést výměnu kontroleru typicky pouze na **jediném** místě aplikace. Každá třída reprezentující kontroler je navíc implementována s použitím návrhového vzoru jedináčka (*singleton*), a tudíž není nutné složitě předávat instance kontrolerů napříč celou aplikací.

Návrh se tak stává přehlednější a čistší. Zohledňuje také požadavek na budoucí vývoj aplikace, který předpokládá podporu přímé komunikace s centrální jednotkou systému iNELS (viz kapitola 2.1.1).

[1] Pro zachování přehlednosti diagramu byly vynechány některé metody.



Obrázek 4.2: Diagram tříd aplikačních rozhraní knihnovny `immtablet-controller`.

4.1.2 Uživatelské rozhraní

Filozofie práce s mobilními zařízeními a jejich aplikacemi je poněkud odlišná než u běžných desktopových aplikací. Klávesnice obsahuje nutné minimum tlačítek, či je zcela vynechána a ovládání tak probíhá skrze dotykový displej. Při návrhu aplikace pro mobilní zařízení musíme také dbát na to, aby bylo ovládání co nejvíce ergonomické. Po spuštění aplikace její okno typicky zabere, až na některé výjimky, veškerou dostupnou plochu displeje. Mezi výjimky patří např. obdoba stavového řádku obsahující základní informace o stavu nabití baterie, síle signálu mobilního operátora atp.

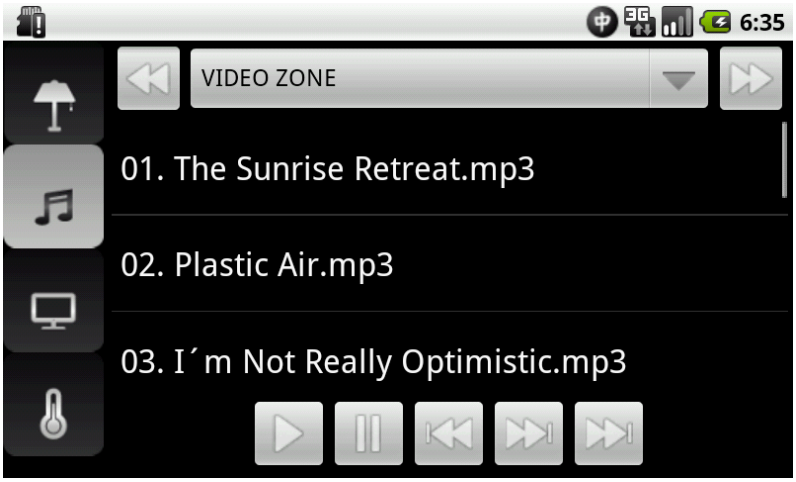
1. iterace

Jak již bylo zmíněno na začátku této kapitoly, byl pro vývoj zvolen iterativní model životního cyklu.

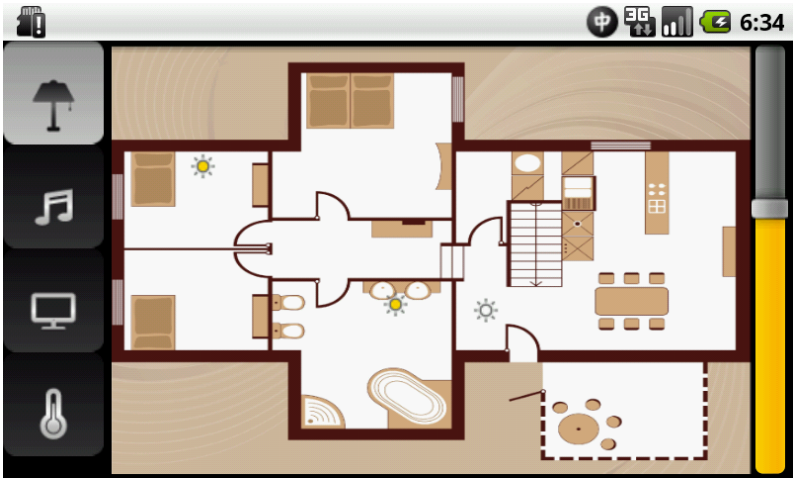
Součástí 1. iterace byl návrh a implementace prototypu knihovny pro komunikaci (viz kapitola 4.1.1) a uživatelského rozhraní aplikace iMM Tablet.

Prototyp rozložení uživatelského rozhraní byl inspirován konkurenčním komerčním produktem od společnosti Iridium Mobile Ltd. Tato firma nabízí vlastní mobilní řešení pro ovládání systémů domácí automatizace, včetně systému iNELS. Hlavní podporovanou platformou jsou mobilní zařízení iPhone a iPad s operačním systémem iOS od společnosti Apple Inc. Tato zařízení se těší ve světě velké oblibě, avšak u nás nejsou zdaleka tak rozšířena. Pro vytvoření vlastního řešení iMM Tablet postaveného na platformě Android vedl také fakt, že mobilní aplikace od společnosti iRidium Mobile Ltd. nepodporovala ovládání systému multimédií iMM společnosti ELKO EP, s.r.o.

Na obr. 4.3 a 4.4 můžeme vidět prototyp rozložení uživatelského rozhraní aplikace iMM Tablet pro přehrávání hudby a ovládání osvětlení.



Obrázek 4.3: Prototyp rozvržení uživatelského rozhraní pro přehrávání hudby.



Obrázek 4.4: Prototyp rozvržení uživatelského rozhraní pro ovládání osvětlení.

Hlavní nevýhodou tohoto rozložení je neefektivní využití volného místa na displeji. V některých místnostech může být velké množství osvětlovací techniky, a pokud se např. podíváme blíže na obr. 4.4, vedlo by zvolené rozložení k nepřehlednosti, a navíc i k nepohodlnému ovládání aplikace.

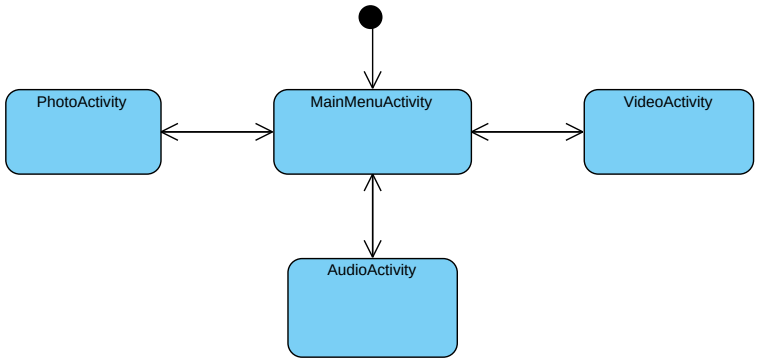
Po prezentaci a následující vzájemné diskusi byl společností ELKO EP, s.r.o. navíc vznesen také požadavek na dodržení příslušného *brandingu* tak, aby bylo docíleno podobného vzhledu uživatelského rozhraní jako u řešení iMM Touch.

2. iterace

V první řadě byly na základě nově poskytnuté implementace serverové strany systému iMM rozšířeny kontrolery pro ovládání multimédií.

V návaznosti na získaných poznatcích z předchozí iterace byl znovu proveden návrh rozložení uživatelského rozhraní aplikace tak, aby co nejvíce odpovídal vznesenému požadavku na jednotný *branding*.

Návrh vedl k vytvoření hlavní aktivity (obrazovky) zastupující menu, odkud jsou volány aktivity pro přehrávání hudby, videa, prohlížení obrázků a další. Přechod mezi jednotlivými aktivitami je znázorněn diagramem na obr. 4.5.



Obrázek 4.5: Aktivity aplikace iMM Tablet a přechody mezi nimi.

Vstupní aktivitou aplikace je `MainMenuActivity`. Ta je rozdělena do šesti kategorií, přičemž jedné kategorii odpovídá právě jeden řádek.

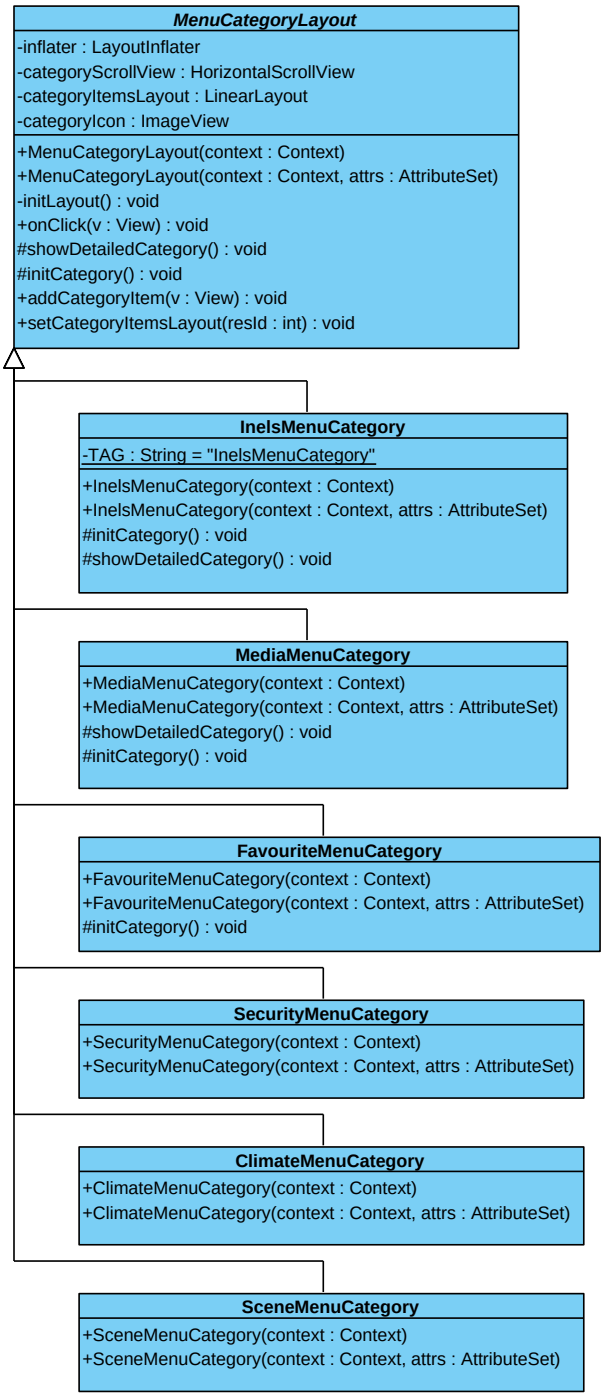
Diagram tříd jednotlivých kategorií znázorňuje obr. 4.6.

Z diagramu vyplývá, že každá třída reprezentující kategorii dědí od abstraktní třídy `MenuCategoryLayout`. Tato třída definuje způsob rozvržení grafických komponent v kategorii a poskytuje metody pro jejich dynamické přidávání (metoda `addCategoryItem()`), případně načtení statického obsahu kategorie z XML souboru (metoda `setCategoryItemsLayout()`).

Díky třídě `MenuCategoryLayout` je zaručen konzistentní vzhled a chování všech kategorií.

V případě, že bude požadováno přidání nové kategorie, stačí vytvořit novou třídu dědící od abstraktní třídy `MenuCategoryLayout` a deklarovat ji v XML souboru popisujícím grafické rozvržení aktivity.

Dynamického přidávání prvků je využito např. v první kategorii, která obsahuje grafické prvky pro ovládání CPU a jednotek systému iNELS. Každá elektroinstalace je unikátní a dynamicky se měnící, tudíž bylo nutné navrhnout způsob, který umožňuje import a aktualizaci jednotek tak, aby odpovídala aktuální elektroinstalaci.



Obrázek 4.6: Diagram tříd kategorií použitých v `MainMenuActivity`.

Byly zváženy dva možné přístupy, z nichž prvním byla možnost dynamicky přidávat a odstraňovat prvky přímo prostřednictvím aplikace iMM Tablet. Tento přístup je však pro rozsáhlou elektroinstalaci velmi nepohodlný. Druhým přístupem, který byl nakonec zvolen, je možnost uložení souboru popisujícího příslušné jednotky elektroinstalace přímo do souborového systému platformy Android. Aplikace se při spuštění pokusí tento soubor přečíst a následně dynamicky vytvořit grafické prvky a přiřadit jim požadované akce.

Následně bylo nutné zvolit vhodný formát souboru poskytujícího všechny potřebné informace – název jednotky a její zařazení do místnosti (zóny). Tomuto požadavku odpovídá soubor `rooms.cfg` (viz kapitola 2.3), který je navíc využíván systémem iMM a iMM Touch.

Jednotky podporují zápis a čtení hodnot typu `REAL`, `BOOL`, nebo `BYTE` (viz kapitola 2.2.1). Definice typu nebyla elementy `item` XML souboru `rooms.cfg` podporována, proto ji bylo nutné doplnit. Bez existence definice typu by mohlo vést ke čtení a zápisu nesprávných hodnot. Jiným způsobem by bylo např. doplnění sufixu `_bool`, `_real`, či `_byte` k názvu (`item_name`) každé jednotky. Tento přístup by však vedl ke zbytečné redundanci souboru *pub*, jehož formát je popsán v kapitole 2.2.1.

Zmíněný element může tedy např. vypadat následovně:

```
<item inels="Halogenova_zarovka_1" type="REAL">Halogen1</item>
```

Kategorie multimédií je statického charakteru a její obsah je tedy načten z XML souboru. Grafické prvky této kategorie spouští další aktivity sloužící k ovládání systému iMM.

V 2. iteraci byla také provedena refaktorizace zdrojového kódu aktivit pro práci s multimédii. Aktivity pro přehrávání hudby a videa (viz obr. 4.8 a 4.9) pracují se zónami a každá z nich obsahuje také prohlížeč souborů umístěných na vzdáleném úložisti. Proto bylo nutné refaktorizovat zdrojový kód tak, aby byla dodržena zásada znovupoužitelnosti a nedocházelo ke zbytečné duplikaci kódu. V návaznosti na tento poznatek byl zdrojový kód prohlížeče a komponenty pro výběr zón přesunut do abstraktních tříd `BrowseableZoneActivity`, `BrowserActivity` a `ZoneActivity`. Tyto abstraktní třídy umožňují dalším aktivitám jednoduše začlenit do svého uživatelského rozhraní prohlížeč vzdálených souborů či komponentu pro výběr zóny, kterou si uživatel přeje ovládat.

Abstraktní třída `BrowseableActivity` definuje veřejná rozhraní, jejichž prostřednictvím je informována aktivita, která je implementuje, o tom, jaký soubor byl vybrán, případně také o dostupnosti metadat pro tento soubor.

Abstraktní třídy `BrowseableZoneActivity` a `ZoneActivity` implementují funkcionalitu pro načtení dostupných zón ze serverové strany a jejich zobrazení v příslušné grafické komponentě umožňující mezi těmito zónami přecházet. Třídy a tedy i zároveň aktivity, které z nich dědí, mohou jednoduše získat vybranou zónu pomocí metody `getZone()`.

Obrázek 4.7: Ukázka hlavní aktivity aplikace iMM Tablet.

Obrázek 4.8: Aktivita `AudioActivity`.

Obrázek 4.9: Aktivita `VideoActivity`.

4.2 ITP Designer

ITP Designer je desktopová aplikace, která vznikla na základě požadavku na vytvoření jednoduchého nástroje, který by umožňoval konfigurovat aplikaci iMM Tablet podle aktuální inteligentní elektroinstalace. Styčným bodem několika částí systému iMM a iNELS je soubor `rooms.cfg` a samozřejmě odpovídající *pub* soubor, bez něhož by nebylo možné tyto systémy vůbec propojit.

Absence jednoduchého nástroje pro tvorbu a editaci souboru `rooms.cfg` vedla k vytvoření aplikace ITP Designer. Tento konfigurační soubor je ve formátu XML, ale přímá editace jeho zdrojového kódu je při rozsáhlé elektroinstalaci značně nepohodlná, a navíc neposkytovala uživatelsky přívětivý způsob modifikace tohoto souboru. Tento fakt znemožňoval koncovému uživateli provedení příslušných změn v konfiguraci.

V předchozí kapitole bylo diskutováno několik přístupů a platforem pro vytvoření desktopové aplikace a nakonec byla vyhodnocena platforma NetBeans jako nejvhodnější.

Díky kvalitní architektuře platformy NetBeans se návrh aplikace zabýval především rozvržením komponent GUI, volbou vhodných aplikačních rozhraní a nástroje pro správu a automatizaci sestavení aplikace, který je při následné implementaci a údržbě velmi důležitý.

4.2.1 Návrh GUI aplikace

Vzhledem k tomu, že na počátku existovala jen pouhá představa o vzhledu a funkci aplikace iMM Tablet, nebylo při analýze požadavků zcela jasné, jakým směrem by se měla aplikace ITP Designer ubírat.

Jediným podkladem bylo konkurenční proprietární řešení návrháře od společnosti iRidium Mobile Ltd. Vzhledem k jeho uzavřenosti posloužilo toto řešení pouze pro získání představy o možnosti využití nově vznikající aplikace ITP Designer.

Konečný návrh byl inspirován konceptem ovládání a rozvržení GUI známým z většiny vývojových prostředí. Základními stavebními prvky vývojových prostředí jsou projekty různých typů, pro které bývá typicky v levé části GUI aplikace umístěna komponenta umožňující procházení a modifikaci struktury projektu, včetně editace konfiguračních souborů a zdrojových kódů. Často jsou také poskytovány nástroje pro rapidní vývoj aplikací neboli *RAD*, mezi jejichž nejvýznamější zástupce patří zejména návrháři GUI.

Aplikace ITP Designer adaptuje koncept projektu tak, aby abstrahoval obsah souboru `rooms.cfg` a umožnil tak jeho jednoduchou modifikaci. Důraz byl kladen především na odstínění uživatele od potřeby modifikace zdrojového kódu tohoto souboru.

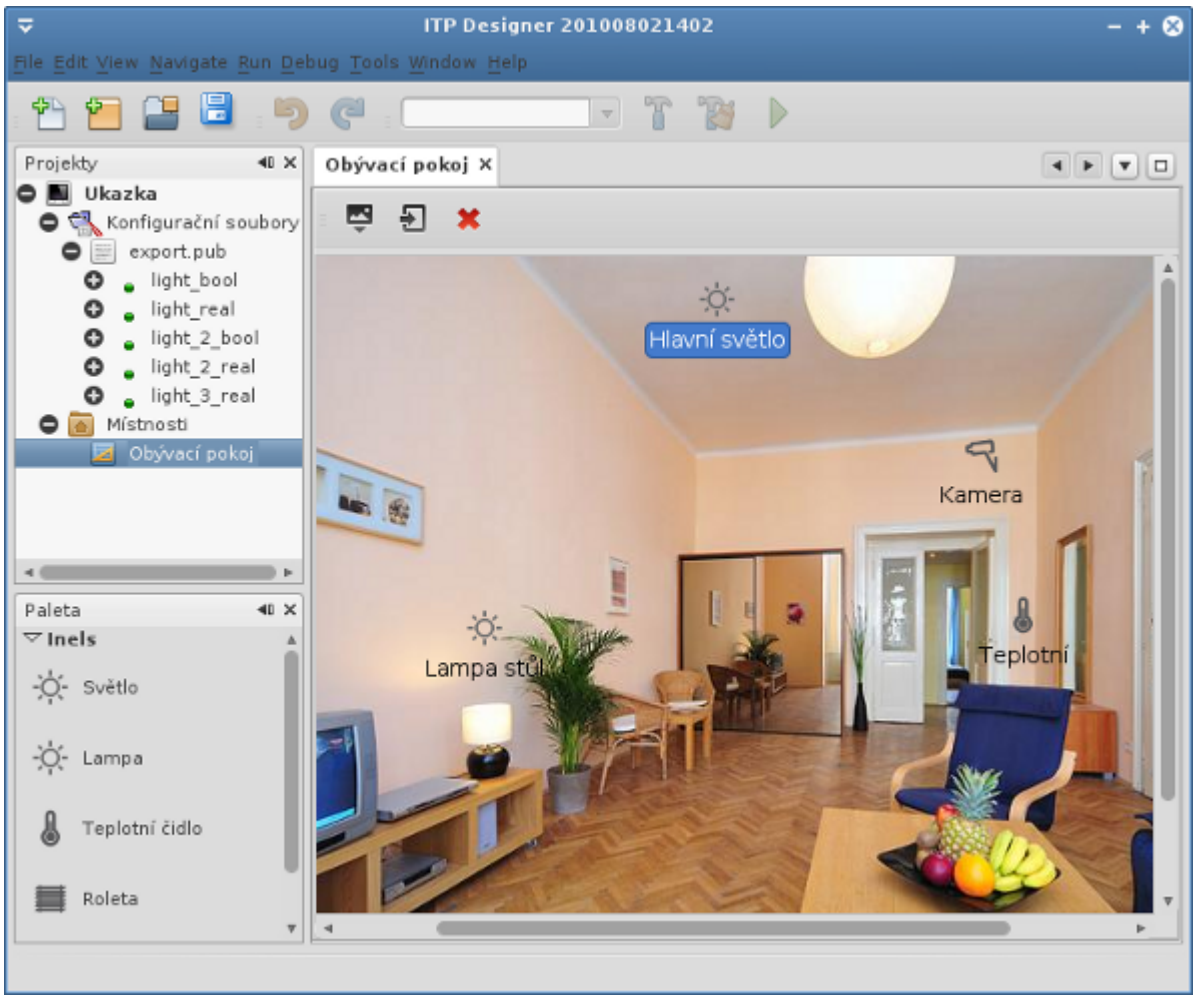
Analýza sémantiky dat souboru `rooms.cfg` vedla k vytvoření následujících akcí, které by měla aplikace ITP Designer uživateli poskytovat:

1. Přidávat a odstraňovat místnosti.
2. Přiřazovat prvky jako jsou např. světla, rolety, teplotní čidla atd. jednotlivým místnostem.
3. Svázat tyto prvky s příslušnými jednotkami systému iNELS.

Jako nejlepším možný způsob pro přiřazování prvků k jednotlivým místnostem se ukázal přístup *drag & drop*, v němž si uživatel vybírá požadované prvky z palety a umisťuje je na scénu reprezentující místnost.

Pro aplikaci iMM Tablet nemá pozice prvků na scéně v současné době význam. Ovšem pro budoucí rozšíření aplikace iMM Tablet či ITP Designer lze požadavek podporující tuto vlastnost předpokládat, a proto byl zahrnut již do první verze návrháře.

Ukázka rozrvžení GUI aplikace ITP Designer je uvedena na obr. 4.10.



Obrázek 4.10: GUI aplikace *ITP Designer*.

4.2.2 Sestavení aplikace

Pro správu a sestavení aplikace byl vybrán nástroj Apache Maven, který byl původně vytvořen, aby zjednodušil sestavování aplikací pro projekt Jakarta Turbine [4]. Hlavním impulzem pro jeho vznik byla snaha o standardizaci a opětovné využití sestavovacích skriptů, které v tehdy používaném nástroji Apache Ant nebylo plně podporováno [22].

Projekt je v Mavenu popsán pomocí modelu známého jako *Project Object Model*. Tento model popisuje softwarový projekt nejen z pohledu zdrojového kódu, ale i včetně jeho závislostí na externích knihovnách. Součástí modelu je i popis procesu sestavování a různých funkcí s tím spojených, jako je např. spouštění testů, sbírání informací o zdrojových kódech apod. [22].

Maven je postaven na modulární architektuře a funguje na principu volání jednotlivých zásuvných modulů [22].

Koncept modelu *Project Object Model* je založen na principu popisu projektu jako objektu. Za tímto účelem je definována jednoduchá XML struktura, která popisuje jednotlivé části projektu a jeho závislosti na externích knihovnách a nástrojích. Současně je možné

definovat konstanty, které pak mohou využít jednotlivé zásuvné moduly. Tento XML dokument se nachází v kořenovém adresáři projektu a je pojmenován `pom.xml`. Pokud je projekt složen z více dílčích projektů nebo modulů, každý z nich má pak svůj vlastní `pom.xml` soubor, který dědí vlastnosti od nadřazeného souboru a může přidávat další položky. Díky této struktuře je pak možné sestavit celý projekt jediným příkazem [22].

Při vývoji aplikací v jazyce Java se často řeší způsob, jakým spravovat a distribuovat jejich knihovny. Pro externí knihovny nabízí soubory `pom.xml` také možnost definovat vzdálené repozitáře knihoven ať již veřejné, či firemní. Závislosti na externí knihovny jsou jednoznačně definovány elementy `<groupId>` a `<artifactId>` neboli artefakty (artifacts), případně i požadovanou verzí dané knihovny. Maven se pak na základě této jednoznačné definice postará o vyhledání příslušných knihoven v repozitářích a jejich instalaci. Při vývoji tak odpadá nutnost distribuce archivů JAR.

Velkou výhodou nástroje Maven je také to, že umožňuje stáhnout příslušný *javadoc* a zdrojový kód knihovny, je-li dostupný. Při vývoji je tak dostupná potřebná dokumentace bez potřeby jejího manuálního stažení a instalace.

4.2.3 Popis implementace

Aplikace ITP Designer byla vytvořena ve vývojovém prostředí NetBeans IDE, která poskytuje kvalitní podporu pro tvorbu aplikací založených na platformě NetBeans.

Aplikace *ITP Designer* je rozdělena do pěti projektů:

- *ITPDesigner:Platform*
- *ITPDesigner:Platform - Application*
- *ITPDesigner:Platform - Branding*
- *ITPDesigner:Module - Core*
- *ITPDesigner:Module - Project Template*

Projekt *ITPDesigner:Platform* je pouze kontejnerem celé aplikace a obsahuje seznam modulů, včetně jejich umístění v repozitářích projektu.

Projekt *ITPDesigner:Platform - Application* specifikuje artefakty potřebné pro kompilaci aplikace. Požadované artefakty (moduly atp.) jsou specifikovány v souboru `pom.xml`.

Projekt *ITPDesigner:Platform - Branding* obsahuje zdroje, které se využívají k tzv. *brandingu* aplikace postavené na platformě NetBeans. Mezi *branding* patří také lokalizace a internacionalizace platformy NetBeans, která je standardně poskytována pouze v anglickém jazyce.

Projekt *ITPDesigner:Module - Core* (dále jen *Core*) obsahuje zdrojové kódy základního modulu aplikace.

Projekt *ITPDesigner:Module - Project Template* obsahuje šablonu a zdrojové kódy pro vytvoření nového projektu.

Každý projekt aplikace ITP Designer je svázán se soubory typu *pub* a `rooms.cfg`. Aby mohla platforma NetBeans s těmito soubory pracovat, bylo nutné nejdříve implementovat příslušné datové objekty (`DataObject`) a zaregistrovat je v souboru *layer* modulu *Core*. V návaznosti na datové objekty byly implementovány uzly reprezentující obsah těchto souborů v rámci GUI aplikace.

Během fáze návrhu aplikace byla pro projekt zvolena následující struktura adresářů a souborů:

```
|-- config
|   `-- export.pub
|-- itpproject
|   `-- project.properties
`-- rooms.cfg
```

Aby platforma výše uvedenou strukturu chápala jako projekt, bylo třeba nejdříve vytvořit třídu `ITPProjectFactory`, která implementuje rozhraní `ProjectFactory`, a `ITPProject` implementující rozhraní `Project`. Obě zmíněná rozhraní jsou součástí *Project API* platformy NetBeans.

Dalším krokem byla implementace třídy `RoomEditorTopComponent`, která slouží jako editor rozložení prvků v místnosti. Pro vizualizaci prvků a jejich rozmístění na plátně reprezentující danou místnost byla zvolena knihovna *Visual Library*, která je součástí platformy NetBeans od verze 6.0.

Mezi hlavní rysy knihovny *Visual Library* patří podpora přístupu *drag & drop* pro umísťování předdefinovaných grafických komponent (*widgets*) v rámci tzv. scény.

Scéna pro práci s komponentami zastupujícími prvky systému iNELS je implementována třídou `RoomEditorGraphSceneImpl`, dědící od generické abstraktní třídy `GraphScene`, která slouží pro vytváření a modifikaci grafově orientovaného modelu. Hrany spojující uzly grafu nejsou pro editor důležité, proto byla implementace abstraktních metod poskytujících funkcionalitu pro práci s hranami jednoduše vynechána.

Vytvoření instance třídy `RoomEditorGraphSceneImpl` zabezpečuje konstruktor třídy `RoomEditorTopComponent`. Scéna je do GUI editoru integrována prostřednictvím komponenty `JScrollPane`, která je součástí aplikačního rámce *Swing*. Fragment zdrojového kódu pro vytvoření instance scény a jejího začlenění do GUI je uveden ve výpisu 4.3.

Výpis 4.3: Ukázka začlenění scény do komponenty RoomEditorTopComponent.

```
private InstanceContent content;
private RoomEditorGraphSceneImpl scene;
private JScrollPane roomEditorScenePane;

public RoomEditorTopComponent() {
    initComponents();

    content = new InstanceContent();
    ProxyLookup lookup = new ProxyLookup(new AbstractLookup(content),
        Lookups.fixed(new Object[]{ItemsPaletteSupport.createPalette()}));
    associateLookup(lookup);

    // Create graph scene for design items in room
    scene = new RoomEditorGraphSceneImpl();
    JComponent view = scene.createView();
    roomEditorScenePane.setViewportView(view);
}
```

Kapitola 5

Nasazení aplikací

Tato kapitola se věnuje popisu distribuce a nasazení aplikací, které byly implementovány jako součást praktické části této práce.

5.1 ITP Designer

Pro distribuci aplikace ITP Designer byly vytvořeny spustitelné soubory s instalátory pro operační systémy Microsoft Windows a Linux. Distribuce a následná instalace je proto velmi snadná.

Po spuštění instalátoru se zobrazí pomocník, který provede uživatele celým instalačním procesem, a po úspěšném dokončení tohoto procesu nabídne možnost spuštění právě nainstalované aplikace.

5.2 iMM Tablet

Aplikace iMM Tablet je určena primárně pro mobilní zařízení Toshiba Folio 100. Jedná se o tablet s multitouch displejem o velikosti 10,1 palců, jehož rozlišení je 1024 x 600 bodů. Podporovanou verzí operačního systému Android je sestavení *Froyo*, tedy verze 2.2.

Pro distribuci aplikace byl zvolen standardní balíček APK (Android Package). Postup instalace na mobilní zařízení je následující:

- Připojení mobilního zařízení prostřednictvím USB rozhraní k počítači.
- Aktivace USB úložiště na tomto zařízení.
- Do libovolného umístění v úložišti zkopírujeme balíček s aplikací `immtablet-x.x.apk`.
- Nakonec provedeme instalaci aplikace prostřednictvím správce souborů dostupného v základní konfiguraci platformy Android.

Díky zvolenému formátu je možné aplikaci distribuovat také on-line prostřednictvím webové stránky nazývané *Android Market*. Standardní součástí platformy Android je stejnojmenná aplikace, která umožňuje stažení a instalaci nových aplikací.

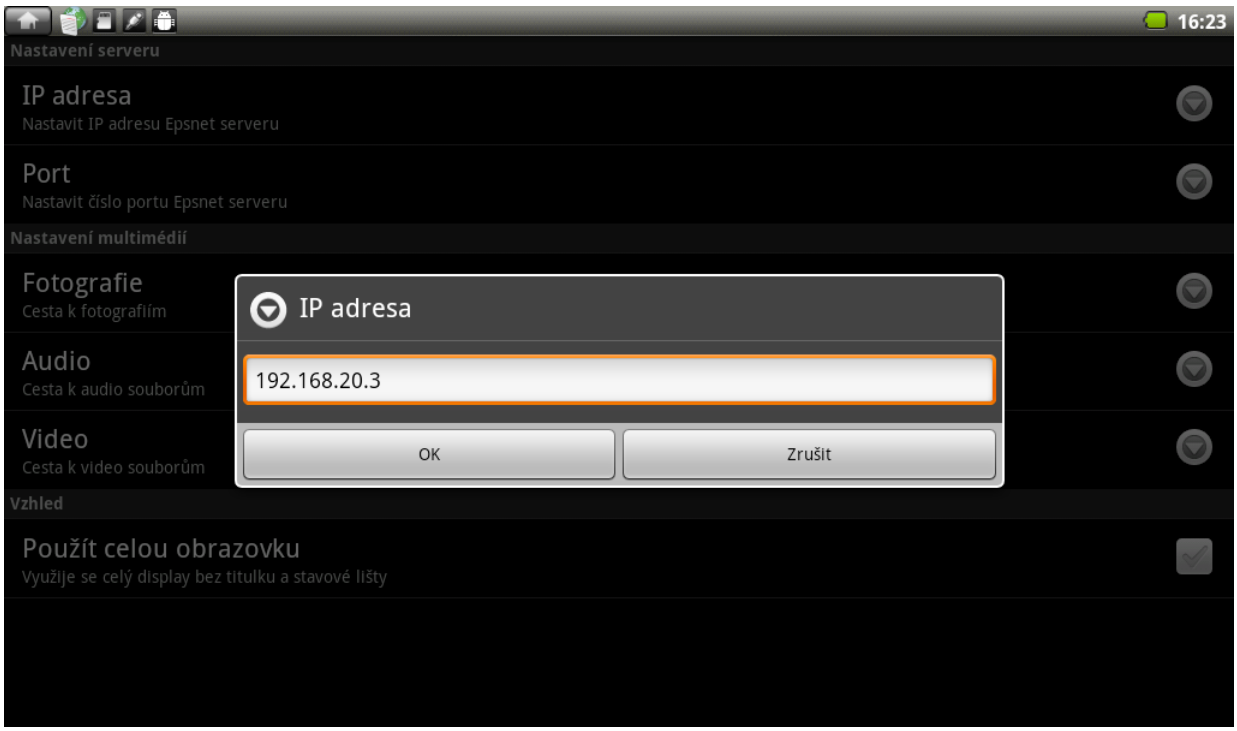
5.2.1 Konfigurace

Soubor `rooms.cfg` vytvořený aplikací ITP Designer je nutné zkopírovat do adresáře `/IMMTablet/conf/` úložiště mobilního zařízení. Pokud tyto adresáře neexistují, je nutné spustit aplikaci iMM Tablet, která zajistí jejich vytvoření. Je-li soubor `rooms.cfg` nalezen ve zmíněném adresáři, aplikace na základě jeho obsahu naplní příslušné kategorie ovladacími prvky.

Při prvním spuštění aplikace je třeba nastavit IP adresu a port, na kterém naslouchá server pro obsluhu XML-RPC požadavků. Důležitým nastavením je také zadání cest k adresářům na cílovém serveru, kde se nacházejí soubory s fotografiemi, hudebními soubory a video soubory.

Přístup k možnostem aplikace je umožněn z aktivty `MainMenuActivity`, která je spuštěna jako první při startu aplikace, stiskem talčítka Menu, které by mělo být součástí každého mobilního zařízení obsahujícího operační systém Android.

Vzhled aktivity s možnostmi nastavení aplikace je uveden na obr. 5.1.



Obrázek 5.1: Aktivita možností nastavení aplikace iMM Tablet.

Kapitola 6

Zhodnocení práce

Na základě analýzy požadavků byl vytvořen návrh řešení, na který navazovala samotná implementace tvořená aplikacemi iMM Tablet a ITP Designer.

Požadavek na implementaci komunikace aplikace iMM Tablet a stávajícího řešení nebyl do zadání původně zahrnut, protože neexistovala implementace serverové strany, která by umožnila ovládání systému.

Společnosti ELKO EP, s.r.o. byl navržen konkrétní způsob řešení problému komunikace, který byl akceptován. Vytvořením serverového řešení byl pověřen vývojář této firmy a samotná implementace vznikala také částečně díky vzájemné komunikaci, v rámci které byly diskutovány jednotlivé požadavky na ovládání stávajícího systému.

V návaznosti na poskytnutí serverového řešení třetí stranou (společností ELKO EP, s.r.o.) byl tedy navíc vytvořen návrh komunikace klientské strany (mobilního zařízení), který byl také implementován (detailní rozbor tohoto problému je součástí kapitoly 4.1.1).

V současné podobě umožňuje aplikace iMM Tablet ovládat prostřednictvím poskytnutého serverového řešení jednotku CPU systému iNELS a multimediální systém iMM.

Reálné ovládání bylo testováno na demonstračním kufru zapůjčeném společností ELKO EP, s.r.o., který obsahuje základní konfiguraci systému iMM, jednotku CPU a několik dalších jednotek vstupů a výstupů. Ke vstupně/výstupním jednotkám jsou připojeny spínače, halogenové žárovky, luminiscenční trubice, čidlo PIR a další.

Pomocí jednoduchého skriptu jazyka Python byla simulována situace, kdy je systém ovládán z více míst zároveň. Odezva na požadavky byla nepatrně delší, avšak nikterak zásadní. Situace ovládání velmi rozsáhlého řešení, kdy je v jeden okamžik přítomné velké množství požadavků, nemohla být za podmínek srovnatelných s laboratorním prostředím simulována. K těmto situacím by však nemělo docházet příliš často, ale pokud by přece jen bylo třeba obsluhovat velké množství požadavků, musela by být provedena také úprava serverové části tak, aby bylo zpracování těchto požadavků rozděleno do několika vláken.

Na základě požadavků byly pro běh aplikace ITP Designer zvoleny operační systémy z rodiny Microsoft Windows, konkrétně Microsoft Windows 7 a XP, avšak již během analýzy byl znám požadavek na možnost budoucího rozšíření o další operační systémy. Tento fakt bylo nutné vzít v úvahu již v samotném počátku návrhu aplikace. Jakmile by byla nově vznikající aplikace postavena na platformě závislých prostředcích, jako jsou např. knihovny, ovladače atd., byla by možnost jejího dalšího rozšíření značně omezena, případně zcela znemožněna.

Díky předešlým pozitivním zkušenostem s vývojem multiplatformních desktopových aplikací byl pro implementaci zvolen programovací jazyk Java (konkrétně Java SE) a na něm založená platforma NetBeans, jež využívá pro tvorbu GUI aplikační rámec *Swing*. Koncept

toho programovacího jazyka umožňuje běh aplikací na libovolném operačním systému či architektuře, pokud je pro ně dostupná implementace virtuálního stroje (JVM).

Základním stavebním prvkem platformy NetBeans je modul. Aplikace vystavěné nad touto platformou jsou již od počátku tvořeny tak, aby respektovaly modulární architekturu, která je u rozsáhlých aplikací samozřejmostí. Kvalitní návrh platformy NetBeans vede při vývoji k užívání obecně uznávaných praktik, mezi které jistě patří návrhové vzory, princip tzv. volných vazeb (*loose coupling*) a další.

Moduly poskytují prostředek pro přidávání nové funkcionality do aplikace tak, aby byly ovlivněny pouze části, které s touto funkcionalitou přímo souvisí. Pro moduly je snažší vytvářet automatizované testy, které se používají pro ověření předpokládaného chování metod.

Modulární architektura nabízí také kvalitní prostředí pro týmovou práci, v rámci které může být implementace souvisejících modulů přenechána určité skupině vývojářů.

Aplikace ITP Designer byla úspěšně otestována na operačních systémech Microsoft Windows 7 (32 a 64bitová verze) a XP, dále na distribucích Debian a Ubutnu operačního systému Linux (platformy i386 a amd64).

Pro operační systémy Microsoft Windows 7/XP a Linux byly vytvořeny spustitelné soubory obsahující průvodce instalačním procesem. Tyto instalační průvodce lze dodatečně vytvořit také pro operační systémy Mac OSX a Solaris.

Požadavek na jednoduchý způsob instalace a distribuce aplikace ITP Designer byl tedy splněn.

Kapitola 7

Rozšíření do budoucnosti

Tato kapitola popisuje možná vylepšení aplikací iMM Tablet a ITP Designer. Vzhledem k povaze této práce a poskytnutým časovým zdrojům byla provedena implementace funkčních prototypů. Při návrhu těchto aplikací byl kladen velký důraz především na možnost jejich budoucího rozšíření.

7.1 iMM Tablet

Uživatelské rozhraní aplikace by bylo možné vylepšit následujícím způsobem:

- Přidáním podpory pro tzv. *Nine-Patch* grafiku, kterou platforma Android využívá pro automatickou změnu velikosti bitmapy na pozadí příslušných grafických komponent (tlačítek, seznamů atd.). Tato grafika umožňuje dynamické přizpůsobení grafických komponent velikosti displeje.
- Vytvořením sady rozmístění grafických komponent v rámci jednotlivých aktivit pro mobilní zařízení s různými velikostmi displeje. Např. pro mobilní telefon a tablet lze mít dvě rozdílná rozložení uživatelského rozhraní pro stejnou aktivitu.
- Vytvořením dalších aktivit pro přehlednější vizualizaci kategorií s velkým počtem ovládacích prvků. Překrytím metody `showDetailedCategory()` v třídě implementující danou kategorii lze reagovat na událost vyvolanou dotekem na ikonu reprezentující typ kategorie.

Komunikaci lze vylepšit vytvořením kontroleru (blíže viz kapitola 4.1.1), umožňujícího přímé ovládání jednotky CPU.

7.2 ITP Designer

Rozšíření aplikace ITP Designer bude vyplývat především z požadavků uživatelů, kteří budou s touto aplikací pracovat. Vzhledem k poměrně krátkému vývojovému cyklu tvořeného jedinou iterací, nebylo možné zjistit od uživatelů této aplikace požadavky na další rozšíření, které by byly součástí následující iterace.

Lze také předpokládat rozšíření palety o nové typy prvků.

Aplikaci by bylo možné rozšířit o podporu autodetekce nově připojeného mobilního zařízení k USB sběrnici počítače. Po detekci této události by bylo uživateli nabídnuto, zda-li si přeje právě aktivní projekt exportovat do zařízení.

Práce se službami a ovladači rozdílných operačních systémů již neumožňuje plně transparentní multiplatformí přístup. Aplikaci by bylo nutné rozšířit o rozpoznání hostitelského operačního systému, na jehož základě by byl vybrán příslušný platformě závislý ovladač sběrnice USB.

Pro napojení aplikací na nativní knihovny a služby operačních systémů lze využít rozhraní JNI (Java Native Iterface) či JNA (Java Native Access).

JNI poskytuje rozhraní pro propojení Java aplikace s knihovnami a programy, které byly napsány v jiných jazycích – např. C/C++.

Nativní knihovna musí být zkompilována zvlášt pro jednotlivé architektury (x86, amd64 atp.) a jednu knihovnu tedy nemůžeme použít pro různé operační systémy zároveň.

JNA je nádstavbou pro JNI, která umožňuje přístup ke sdíleným nativním knihovnách pouze prostřednictvím kódu napsaném v jazyce Java, a není tedy vyžadována implementace nativního kódu tak, jak tomu je u JNI.

Dalším možným vylepšením je vytvoření nástroje pro vygenerování optimálního rozložení prvků v místnosti a poskytnutí několika základních přístupů pro generování, např. rozmístění prvků do mřížky atd.

Kapitola 8

Závěr

Na úvod byl čtenář této práce postupně seznámen s pojmy inteligentní elektroinstalace a inteligentní domov. Dále bylo představeno jejich konkrétní řešení od společnosti ELKO EP, s.r.o., jehož vlastnosti byly porovnány s open source projekty LinuxMCE a XBMC.

Cílem této práce bylo analyzovat stávající řešení společnosti ELKO EP, s.r.o. a navrhnout aplikaci pro mobilní zařízení (iMM Tablet), která by umožnila jeho ovládání. Dalším požadavkem bylo vytvoření desktopové aplikace (ITP Designer) jež by sloužila jako jednoduchý nástroj pro konfiguraci mobilní aplikace.

Analýze dostupných prostředků pro tvorbu požadovaných aplikací a jejich návrhu byla věnována významná část této práce. Již během rané fáze analýzy požadavků bylo odhaleno slabé místo stávajícího řešení, které bylo ukryto ve zvoleném komunikačním protokolu. Na základě tohoto poznatku byl navržen způsob řešení tohoto problému, který byl akceptován a následně také zahrnut do fáze návrhu.

Při návrhu a následné implementaci byl brán ohled na předpokládaný budoucí vývoj a rozšíření obou aplikací. Pro aplikaci ITP Designer byla proto zvolena modulární architektura platformy NetBeans. Díky tomu lze aplikaci jednoduše rozšířit například o další typy projektů. Výhodou aplikace ITP Designer je také možnost jejího multiplatformního nasazení.

Aplikace iMM Tablet je určena pro mobilní platformu Android, u které lze předpokládat její další budoucí vývoj. Vzhledem k navrženému způsobu komunikace je možné vytvořit aplikace i pro další mobilní platformy – např. iOS, Windows Mobile a další.

Implementované aplikace jsou funkčními prototypy, jejichž možná budoucí rozšíření jsou popsána v kapitole 7.

S ohledem na další rozšíření aplikací by bylo vhodné analyzovat zkušenosti uživatelů s jejich ovládáním a získané poznatky začlenit do budoucího vývoje.

Literatura

[1] Android Developers: Activities. [online], [cit. 2011-05-14].
URL
http://developer.android.com/guide/topics/fundamentals/activities.html

[2] Android Developers: Android Architecture. [online], [cit. 2011-01-02].
URL http://developer.android.com/guide/basics/what-is-android.html

[3] Android Developers: Application Fundamentals. [online], [cit. 2011-05-14].
URL http://developer.android.com/guide/topics/fundamentals.html

[4] Apache Maven Project: Introduction. [online], [cit. 2011-05-20].
URL http://maven.apache.org/what-is-maven.html

[5] Boeck, H.: *The Definitive Guide to NetBeans Platform (Books for Professionals by Professionals)*. Apress, 2009, ISBN 9781430224174.

[6] Edd Dumbill, Joe Johnston, Simon St. Laurent: *Programming Web Services with XML-RPC*. O'Reilly, 2001, ISBN 0-596-00119-3.

[7] J. Zendulka,V. Bartík,Š. Květoňová: *Studijní podpora k předmětu AIS : Analýza a návrh informačních systémů* [online].
https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/AIS-IT/texts/AIS.pdf,
2006-10-31 [cit. 2011-04-12].

[8] James Snell, Doug Tidwell, Pavel Kulchenko: *Programming Web Services with SOAP*. O'Reilly, 2001, ISBN 978-0-596-00095-0.

[9] Krejčí, L.: *NetBeans Platform jako EJB klient*. Bakalářská práce, Brno : Masarykova Univerzita, Fakulta Informatiky. 2009.

[10] Meier, R.: *Android^{TM} 2 Application Development*. Wiley Publishing, Inc., 2010, ISBN 978-0-470-56552-0.

[11] Minář, M.: *Vytvoření multimediálního ovládacího systému v Linuxu*. Bakalářská práce, Ústav inteligentních systémů FIT VUT v Brně. 2010.

[12] Oracle Corporation: A Brief History of NetBeans. [online], [cit. 2011-04-05].
URL http://netbeans.org/about/history.html

[13] Robert Harris, Rob Warner: *The Definitive Guide to SWT and JFace*. Apress, 2004, ISBN 1590593251.

[14] Stýskalík, J.: Příručka pro software Inels Designer a Manager. Technická zpráva, ELKO EP, s.r.o., 2008, 1. vydání, rev. číslo 2, říjen 2008.

[15] Sun Microsystems, Inc.: Extension Mechanism Architecture. [online], [cit. 2011-05-03]. URL http://download.oracle.com/javase/1.4.2/docs/guide/extensions/spec.html

[16] The LinuxMCE wiki: Automation. [online], [cit. 2011-01-02]. URL http://wiki.linuxmce.org/index.php/Home_Automation

[17] The LinuxMCE wiki: LinuxMCE. [online], [cit. 2011-01-02]. URL http://wiki.linuxmce.org/index.php/LinuxMCE

[18] Vogel, L.: Eclipse RCP Tutorial. [online], [cit. 2011-01-02]. URL http://www.vogella.de/articles/EclipseRCP/article.html

[19] Wikipedia: Rich Client Platform — Wikipedia, The Free Encyclopedia. 2011, [online], [cit. 2011-05-06]. URL http://en.wikipedia.org/w/index.php?title=Rich_Client_Platform&oldid=427789901

[20] Wikipedia: Software framework — Wikipedia, The Free Encyclopedia. 2011, [online], [cit. 2011-05-03]. URL http://en.wikipedia.org/w/index.php?title=Software_framework&oldid=429585284

[21] Wikipedie: Eclipse RCP. [online], [cit. 2011-01-02]. URL http://cs.wikipedia.org/wiki/Eclipse_RCP

[22] Wikipedie: Apache maven — Wikipedie: Otevřená encyklopedie. 2011, [online], [cit. 2011-05-20]. URL http://cs.wikipedia.org/w/index.php?title=Apache_maven&oldid=6671087

[23] Wikipedie: XML-RPC — Wikipedie: Otevřená encyklopedie. 2011, [online], [cit. 2011-05-04]. URL http://cs.wikipedia.org/w/index.php?title=XML-RPC&oldid=6857176

[24] XBMC Wikipedia: XBMC Features and Supported Formats/Codecs. [online], [cit. 2011-01-02]. URL http://wiki.xbmc.org/index.php?title=XBMC_Features_and_Supported_Formats/Codecs

Seznam použitých zkratek

API Application Programming Interface

AWT Abstract Window Toolkit

CVS Concurrent Version System

FTP File Transfer Protocol

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

iOS iPhone OS

iMM iNELS Multimedia

ITP iNELS Touch Panel

JNA Java Native Access

JNI Java Native Interface

JVM Java Virtual Machine

MVC Model View Controller

PIR Passive Infrared Sensor

PYRO Python Remote Object

RAD Rapid Application Development

RCP Rich Client Platform

RPC Remote Procedure Call

SWT Standard Widget Toolkit

SOAP Simple Object Access Protocol

UDP User Datagram Protocol

VM Virtual Machine

XML Extensible Markup Language

Seznam příloh

- Příloha A – Ukázka zdrojových kódů.
- Příloha B – Obsah přiloženého CD.

Příloha A

Ukázka zdrojových kódů

A.1 Třída zapouzdřující volání vzdálených procedur

```
public class XMLRPCMethod extends Thread {
  private String method;
  private Object[] params;
  private Handler handler;
  private Handler exHandler;
  private RPCMethodCallBack callBack;
  private XMLRPCClient client;

  private static final String TAG = "RPCMethod";

  /**
   * Method for handling XML RPC Method calling with callback
   *
   * @param method
   * @param client
   * @param handler
   * @param callBack
   */
  public XMLRPCMethod(String method, XMLRPCClient client, Handler handler,
      RPCMethodCallBack callBack) {
    this.method = method;
    this.callBack = callBack;
    this.handler = new Handler();
    this.client = client;
    this.exHandler = handler;
  }

  /**
   * Method for handling XML RPC Method calling without callback
   *
   * @param method
   * @param client
   * @param handler
   * @param callBack
   */
  public XMLRPCMethod(String method, XMLRPCClient client, Handler handler) {
    this.method = method;
    this.handler = new Handler();
    this.client = client;
    this.exHandler = handler;
```

```
  }

  public void call() {
    call(null);
  }

  public void call(Object params[]) {
    this.params = params;
    start();

  }

  @Override
  public void run() {
    try {
      final Object result = client.callEx(method, params);
      handler.post(new Runnable() {

        public void run() {
          if (callBack != null) {
            callBack.callFinished(result);
          }
        }
      });
    } catch (final XMLRPCFault e) {
      handler.post(new Runnable() {

        public void run() {
          Message faultMessage = Message.obtain();
          faultMessage.setTarget(exHandler);
          faultMessage.what = MessagesConstants.RPC_METHOD_FAULT;
          faultMessage.sendToTarget();
          Log.d(TAG, "error", e);
        }
      });
    } catch (final XMLRPCException e) {
      Message connectionRefusedMessage = Message.obtain();
      connectionRefusedMessage.setTarget(exHandler);
      connectionRefusedMessage.what = MessagesConstants.CONNECTION_REFUSED;

      if (e.getCause() instanceof HttpHostConnectException) {
        Log.e(TAG, "Can not connect to the server");
        connectionRefusedMessage.sendToTarget();

      } else if (e.getCause() instanceof SocketException) {
        Message message = Message.obtain();
        message.setTarget(exHandler);
        message.what = MessagesConstants.NO_ROUTE_TO_HOST;
        message.sendToTarget();
      }
    }
  }
}
```

Příloha B

Obsah přiloženého CD

Struktura složek a popis jejich obsahu:

```
.
|-- demo
|   |-- Ukazka
|   `-- itpdesigner-demo.ogv
|-- dip
|   |-- dip-xkucer51.pdf
|   `-- dip-xkucer51.tar.gz
|-- dist
|   |-- immtablet-11.05.apk
|   |-- itp-designer-windows.exe
|   `-- itp-designer-1.0-linux.sh
`-- src
    |-- immtablet.tar.gz
    `-- ITPDesigner.tar.gz
```

- `demo` – Složka obsahuje ukázkový projekt a video demonstrující práci s aplikací ITP Designer.
- `dip` – Složka obsahuje technickou zprávu ve formátu pdf a archiv s jejími zdrojovými kódy ve formátu LaTeX.
- `dist` – Složka obsahuje balíček s aplikací iMM Tablet pro operační systém Android a instalátory aplikace ITP Designer pro operační systémy Microsoft Windows a Linux.
- `src` – Složka s archivy zdrojových kódů aplikací iMM Tablet a ITP Designer, včetně všech závislostí nutných k jejich sestavení.