**UNIVERZITA PARDUBICE**

**FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

# DIPLOMOVÁ PRÁCE

2012 Bc. Vojtěch BIBERLE

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

# Ovládání linuxového OS ze systému Android

Bc. Vojtěch Biberle

Diplomová práce

2012

**Univerzita Pardubice**
**Fakulta elektrotechniky a informatiky**
Akademický rok: **2011/2012**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Vojtěch Biberle** |
| Osobní číslo: | **I09352** |
| Studijní program: | **N2646 Informační technologie** |
| Studijní obor: | **Informační technologie** |
| Název tématu: | **Ovládání linuxového OS ze systému Android** |
| Zadávající katedra: | **Katedra softwarových technologií** |

Zásady pro vypracování:

Teoretická část:
- Bude provedena analýza možností ovládání grafických aplikací Linuxu vzdáleně (tj. bez přístupu ke grafickému rozhraní např. pomocí VNC apod.).
- Bude navržen vhodný bezpečný způsob ovládání těchto aplikací vzdáleně ze systému Android.

Praktická část:
- Bude implementován vhodný způsob ovládání aplikací Linuxu ze systému Android včetně jejich vzdáleného spouštění a ukončování.
- Bude-li to možné, bude aplikace implementována tak, aby nevyžadovala implementaci speciálního SW na systému Linux.
- Bude řešen i bezpečný přístup.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

**The Developer's Guide [online]. Google, 2010 [cit. 2010-10-12]. URL: http://developer.android.com/guide/.**
**Google Projects for Android [online]. Google, 2010 [cit. 2010-10-12]. URL: http://code.google.com/android/.**
**Meier, Reto. Professional Android 2 Application Development (Wrox Programmer to Programmer). Indianapolis: Wiley Publishing, 2010. ISBN: 978-0-470-56552-0.**

Vedoucí diplomové práce: **Mgr. Tomáš Hudec**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2011**
Termín odevzdání diplomové práce: **18. května 2012**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2011

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 31. 8. 2012

Bc. Vojtěch Biberle

**Poděkování**

Zde děkuji panu Mgr. Tomáši Hudcovi, že mi umožnil zpracovat diplomovou práci, ve které jsem vytvořil aplikaci pro Android, která přinese užitek mě, a doufám, že i jiným lidem. Tato diplomová práce mne obohatila o zkušenosti se systematickým návrhem aplikace pro Android, které určitě vyžiji ve svém profesním životě.

## ANOTACE

Návrh a implementace řešení pro ovládání OS linuxového typu pomocí systému Android.



## KLÍČOVÁ SLOVA

Vzdálená správa, Linux, Android.


## TITLE

Remote control of the Linux OS from Android


## ANNOTATION

Design and implementation solution for remote controling of Linux type OS from Android.



## KEYWORDS

Remote controlling, Linux, Android

# Obsah

# Úvod

Rozmach chytrých telefonů přinesl i vyšší nároky na telefony samotné. Už se nepoužívají jen k pouhému telefonování a psaní SMS, ale zastávají funkci minipočítačů. Nároky na tyto minipočítače jsou velké a stále se zvětšují. Dnes chtějí uživatelé od svého telefonu možnost připojit se k internetu a prohlížet si libovolnou stránku na internetu, vyfotografovat zážitky z dovolené, poslouchat hudbu, komunikovat s přáteli prostřednictvím IM a emailu, připojit se k různým geolokačním hrám, jako je GeoCaching nebo Foursquare ale třeba i řešit složitější matematické úkoly, vytvořit tabulky v tabulkovém editoru nebo si prohlédnout dokumenty vytvořené v MS Office.

Dalším druhem přenosných zařízení, která momentálně zažívají úspěch, jsou tablety. Opět jsou používány převážně k zábavě, jako výše zmiňované telefony, ale hodí se už i například na sledování filmů a hlavně na složitější pracovní úkoly, protože mají větší obrazovku, na které se dá pracovat mnohem lépe, než na malých obrazovkách telefonů. Většina tabletů je také hardwarově výkonnější než telefony, proto je na nich možné pouštět náročnější aplikace nebo provozovat stejné aplikace rychleji, než na telefonu.

Jednou z možností, jak využívat chytré telefony je vzdálené ovládání počítačů. Už nyní existuje několik programů, které jsou schopny ve větší či menší míře počítač ovládat. Některé se zaměřují na kompletní ovládání počítače, jiné jen na jeden konkrétní program. Většina těchto programů má společnou vlastnost, kterou je nutnost instalovat na ovládaný počítač serverovou část, která zprostředkuje programu možnost, ovládat jednotlivé aplikace. Existuje sice několik aplikací, které nepotřebují server, ale tyto programy jsou nedokonalé, často chybují při přerušení spojení s počítačem a spouští aplikaci znovu a nejsou jednoduše rozšiřitelné o další vzdáleně ovládané programy.

Cílem práce je navrhnout aplikaci pro operační systém Android, která nebude potřebovat ke své činnosti speciální serverovou část. Bude zaměřena na ovládání Linuxového operačního systému a k ovládání bude využívat SSH, který je standardní součástí všech distribucí Linuxu. Důraz bude kladen na bezpečné přihlašování k systému, které bude využívat šifrování v SSH i možnost přihlásit se pomocí SSH klíče. Návrh systém modulů, které umožní jednoduché rozšíření aplikace o možnost ovládat další programy na počítači, tak aby si zkušenější uživatelé mohli sami přesně nadefinovat, jaké činnosti má aplikace s počítačem provádět. Vznikne i několik modulů, které umožní používání aplikace i uživatelům, kteří nechtějí hlouběji zasahovat do připraveného rozhraní aplikace a chtějí aplikaci jednoduše používat. Vznikne uživatelsky příjemné rozhraní,

které bude umožňovat jednoduché přidávání modulů, správu seznamu ovládaných počítačů i ovládání samotných počítačů. Rozhraní bude obsahovat předdefinovaný vzhled pro multimediální aplikace a možnost, vytvořit si vlastní rozhraní pro libovolnou aplikaci.

# 1 Dostupné aplikace pro Android

Momentálně existuje více možností, jak se připojit k počítači a ovládat veškerou jeho činnost. Tyto aplikace ale nesplňují jednu nebo i více podmínek, které jsou stanoveny pro výslednou aplikaci. Jednotlivé aplikace můžeme zařadit do třech hlavních skupin:

- Ovládají celý počítač prostřednictvím přenosu obrazovky, napodobují myš a klávesnici.
- Ovládají skupinu aplikací, pro které byly navrženy a obsahují specifické příkazy pro každou aplikaci, bez možnosti rozšíření.
- Ovládají jen jednu aplikaci, obsahují pro ni specifické příkazy a snaží se zpřístupnit ovládání většího množství funkcí, než dříve uvedené. Jsou bez možnosti rozšíření.

Následují jednotlivé příklady aplikací, pomocí kterých se dá ovládat počítač a popis funkčnosti aplikací, jejich výhody a nevýhody.

## 1.1 TeamViewer

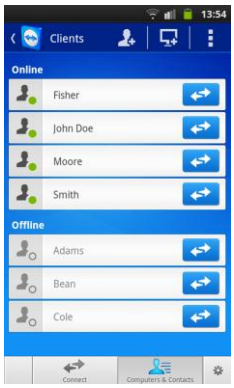
Je aplikace určená k celkovému ovládání PC a to prostřednictvím přenosu obrazu na ovládací zařízení a napodobováním klávesnice a myši. Původně se jednalo o čistě desktopovou aplikaci, která běžela na Windows. Následně se rozšířila i na další operační systémy a v poslední řadě i na Android. Výhodou této aplikace je, že propojení probíhá přes servery společnosti TeamViewer GmbH, která TeamViewer vyvíjí. Je tedy možné se připojit i na zařízení, která jsou navzájem schovaná za NAT.

Tabulka 1: TeamViewer - výhody a nevýhody

| Výhody | Nevýhody |
|---|---|
| • Ovládání celého PC<br>• Překonání NAT<br>• Zdarma pro osobní účely<br>• Multiplatformní (neumožnuje ovládat některé OS)<br>• Přenos souborů<br>• Vytvoření VPN | • Nepřehledný na malých obrazovkách, celé PC se ovládá hůře než se specializovanou aplikací<br>• Nemožnost ovládat Linuxové OS<br>• Absence modulárnosti a rozšiřitelnosti<br>• Datová náročnost<br>• Potřebuje server<br>• Proprietární server na cizím PC |

TeamViewer je užitečný program, který se snaží přenést na Android principy z desktopových počítačů. Na Androidu se dá použít spíše jako neplnohodnotná náhrada ovládání PC. Jeho velkou výhodou je jednoduché překonání NAT a možnost se spojit s jakýmkoliv počítačem. Dalším plusem je možnost registrace. Registrovaný uživatel se může na jakémkoliv zařízení, které obsahuje TeamViewer přihlásit a okamžitě uvidí všechny své počítače, které má ke svému účtu přidané. Nemusí si tedy pamatovat všechna čísla počítačů a hesla, stačí jen přihlašovací údaje pro TeamViewer. Nevýhodou je (na Androidu) právě jeho nespecializovanost. Na malém displeji se celé PC ovládá špatně a některé operace jsou velmi těžce proveditelné, málem až nemožné. TeamViewer se tuto slabinu snaží minimalizovat vhodným zvětšením či zmenšením přenášeného obrazu a zobrazováním pouze výřezů obrazovky ovládaného PC. Zdrojem obrázků web společnosti TeamViewer GmbH: www.teamviewer.com/cs/download/mobile.aspx.


Obrázek 1: TeamViewer - úvod




Obrázek 2: TeamViewer - partneři

Obrázek 3: TeamViewer - Windows

Obrázek 4: TeamViewer - Mac OS X

## 1.2 RealVNC

VNC klient, známý i z desktopových počítačů. Pracuje s VNC protokolem, který je určený pro přenos celé obrazovky. Proto je RealVNC také nevhodné pro používání na mobilních zařízeních, kde malá obrazovka neumožňuje vidět celou obrazovku PC. RealVNC opět kompenzuje tyto nevýhody nabídkou zmenšení obrazovky na velikost obrazovky telefonu, případně zobrazením pouze výřezu obrazovky, ale pro reálnou práci s počítačem toto nestačí.

Tabulka 2: RealVNC - výhody a nevýhody

| Výhody | Nevýhody |
|---|---|
| • Ovládání celého PC<br>• Multiplatformní protokol VNC | • Placené (existují i neplacené VNC prohlížeče pro Android)<br>• Nepřehledné na malých obrazovkách<br>• Absence modulárnosti a rozšiřitelnosti<br>• Datová náročnost<br>• Potřebuje server |

I když je RealVNC pokročilý VNC klient, nemůže se vypořádat VNC protokolem jinak, než pouze výřezy obrazovky, případně jejím zmenšením. Neposkytuje tedy snadné ovládání PC. Také se musí na ovládaném počítači nainstalovat VNC server a jeho instalace není vždy jednoduchá. Bezplatní VNC klienti pro Android přináší nanejvýš stejné funkce, jako komerční RealVNC a v mnoha případech tolik funkcí ani nenabízí. Obrázky jsou získány z webu společnosti, která aplikaci vyvíjí: RealVNC: www.realvnc.com/products/android/.


Obrázek 5: RealVNC - Windows


Obrázek 6: RealVNC - seznam

Obrázek 7: RealVNC - Windows XP

Obrázek 8: RealVNC - Windows XP 2

Obrázek 9: RealVNC - Windows XP 3

Obrázek 10: RealVNC - Mac OS X

## 1.3 RemoteDroid

Tato aplikace zvolila ojedinělý přístup k ovládání PC. Její autoři z telefonu udělali bezdrátový touchpad a klávesnici v jednom. Jen je potřeba mezi jednotlivými zařízeními přepínat. Nevýhodou je menší přesnost ukazatele myši a malá klávesnice. Uživatel také potřebuje stále dobře vidět, co na PC dělá. Proto je aplikace použitelná jen jako náhražka bezdrátové myši a klávesnice. RemoteDroid ke svému běhu také potřebuje server, prostřednictvím kterého komunikuje s ovládaným počítačem. Tento server je napsaný v Javě a tudíž je multiplatformní.

Tabulka 3: RemoteDroid - výhody a nevýhody

| Výhody | Nevýhody |
|---|---|
| • Základní verze zdarma<br>• Multiplatformní | • Zastupuje bezdrátovou myš a klávesnici<br>• Je potřeba server<br>• Malá přesnost<br>• Nutnost dobře vidět na ovládané PC<br>• Absence modulárnosti a rozšiřitelnosti |

Aplikace RemoteDroid se jistě vydala zajímavou cestou, ale dnes je pro stejnou a pohodlnější funkčnost lepší koupit plnohodnotnou bezdrátovou myš a klávesnici, které nabídnou větší výdrž, větší dosah a hlavně lepší pohodlí uživatele.

## 1.4 GMote

V této aplikací můžeme ovládat program VLC Player, který je multiplatformní. GMote ke komunikaci s VLC používá svůj vlastní server, napsaný v Javě. To zajišťuje GMote multiplatformnost. Zároveň tvůrci GMote tvrdí, že je možné ovládat více multimediálních přehrávačů, ale v současné době podporují pouze VLC a experimentálně Windows Media Player. GMote tedy umožňuje své funkce rozšířit o další přehrávače, ale je nutný zásah do zdrojových kódů serveru přímo od samotných tvůrců aplikace. Nesplňuje tedy požadavky na rozšiřitelnost a modulárnost.

Tabulka 4: GMote - výhody a nevýhody

| Výhody | Nevýhody |
|---|---|
| • Plná funkcionalita zdarma<br>• Multiplatformnost<br>• Jednoduchá instalace serveru<br>• Open Source<br>• Automatické vyhledání serveru v síti | • Je potřeba server<br>• Zatím pouze pro VLC Player<br>• Absence modulárnosti a rozšiřitelnosti |

GMote je pokročilou aplikací na ovládání multimediálních programů na PC. Nevýhodou je nutnost, čekat na tvůrce aplikace, až přidají další multimediální programy, které bude moci GMote ovládat. Přidání dalšího rozhraní, případně dokončení podpory Windows Media Playeru je ale v nedohledu, protože aplikace se přestala aktivně vyvíjet v roce 2011. Zdrojem obrázků jsou stránky GMote: http://www.gmote.org/screenshots.


Obrázek 11: GMote - hledání serveru


Obrázek 12: GMote - přidání serveru


Obrázek 13: GMote - ovládací obrazovka


Obrázek 14: GMote - výběr hudby

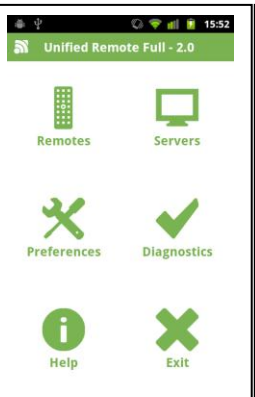
## 1.5 Unified Remote

Jedna z nejlepších aplikací pro ovládání PC pro Android. Bohužel i zde je potřeba serverová část nainstalovaná na PC a vývojáři podporují pouze operační systém Windows. Unified Remote podporuje simulaci bezdrátové klávesnice a myši a dalších 13 různých metod ovládání PC ve verzi zdarma. Jsou to ovládání napájeni, VLC Player i Windows Media Player, Youtube a Task Manager. Ve své placené verzi přidává dalších 33 metod ovládání, možnost nastavit si vlastní příkazy a hlasové příkazy. Možnost ovládat PC přes telefon hlasem je ojedinělá.

Tabulka 5: Unified Remote - výhody a nevýhody

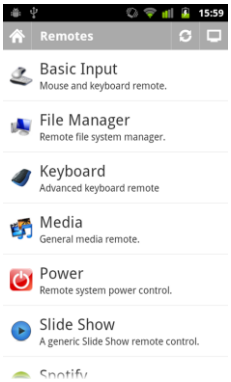
| Výhody | Nevýhody |
|---|---|
| • Celkem 24 způsobů ovládání PC v placené verzi<br>• Ovládání hlasem<br>• Vyhledávání serverů v síti<br>• Podpora BlueTooth<br>• 256 bitové AES šifrování<br>• Podpora pro WOL<br>• Rozšiřitelnost o vlastní příkazy (pouze placená verze) | • Pouze pro Windows<br>• Nutnost instalovat server (ale jednoduchá instalace) |

Unified Remote je skvělým příkladem, jak má vypadat aplikace na ovládání PC pro Android. Bohužel vývojáři podporují pouze platformu Windows. Pokud by byla aplikace multiplatformní, nevadila by ani nutnost instalovat speciální server pro podporu ovládání. Zdrojem obrázků je web společnosti Unified Intents AB: www.unifiedremote.com/screenshots.

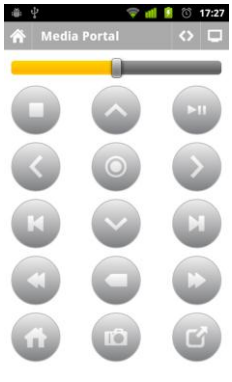



Obrázek 15: Unified Remote - hlavní obrazovka


Obrázek 16: Unified Remote - přidání serveru




Obrázek 17: Unified Remote - ovldače




Obrázek 18: Unified Remote - mediaportál

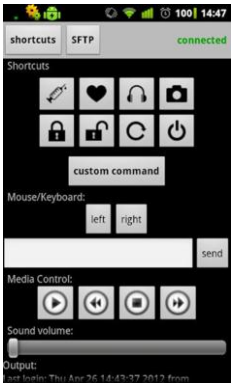
## 1.6 Ubuntu Remote Control

Jak už název napovídá, takto aplikace je určena pro ovládání distribuce Ubuntu a jejich odnoží. Ale je s ní možné ovládat i ostatní distribuce, protože k posílání příkazů používá program xdotools. Umožňuje se přihlásit pomocí SSH s heslem i s klíčem. Dále umí poslat jakýkoliv příkaz, bohužel jej ale neumí uložit pro pozdější použití. Její výhodou je implementace jednoduchého sftp klienta, pro stahování souborů z počítače do mobilu.

Tabulka 6: Ubuntu Remote Control - výhody a nevýhody

| Výhody | Nevýhody |
|---|---|
| • Potřebuje jen SSH server<br>• Podporuje i SSH klíče<br>• Umí SFTP<br>• Jednoduché uživatelské rozhraní | • Nelze ukládat uživatelsky definované příkazy<br>• Nelze definovat celé sety příkazů pro jednotlivé programy na PC |

Ubuntu Remote Control je uživatelsky velmi jednoduchá aplikace, která dovoluje jen základní práci se systémem, kterou připravil tvůrce aplikace. Neumožňuje definovat další příkazy, sekvence příkazů a tyto příkazy ukládat pro další použití. Je tedy vhodná na použití příkazů, které obsahuje, ale nehodí se pro ovládání více aplikací na jednom PC nebo stejné aplikace na více PC. Zdrojem obrázků jsou stránky Ubuntu Remote Control: http://pehux.bplaced.net/index.php#app-ubunturemotecontrol.




Obrázek 19: UCR - hlavní obrazovka


Obrázek 20: UCR - SFTP

## 1.7 Coversal

Nejpokročilejší aplikací, kterou lze momentálně nalézt a používat pro ovládání PC pomocí SSH. Coversal přichází s vlastní správou pluginů, které dovolují programu ovládat více programů na PC. Nabízí také přímo v aplikaci možnost definovat vlastní sety příkazů a uložit je pro pozdější použití. Umožňuje uložené sety exportovat do XML a sdílet je s ostatními a samozřejmě importovat sety z XML. Bohužel je Coversal navržen s takovou flexibilitou, že se jeho pokročilejší konfigurace stává nepřehlednou, umožňuje také definovat jakékoliv spustitelné příkazy, ale pouze na předdefinovaných tlačítcích grafického layoutu. Může se tedy stát, že budete nuceni definovat akci neodpovídající ikoně tlačítka. Dále Coversal přichází s vlastním repositářem pluginů, který dovolí přidání nového pluginu pouze jeho kompletním naprogramováním v Android SDK.

Tabulka 7: Coversal - výhody a nevýhody

| Výhody | Nevýhody |
|---|---|
| • Potřebuje jen SSH server<br>• Podporuje i SSH klíče<br>• Umožňuje ukládat uživatelské příkazy<br>• Umožňuje exportovat a importovat XML s uživatelskými příkazy<br>• Obsahuje repositář pluginů<br>• Umožňuje spouštět sekvence příkazů | • Nepřehledná a složitá konfigurace uživatelských příkazů<br>• Nemožnost otestovat příkazy v telefonu |

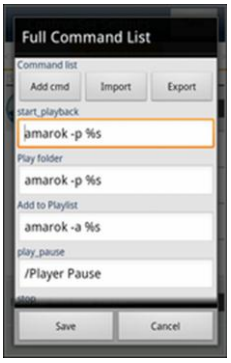
Kromě místy složitého uživatelského interface je Coversal ideálním programem pro ovládání PC po SSH. Přidávání uživatelských příkazů v telefonu je opravdu neintuitivní a pevně dané ikony tlačítek nutí k zapamatování si, jaká ikona jakou akci představuje. Malou záchranou je

možnost definovat celé sety příkazů pomocí XML, které je naštěstí dostatečně jednoduché. Zdrojem obrázků je Wiki stránka projektu Coversal:

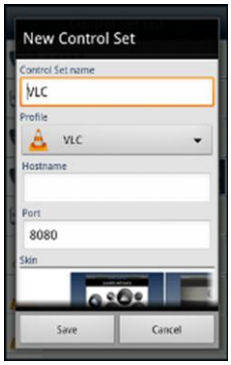
https://developers.coversal.com/projects/coversal/wiki.


Obrázek 21: Coversal - ovládací obrazovka


Obrázek 22: Coversal - plugin manager




Obrázek 23: Coversal - uživatelské příkazy




Obrázek 24: Coversal - nový set příkazů

## 1.8 Zhodnocení představených aplikací

Zde představené aplikace jsou výběrem nejlépe hodnocených aplikací v obchodu Google Play. Nesplňují však všechny podmínky pro ideální aplikaci pro vzdálení ovládání počítače pomocí SSH. Většina jich vyžaduje server a velká část jich je pouze pro operační systém Windows. Pokud jsou aplikace multiplatformní, nemají všechny požadované funkce, nejsou dostatečně intuitivní, případně požadují více či méně složitou instalaci serveru.

Jsou zde představeny i 2 aplikace, které využívají jako server pouze SSH. Dále lze najít aplikaci SSHMote, která už se nevyvíjí a jejím nástupcem je Coversal, který převzal veškerou funkcionalitu a ještě ji vylepšil. Coversal bohužel trpí nepřehledným ovládáním, hlavně v oblasti definice nových uživatelských příkazů. Na trhu momentálně není více aplikací, které by k ovládání využívaly jen SSH server.

# 2 Možnosti ovládání grafických aplikací linuxového OS pomocí SSH

Jelikož jsou linuxové distribuce značně roztříštěné prostředí, neexistuje jedna jediná možnost, jak posílat grafickým programům příkazy z příkazové řádky (tedy i vzdáleně pomocí SSH). Jsou tu dvě hlavní prostředí, která implementují systém posílání-poslouchání zpráv. První je Gnome se svým dbus a druhé je KDE s dcop. Naštěstí se vývojáři hlavních desktopových prostředí (tedy Gnome a KDE) shodli na používání jediného systému pro zasílání zpráv a to na dbus. Nové KDE4 tedy využívá také jen systém dbus. Proto se zde systémem dcop nebudu více zabývat. V případě zájmu si kdokoliv může definovat velice jednoduše svůj vlastní plugin, který bude využívat právě dcop.

Většina aplikací nabízí také možnost, zavolat aplikaci s parametrem dané akce, kterou má provést. Toto rozhraní často nabízí i aplikace, které podporují i dbus. Rozhraní příkazové řádky je většinou podstatně jednodušší, než ovládání pomocí systémových zpráv, ale často nabízí i méně možností, jak aplikaci ovládat. Já zde budu preferovat právě ovládání pomocí příkazové řádky, protože chci vytvořit jednoduché ukázky ovládání. Pokud bych rovnou využíval systému pro zasílání zpráv, pravděpodobně bych potencionální vývojáře pluginů odradil, protože příkazový interface k dbus je složitý a vyžaduje znalost mnoha parametrů. U každé aplikace jiných. Uživatelé, kteří se vyznají v dbus nebudou mít problém napsat pluginy pro aplikaci i za použití dbus.

Poslední možností je využít služby programu xdotool, který umožňuje vyhledat okno konkrétní aplikace (např. pomocí titulku, třídy okna) a poslat mu stisknutelný znak nebo kombinaci znaků, tak jako by ji program obdržel přímo od xserveru. Pomocí programu xdotool můžeme celému systému zasílat i multimediální klávesy. Systém se bude chovat stejně, jak by tyto klávesy byly zmáčknuty přímo na klávesnici, tedy tak, jak si nastavil uživatel.

Vzhledem k této rozmanitosti prostředí je nelepší cestou navrhnout systém pluginů, který umožní uživatelům definovat plugin pro jednotlivé aplikace, které chtějí na počítači ovládat. Tyto pluginy budou reprezentovány XML soubory s linuxovými příkazy, které se budou spouštět při zapnutí / vypnutí aplikace, případně při kliknutí na dané tlačítko v aplikaci pro Android. Toto řešení nabízí programátorovi, jednoduchou správu pluginů a uživatelům možnost jednoduše si napsat vlastní pluginy a přidat je do aplikace, případně pluginy sdílet s ostatními uživateli.

# 3 Návrh aplikace

V následujících odstavcích si představíme návrh aplikace, která splňuje požadavky na bezpečné přihlašování k systému, intuitivní ovládání a rozšiřitelnost. Všechny části aplikace budou maximálně jednoduché, aby k jejich ovládání nebylo potřeba hlubších znalostí. Zároveň si ukážeme možnosti rozšíření tak, aby pokročilí uživatelé mohli definovat složité sety příkazů, které potom mohou sdílet s méně zkušenými uživateli. Budou zde také vybrané pasáže zdrojového kódu a návrhové diagramy a koncepty uživatelského rozhraní.

Jednou z podmínek je zaměření na uživatelskou přívětivost a zároveň rozšiřitelnost aplikace, volme postup navrhování aplikace nejprve definicí uživatelského rozhraní a nakreslení wireframes. Při návrhu UI je dobré řídit se doporučeními pro návrh Android aplikací, tak jak jej stanovila firma Google (1) v programátorské příručce k poslednímu Android SDK, tedy API 16. Protože se Android velmi rychle vyvíjí, nemohu být použity při návrhu UI všechny nejmodernější prvky z nového API, protože ve starších API tyto prvky nejsou podporovány. Naštěstí Google Inc.

vytvořil „Support Library“ (2) která většinu novinek implementuje i ve starších verzích Android SDK. Nejnižší podporované API má číslo 4. Nejvýraznější prvek, který nemůžu z důvodu podpory starších zařízení je ActionBar (3). Tedy nástroj pro pohodlnější navigaci uživatele v aplikaci. Naštěstí se dá do určité míry simulovat prostřednictvím starších menu.

Druhým krokem návrhu bude vytvoření datové vrstvy aplikace. Protože úkolem umožnit uživatelům jednoduché rozšíření o další pluginy pro ovládání více činností počítače, je potřeba se zaměřit na potřebnou manipulaci s datovými zdroji, jejich správné udržování a poskytování v rámci aplikace ostatním komponentám. Při návrhu bude využito třech standardních datových úložišť, které Android poskytuje. Je to databáze SQLite, souborový systém SD karty a key-value databáze, určená zejména pro uložení nastavení aplikace. Android poskytuje všeobecný návrhový vzor, jak poskytovat data z aplikace i ostatním aplikacím v systému, který se jmenuje „Content provider“. Protože není potřeba poskytovat data ostatním aplikacím, nebude se tento vzor využívat.

Posledním krokem je spojení UI a datové složky. Android poskytuje přístup k návrhu aplikací podobný MVC/P a kontrolery nazývá aktivitami. Aktivita je v Androidu jeden ucelený prvek, zajišťující jednu činnost. V minulosti platilo, že jedna aktivita má jeden view (UI prvek) a pracuje pouze s ním a do určité míry se tento princip používá stále. S příchodem tabletů se značně zvětšily dostupné obrazovky pro zobrazování aplikací a proto Google představil nový prvek: fragment. Fragment je část aktivity, která má vlastní view, ale nemůže se zobrazovat samostatně, musí spolupracovat s aktivitou, která jej zobrazí. Výhodou fragmentů je, že jsou znovupoužitelné, tudíž jeden fragment může zobrazovat více aktivit.

Protože by bylo dobré aplikaci i nadále rozvíjet, použiji vývojový proces, který se zakládá na UP. Stanovením vývojových postupů zajistím předpokladatelný vývoj aplikace, jasné vývojové cykly a umožním snazší zapojení a orientaci v projektu případným budoucím vývojářům. Vývojový cyklus se skládá z následujících částí:

A) Slovní vyjádření nových funkcí
B) Definice (změna) uživatelského rozhraní
C) Definice (změna) datové vrstvy aplikace
D) Změny v řídící vrstvě aplikace (změna Aktivit / Fragmentů)
E) Testování
F) Vydání nové verze

Sám Android je navržen tak, aby podporoval podobný vývojový cyklus. Každá nově nahraná verze aplikace na Google Play, což je oficiální obchod s aplikacemi pro Android, musí mít interní číslo aplikace zvýšené minimálně o 1. Dále Android obsahuje mechanismus, který

umožňuje automaticky při prvním startu aktualizované aplikace změnit strukturu databáze, pokud je tato změna potřeba. Tyto dva mechanizmy společně zajišťují hladkou aktualizaci aplikace na všech zařízeních, na kterých je nainstalována a která jsou připojena ke Google Play.

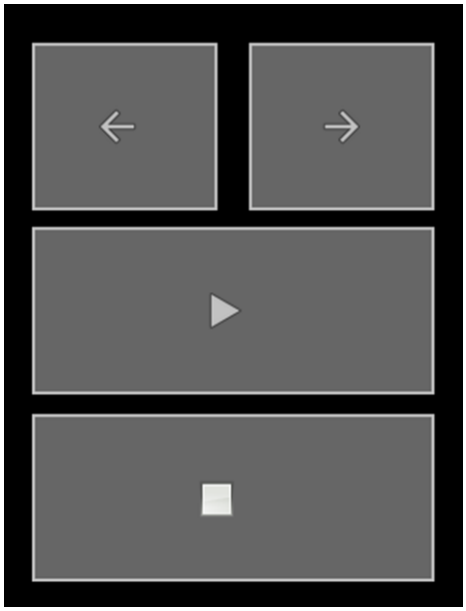
## 3.1 Slovní definice aplikace

Navrhněte a vytvořte aplikaci pro systém Android, která bude komunikovat s počítači s nainstalovaným systémem Linuxového typu pomocí SSH. Aplikace bude určena k ovládání těchto počítačů. Aplikace bude využívat standardních možností linuxových distribucí a nebude vyžadovat instalaci speciálního dodatečného SW (je možné požadovat instalaci softwaru, který se běžně nachází v repozitářích distribucí). Budu využívat zabezpečení SSH protokolu a umožním uživateli přihlásit se jak pomocí uživatelského jména a hesla, tak pomocí jména a SSH klíče. Při návrhu UI budu dodržovat standardy návrhu UI pro systém Android a zároveň budu udržovat UI přehledné pro uživatele. Umožněte uživateli přidávat další ovládané prvky v počítači pomocí pluginů. Jeden plugin bude určen pro ovládání konkrétní aplikace na počítači, případně pro provádění skupiny souvisejících úkolů.

## 3.2 Návrh uživatelského rozhraní

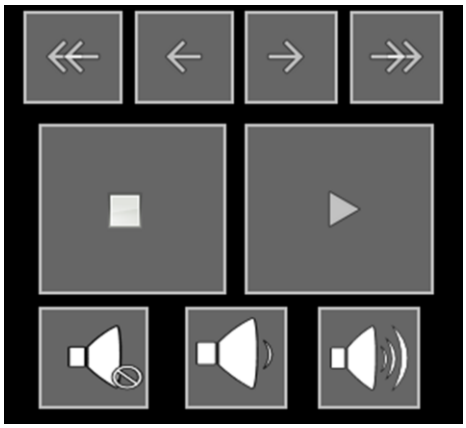
Základem aplikace je ovládání programů spuštěných na počítači, protože chci uživateli dát větší volnost a více možností při vytváření pluginů, navrhl jsem tři základní obrazovky, které budou dále obsluhovat pluginy. V tomto případě to budou fragmenty z důvodu vylepšeného zobrazování aplikace na tabletech.

Obrázek 25 ukazuje základní obrazovku, navrženou jen pro nejjednodušší činnosti. Toto UI může být použito například jako jednoduchý ovladač multimediální aplikace případně ovladač prezentace spuštěné na počítači. Díky své jednoduchosti nebude vyžadovat složité pluginy pro svou funkci. Tlačítka jsou velká, aby se na ně lépe klikalo i bez pohledu na display telefonu, jen podle přibližné polohy. Tím může tento fragment zastupovat funkci jednoduchého ovladače multimediálních programů a umožní rychlé ovládání přehrávače a také prezentujícímu nevšímat si obrazovky telefonu a po paměti klikat na tlačítka.



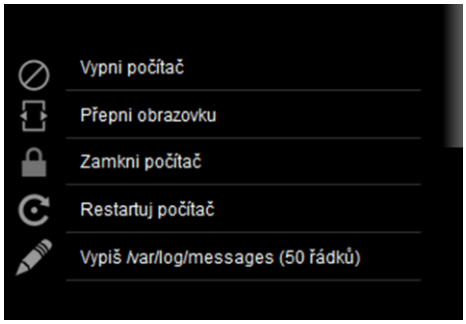
Obrázek 25: SimpleControlFragment

Druhý UI fragment zobrazený na obrázku 26 používá velká hlavní tlačítka z předchozího fragmentu, ale přidává několik menších tlačítek, pro detailnější ovládání aplikace. Tento fragment je přímo navržen pro ovládání multimediálních aplikací, zejména pak přehrávačů filmů, ale dá se použít i pro přehrávače hudby, případně pro prohlížeče obrázků.



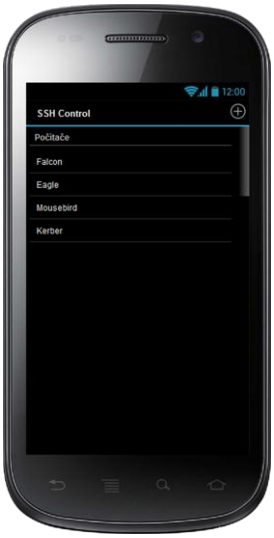
Obrázek 26: AdvancedControlFragment

Poslední navrhovaný fragment, který je na obrázku 27, představuje nejflexibilnější rozhraní, které momentálně aplikace bude poskytovat. Zobrazí totiž list příkazů, kterých bude moci být libovolně mnoho. Bude záležet jen na definici pluginu, tedy na uživateli, kolik příkazů chce umístit na jednu obrazovku. Tento fragment bude vyžadovat nejvíce nastavení i nejsložitější XML soubor, přesto bude v některých případech velmi užitečný.



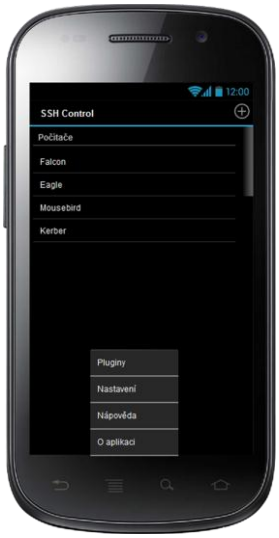

Obrázek 27: CustomControlFragment

Dalším důležitým prvkem aplikace je seznam počítačů, které si uživatel přeje ovládat. Tento seznam bude hlavním rozcestníkem aplikace, bude se tedy zobrazovat jako první po spuštění. Bude umožňovat přidat nový počítač, přihlásit se k počítači, editovat jeho nastavení a samozřejmě smazat počítač. Dále bude umožňovat přepnout se na obrazovku správy pluginů, na obrazovku nastavení aplikace, o aplikaci a také na obrazovku nápovědy. Obrázek 28 zobrazuje výchozí obrazovku aplikace, obrázek 29 zobrazené kontextové menu, které se zobrazí po dlouhém kliknutí na položku seznamu počítačů a obrázek 30 zobrazuje aplikační menu, které umožní přepnutí do dalších obrazovek aplikace. Přepnutí na ovládání počítače se provede kliknutím na položku v seznamu počítačů.




Obrázek 28: ServersActivity


Obrázek 29: ServersActivity - context menu




Obrázek 30: ServersActivity - menu

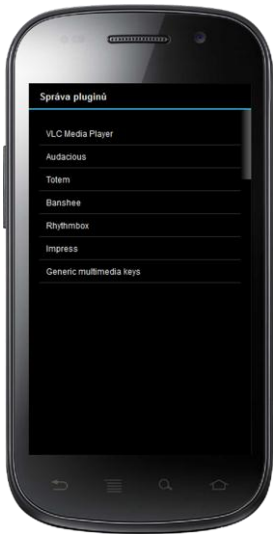
Nový počítač se přidává pomocí tlačítka s vyobrazeným plus. Toto tlačítko je umístěno do panelu s titulkem aplikace, což se považuje za obvyklé při navrhování UI pro Android. Výše je zmiňována nemožnost použití komponenty ActionBar, kvůli nekompatibilitě se staršími API a tudíž se staršími zařízeními. Právě ActionBar přináší oficiální podporu pro umístění důležitých funkcí programu pro Android do vrchní lišty se jménem aplikace, stejně jako umisťuje na pravý okraj

vrchní lišty symbol menu. Při použití ActionBaru se tedy menu může vyvolat jak HW tlačítkem telefonu, tak stisknutím tlačítka pro menu a celé menu se přesune k vrchnímu okraji obrazovky. Tento prvek byl zaveden pro zvýšení uživatelského komfortu, protože přímo ukazuje, že aplikace má i další možnosti skryté v menu. Menu, které se objeví po stisknutí HW tlačítka, totiž není nikde signalizováno. Naštěstí si na toto menu uživatelé Androidu zvykli, proto jej budu používat. Toto rozhodnutí nijak nesníží uživatelský komfort, ale přinese možnost spouštět aplikaci i na starších zařízeních.

Důležitou komponentou aplikace je přidání nového počítače, který chce uživatel ovládat, a přiřazení pluginu, který se má použít pro tento konkrétní počítač. Při přidání nového počítače sice uživatel může vybrat jen jeden plugin, ale nic nebrání tomu, aby přidal více záznamů, které budou směřovat na stejný počítač, ale budou vždy využívat jiný plugin. Obrazovka pro přidání počítače je na obrázku 31.


Obrázek 31: AddServerActivity

Posledním prvkem, který budu navrhovat je seznam pluginů, jak jej zobrazuje obrázek 32. Je to seznam všech pluginů, které jsou nahrány v zařízení. Pluginy budou uloženy na SD kartě, takže půjdou jednoduše upravovat případně kopírovat. Do jedné z příštích verzí aplikace bude přidána podpora pro online repozitář pluginů. Registrovaní uživatelé budou moci pluginy nahrávat a všichni uživatelé budou mít možnost je, přímo v aplikaci, stáhnout do zařízení. Další plánovanou funkcí je i hodnocení pluginu přímo v repozitáři a také prostřednictvím aplikace.




Obrázek 32: PluginManagerActivity

V aplikaci jsou obsaženy samozřejmě i obrazovky s nastavením, nápovědou a informacemi o aplikaci, ale tyto obrazovky nebudou samostatně navrhovány. Nastavení totiž Android SDK generuje samo na základě definice v XML souboru a další dvě obrazovky budou jen text s posuvníkem.

## 3.3 Návrh datové vrstvy aplikace

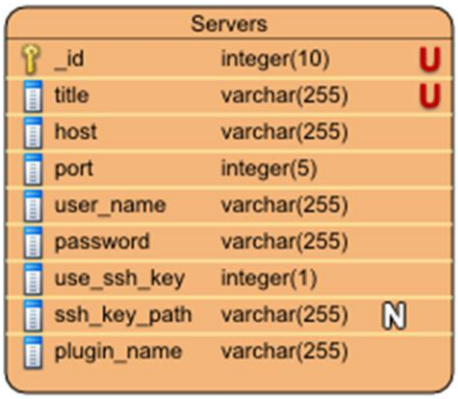
Aplikace bude pracovat se třemi datovými vrstvami. Prvním je seznam počítačů, které chce uživatel ovládat prostřednictvím pluginů, druhým je seznam pluginů, uložený na SD kartě a posledním je uživatelské nastavení aplikace. Protože Android SDK obsahuje třídu, která s nastavením pracuje a zajišťuje jeho ukládání i zobrazování obrazovky pro změnu nastavení (4), nebudu zde navrhovat vlastní prvky pro práci s nastavením. Pro tvorbu nastavení stačí jen správně definovat XML a v něm všechna požadovaná nastavení aplikace.

Seznam počítačů bude uložen v interní SQLite databázi, kterou Android poskytuje všem aplikacím. V této databázi definuji jednu tabulku pro ukládání nastavení všech počítačů, které bude uživatel chtít ovládat. Nad databází bude vytvořena vlastní třída, která bude usnadňovat práci se samotnou databází.

Pluginy budou uloženy na SD kartě ve speciálním adresáři pro ně vyhrazeném. Budou tedy použity třídy pro práci se soubory. Opět bude vytvořena vlastní třídu, která bude obalovat samotnou práci se soubory i práci s pluginy. Protože pluginy budou reprezentovány XML soubory, bude tato třída obalovat i práci s XML formátem dat. Také zde bude vytvořen návrh XML pro definici pluginů.

### 3.3.1 Datové úložiště počítačů

Návrh SQL tabulky je v tomto případě jednoduchý. U počítačů je potřeba ukládat následující data, která vyplívají z aktivity pro přidání počítače: název spojení, adresa počítače, port pro připojení, uživatelské jméno, heslo, cesta k SSH klíči a příznak, jestli je použit a nakonec zvolený plugin pro ovládání aplikace v daném spojení. K mým atributům je potřeba ještě přidat _id. Ten slouží jako primární klíč databáze a musí být pojmenován přesně takto, pokud má být využita vestavěná podpora pro práci s daty, získanými z databáze. Tabulka je vyobrazena na obrázku 33.

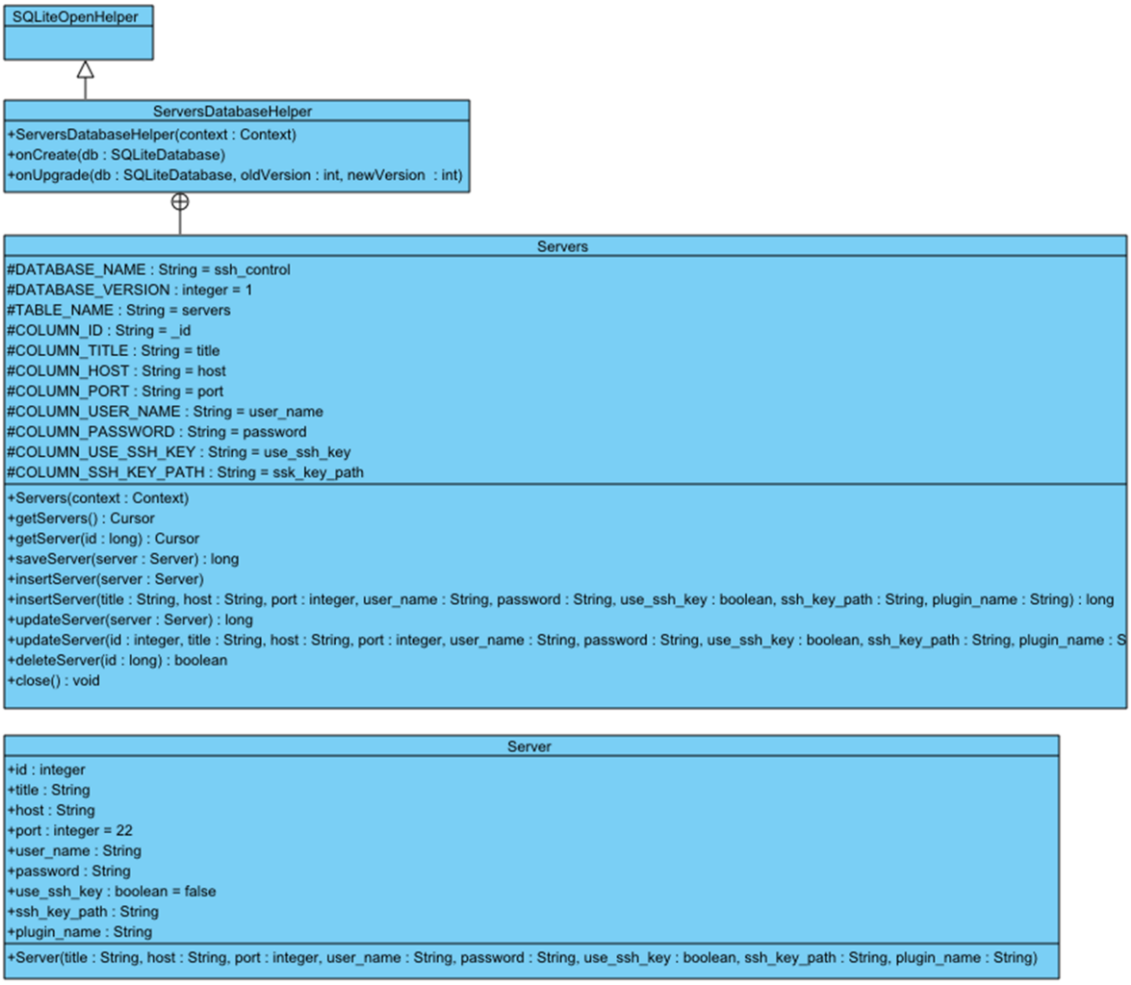



Obrázek 33: Tabulka Servers

Návrh ORM mapování a obsluhy databáze je poněkud složitější. Samotnou obsluhu databáze zajištuje sám systém Android a třída `SQLiteOpenHelper`. Tato třída se stará o připojení k databázi, manipulaci s daty i s databázovými objekty. Zde z této třídy dědím, protože potřebuji zajistit přepsání dvou metod. První je metoda `onCreate(SQLiteDatabase db)`, která je volána při nainstalování aplikace do zařízení. V této metodě se tedy musím postarat o vytvoření databáze, kterou budu používat. Druhou metodou je `onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`, kterou Android volá při aktualizaci aplikace. Tato metoda se musí postarat o případné změny tabulek, pokud při vývojovém cyklu nastanou. Zde by se měly používat nedestruktivní úpravy nad databází tak, aby uživatel aplikace nepřišel o svá data. Ve své ORM třídě tedy budu používat služby zděděné třídy `ServersDatabaseHelper`.

Třída `Server` je pouze obalovou třídou, která zajišťuje lepší manipulaci s daty v rámci aplikace. Uchovává všechny informace o serveru, s kterými je právě potřeba pracovat. Obsahuje jednoduchý konstruktor, který zajišťuje nastavení dat. V návrhu jsou uvedeny veřejné datové složky třídy, což je běžná praxe v UML, používaná pro zkrácení a zpřehlednění návrhu. Při implementaci se poté použijí properties třídy. Java bohužel neobsahuje přímou podporu pro properties, jako například C#. Proto budu properties implementovány pomocí getrů a setrů.

Hlavní datová třída `Servers` bude mít na zodpovědnost připojení a odpojení databáze a práci s položkami databáze. Zpřístupňuje metody pro zpřístupnění seznamu všech počítačů v databázi a také jen jednoho počítače. K tomu využívám podporu databázových kurzorů v Androidu. Kurzor je speciálně upravený iterátor pole záznamů, vrácených z databáze. Díky kurzoru mohu jednodušeji pracovat s celým výsledkem z databáze. Dále zpřístupňuje metody pro vložení nového záznamu a aktualizaci existujícího záznamu. Metody `insertServer()` a `updateServer()` se chovají podle očekávání, tedy vloží nebo upraví v databázi předaný záznam. Specialitou je zde metoda `saveServer()`, která se chová podle dodaných dat. Pokud metoda nalezne v předaných datech i platné ID serveru, aktualizuje záznam, pokud ID nenalezne, nebo je ID neplatné, vytvoří nový záznam. Tento princip umožňuje používat v jakékoliv situaci, která vyžaduje ukládání, metodu `saveServer()` a můžeme tak například zjednodušit metody aktivity pro vytváření a aktualizaci serveru, protože mohou mít větší část společnou. Návrhový diagram je zobrazen na obrázku 34.




**Obrázek 34: Obsluha databáze a ORM**

V návrhu třídy `Servers` existuje několik pojmenovaných konstant, které obsahují pojmenování databázových objektů. Není nutné takto pojmenovávat databázové objekty, ale je to doporučená praxe. Pokud se v budoucnu programátor rozhodne některý objekt v databázi přejmenovat, postačí jej potom přejmenovat pouze na jednom místě v programovém kódu. Tento návrhový princip pomůže předcházet v budoucím vývoji možným chybám, a proto zde bude použit.

### 3.3.2 Datové úložiště pluginů

Pro návrh úložiště pluginů je nejdříve potřeba znát jejich strukturu, aby mohly být navrhnuty třídy, které je budou reprezentovat. Protože pluginy budou uloženy jako XML soubory, bude zde vytvořen návrh těchto XML souborů. Návrh je založený na vzhledu grafického rozhraní fragmentů, které provádí samotné ovládání počítače. Zároveň v každém pluginu musím uchovávat dodatečné informace, podle kterých uživatel plugin pozná, jako je třeba jméno pluginu, popis nebo požadavky na nainstalované programy na ovládaném PC. Zároveň budou přidány dvě speciální položky, které budou spouštěny automaticky: `run` a `quit`.

První je jednoduché rozhraní nazvané `SimpleControlFragment` (Obrázek 25). Toto rozhraní vyžaduje pouze čtyři dynamická tlačítka a samozřejmě dodatečné informace o pluginu. Ukázkový XML soubor pro toto rozhraní vypadá následovně:

```
<?xml version="1.0" encoding="utf-8"?>
<plugin type="simple">
	<author>
		<name>Vojtěch Biberle</name>
		<email>vojtech.biberle@gmail.com</email>
	</author>
	<title>Totem simple</title>
	<description>
Jednoduché ovládání TOTEM Playeru.
	</description>
	<requires>
Nainstalovaný TOTEM Player.
	</requires>

	<!--nasleduje blok prikazu pro ovladani -->
	<run>
		<![CDATA[ export DISPLAY=:0; totem& ]]>
	</run>
	<play>
		<![CDATA[ totem --play-pause; ]]>
	</play>
	<stop>
		<![CDATA[ totem --pause; ]]>
	</stop>
	<next>
		<![CDATA[ totem --next; ]]>
	</next>
	<prev>
		<![CDATA[ totem --prev; ]]>
	</prev>
	<quit>
		<![CDATA[ totem --quit; ]]>
	</quit>
</plugin>
```

Druhé rozhraní, reprezentované fragmentem `AdvancedControlFragment` (Obrázek 26) vytváří stejnou funkcionalitu, jako jednoduché rozhraní. Rozdílné je pouze v definici více ovládacích prvků. Bude zde tedy definováno více XML elementů pro možné ovládací příkazy. Shodné definování těchto dvou rozhraní také umožní, při vytváření obslužných tříd, použít dědění a zjednodušit tím návrh i údržbu aplikace. Druhé rozhraní tedy reprezentuje následující XML soubor:

```
<?xml version="1.0" encoding="utf-8"?>
<plugin type="advanced">
	<author>
		<name>Vojtěch Biberle</name>
		<email>vojtech.biberle@gmail.com</email>
	</author>
	<title>Totem advanced</title>
	<description>
Rozšířené ovládání TOTEM Playeru.
	</description>
	<requires>
Nainstalovaný TOTEM Player a xdotool.
	</requires>

	<!-- nasleduje blok prikazu pro ovladani -->
	<run>
		<![CDATA[ export DISPLAY=:0; totem& ]]>
	</run>
	<play>
		<![CDATA[ totem --play-pause; ]]>
	</play>
	<stop>
		<![CDATA[ totem --pause; ]]>
	</stop>
	<next>
		<![CDATA[ totem --next; ]]>
	</next>
	<prev>
		<![CDATA[ totem --prev; ]]>
	</prev>
	<forward>
		<![CDATA[ totem --seek-fwd; ]]>
	</forward>
	<backward>
		<![CDATA[ totem --seek-bwd; ]]>
	</backward>
	<volup>
		<![CDATA[ xdotool key XF86AudioRaiseVolume; ]]>
	</volup>
	<voldown>
		<![CDATA[ xdotool key XF86AudioLowerVolume; ]]>
	</voldown>
	<volmute>
		<![CDATA[ xdotool key XF86AudioMute; ]]>
	</volmute>
	<quit>
		<![CDATA[ totem --quit; ]]>
	</quit>
</plugin>
```

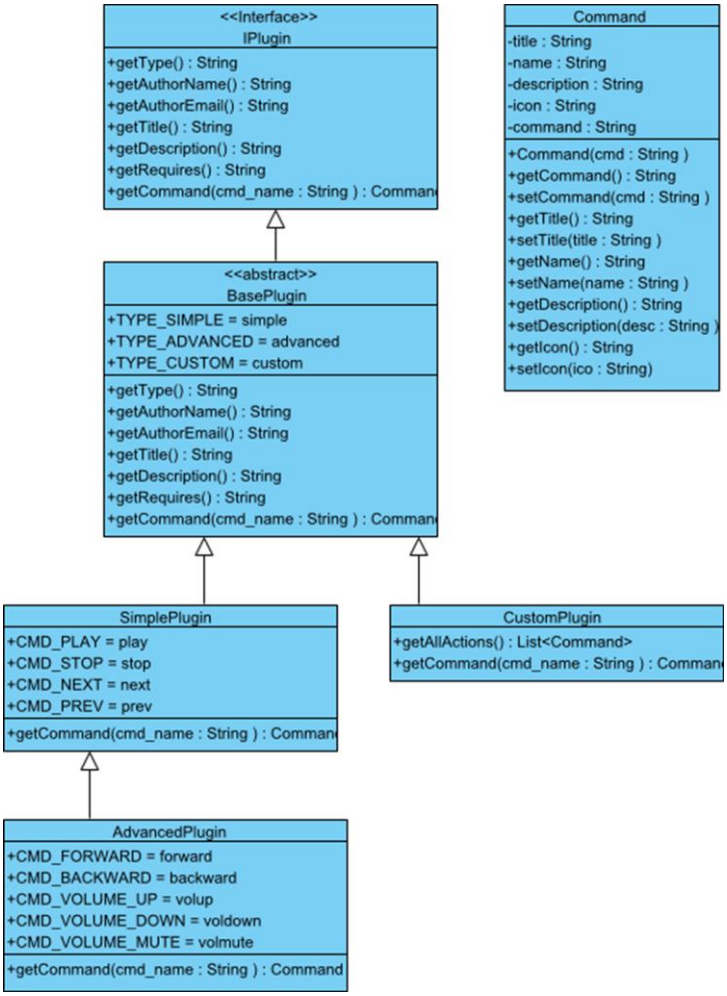
Poslední navrhnuté rozhraní, které tvoří fragment CustomControlFragment (Obrázek 27), umožňuje tvůrci pluginu definovat libovolný počet příkazů, uspořádaný volně pod sebou. Toto rozhraní tedy vyžaduje odlišný přístup při uchovávání dat v XML a samozřejmě i jinak navrhnutý XML soubor. Zde bude vytvořen XML soubor pouze pro tři příkazy, ale položka `action` se může

libovolně opakovat. Jeden element `action` reprezentuje jednu položku v seznamu příkazů, kterou aplikace zobrazí. XML vypadá takto:

```
<?xml version="1.0" encoding="utf-8"?>
<plugin type="advanced">
	<author>
		<name>Vojtěch Biberle</name>
		<email>vojtech.biberle@gmail.com</email>
	</author>
	<title>Custom Volume control</title>
	<description>
Ovládání hlasitosti.
	</description>
	<requires>
Nainstalovaný xdotool.
	</requires>

	<actionlist>
		<action>
			<title>Volume UP</title>
			<name>volume_up</name> <!-- programove rozeznatelny nazev akce -->
			<desc></desc>
			<icon>relative/path/to/icon.ico</icon>
			<command>
				<![CDATA[ DISPLAY=:0; xdotool key XF86AudioRaiseVolume; ]]>
			</command>
		</action>
		<action>
			<title>Volume Down</title>
			<name>volume_down</name> <!-- programove rozeznatelny nazev akce -->
			<desc></desc>
			<icon>relative/path/to/icon.ico</icon>
			<command>
				<![CDATA[ DISPLAY=:0; xdotool key XF86AudioLowerVolume; ]]>
			</command>
		</action>
		<action>
			<title>Volume Mute</title>
			<name>volume_mute</name> <!-- programove rozeznatelny nazev akce -->
			<desc></desc>
			<icon>relative/path/to/icon.ico</icon>
			<command>
				<![CDATA[ DISPLAY=:0; xdotool key XF86AudioMute; ]]>
			</command>
		</action>
	</actionlist>
</plugin>
```

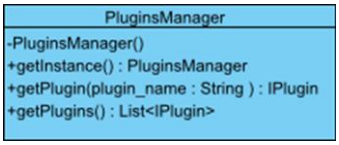
Nyní máme definována data, se kterými bude pracovat úložiště pluginů. Prvním krokem v návrhu úložiště je vytvoření objektové reprezentace XML souborů s pluginy. Aby byla zachována možnost využití polymorfizmu a také seskupen společný programový kód tříd na jedno místo, použijeme jako základ abstraktní bázovou třídu všech pluginů. Zároveň zachováme možnost vytvoření naprosto jiné implementace pluginů se stejným chováním. Proto bude vytvořený i interface, který bude definovat povinné chování pluginů. Bázová třída bude poté dědit od interface a bude implementovat chování pluginů. Každá třída, reprezentující typ pluginu bude poté dědit z bázové třídy, případně z třídy pluginu (toto bude případ Simple a Advanced pluginu, kde třída Advanced pluginu bude rozšiřovat funkčnost Simple pluginu). Diagram tříd pro tuto část plugin manageru je na obrázku 35.




Obrázek 35: Diagram pluginů

V diagramu na obrázku 35 je znázorněna i třída `Command`, která zastupuje příkaz, případně skupinu příkazů, načtených z XML souboru pluginu. Nyní tato třída slouží pouze jako jednosměrná přepravka pro příkazy v podobě řetězce a třídě `CustomPlugin` zajišťuje i přepravku dalších parametrů, které obsahuje element `action`. Zde by mohlo být užito pouze předávání příkazů pomocí řetězců, ale pokud bychom chtěli v budoucnu rozšířit možnosti příkazu, třeba o vypisování navrácených dat, museli bychom celou aplikaci přepracovávat. Proto je hned používán návrhový vzor přepravka, i když zde není přímo nutný.

Posledním prvkem celého datového úložiště je třída `PluginsManager`. Ta se stará o načítání pluginů podle jejich jména ze složky pluginů a o výpis všech pluginů. Je užit návrhový vzor Singletone, protože není potřeba více instancí této třídy. Aktivita vždy požádá o instanci třídy a poté danou instanci požádá o vytvoření příslušného pluginu podle zadaného jména pluginu. Třída `PluginsManager` požadovaný objekt pluginu vytvoří a předá k dalšímu použití aktivitě. Diagram třídy je na obrázku 36.

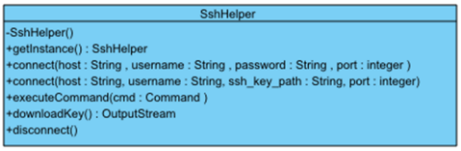



Obrázek 36: PluginsManager

Součástí třídy `PluginsManager` je i načítání a parsování XML souborů. Tyto činnosti nejsou zaznamenány v návrhu, protože je budou zajišťovat privátní metody třídy `PluginsManager`. Čtení souboru bude zajišťovat metoda `openFileInput` třídy `Context`, která vrací `InputStream` (5). Parsovat XML budu pomocí XmlPullParser, která je opět součástí Android SDK (6).

## 3.4 Návrh komunikační vrstvy

Od komunikační vrstvy je požadováno připojení pomocí SSH k ovládanému počítači, spuštění příkazu na počítači a stáhnutí veřejného klíče, pro připojování pomocí klíče. Komunikační vrstvu představuje třída `SshHelper`, která obaluje SSH knihovnu JSch (7). Tato třída, bude implementovat návrhový vzor `Singletone`, protože aplikace vždy bude připojena pouze k jednomu PC, které bude ovládat. Diagram třídy je na obrázku 37.




Obrázek 37: SshHelper

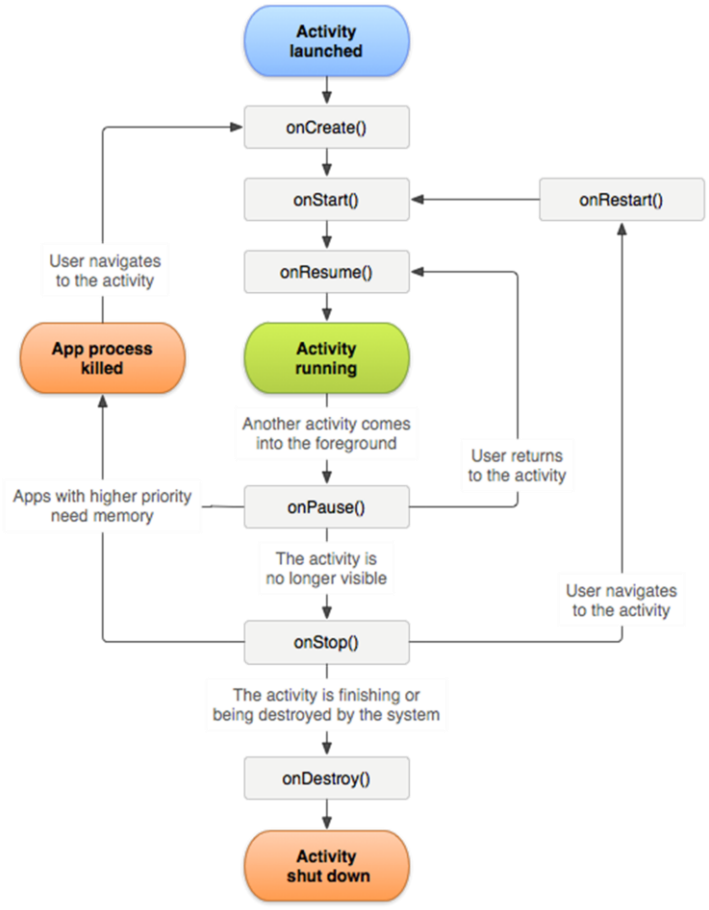
## 3.5 Návrh fragmentů a aktivit aplikace

Aktivity jsou řídícím prvkem celé aplikace pro Android. Každá aktivita má přiřazeno jedno view, což je grafický pohled na data aplikace. View posílá aktivitě ke zpracování události, které nastaly na jeho grafických prvcích. Aktivita dále může komunikovat pomocí zpráv s ostatními aktivitami, třídami nebo i aplikacemi v systému Android. Vnitřní komunikaci v rámci aplikace většinou zajišťuje volání metod jiných objektů. Pro vnější komunikaci s ostatními aplikacemi nabízí Android SDK takzvané Intenty. Intent je univerzální komunikační prostředek, který lze použít pro komunikaci v rámci jedné aplikace i mezi více aplikacemi. Intent obaluje linuxovou meziprocesovou komunikaci a nabízí možnost přibalit ke zprávě dodatečná data pro cílovou aplikaci.

Systém Android obstarává veškerou práci s procesy sám. Rozhoduje, kdy který proces bude běžet, nebo bude pozastaven případně zabit. Je to dáno možnostmi zařízení, na kterých je Android spouštěn. Ve většině případů nemají dostatečně výkonné procesory a velikou

paměť RAM, aby mohlo běžet současně několik aplikací. Android zároveň ale umožňuje uživateli přepínat obrazovky, mezi aplikacemi i aplikace samotné. Důsledkem toho je, že systém může kdykoliv zabít proces a uklidit po něm prostředky, ale uživatel se k němu bude chtít vrátit a pokračovat v práci tam, kde přestal. Proto systém Android přináší přesně definovaný životní cyklus aktivity založený na spouštění metod při různých událostech. Aby programátor mohl zajistit obnovu stopnuté nebo zabité aplikace do stavu, ve kterém ji uživatel opustil naposled, poskytuje Android systém `Bundle`. `Bundle` je speciální implementací návrhového vzoru přepravka. Poskytuje dočasné úložiště pro základní datové typy, pro aplikace, které byly pozastaveny. Tyto bundle sytém Android nemaže, ale při obnovení činnosti aplikace je předá aplikaci. Na programátorovi je, aby zajistil správné uložení hodnot do bundle a následné vybrání a nastavení stavu aplikace.

Pro zajištění správného chodu aplikace je tedy potřeba dědit všechny aktivity od třídy `Activity` případně od potomků této třídy. Dále je potřeba správně přepsat metody, potřebné k chodu aktivity. Na obrázku 38 je ukázán a popsán životní cyklus aktivity.



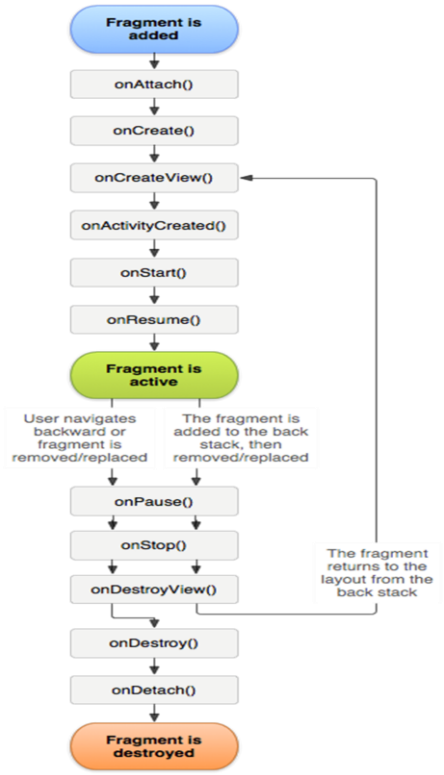

Obrázek 38: Životní cyklus aktivity

Zdroj: http://developer.android.com/reference/android/app/Activity.html

Všechny aktivity musí dodržet toto ujednání a přepsat potřebné metody. Ne všechny metody je vždy nutné přepsat. Záleží na uvážení programátora, jestli potřebuje reagovat na správy od systému, v jakém stavu se zrovna nachází aktivita. Nutností je ale přepsání metody `onCreate()`, ve které se nastavuje pohled, který je přiřazen aktivitě. Bez nastaveného pohledu uživatel nemá prostředek, jak komunikovat s aktivitou, tudíž je taková aktivita nepoužitelná. Doporučuje se také přepsání metody `onDestroy()` pokud programátor používá prostředky, které musí uvolnit.

Další stavební jednotkou aplikací pro Android jsou fragmenty. Fragment je část uživatelského UI, která je oddělena do samostatného funkčního celku. Platí totiž, jak aktivita nemá smysl bez view, tak nemůžeme spustit view bez aktivity nebo fragmentu. Fragmenty tedy slouží na rozdělení aplikace do menších funkčních UI celků, které jsou znovupoužitelné. Dále fragmenty dovolují zjednodušení programování aplikací, protože velmi usnadňují tvorbu přizpůsobitelného layoutu aplikace, který je poté možno použít jak na telefonech, tak na tabletech, kde takovýto layout využívá efektivněji místo dostupné na velké obrazovce. Fragment nikdy nemůže být spuštěn sám a vždy je závislý na své rodičovské aktivitě. Má podobný životní cyklus, jako aktivita a je pevně svázán s životním cyklem aktivity. Žádný fragment, který je potomkem pozastavené aktivity nemůže být spuštěn a pokud systém zničí aktivitu, zničí automaticky i všechny fragmenty, kterých je rodičem.

Fragment může být za běhu aplikace vytvořen a připojen k aktivitě. Programátor může i rozhodnout, který z více fragmentů se připojí v dané situaci. Životní cyklus fragmentu je tedy přizpůsoben těmto podmínkám a přidává metody, které oznamují připojení k aktivitě, vytvoření aktivity a vytvoření view. Kompletní životní cyklus je na obrázku 39.




Obrázek 39: Životní cyklus fragmentu

Zdroj: http://developer.android.com/guide/components/fragments.html

Fragmenty, stejně jako aktivity, nemusí přepsat všechny dostupné metody, které poskytuje Android SKD a informuje jimi o stavu fragmentu. Opět je ale nutné přepsat minimálně metodu `onCreate()`, ve které, se stejně jako v aktivitě, připojuje view k fragmentu. V návrhu aktivit a fragmentů tedy bude každá třída přepisovat jednu nebo i více metod, uvedených na obrázcích výše. Zároveň je samozřejmě možné do aktivit i fragmentů přidat libovolný počet vlastních metod pro jakékoliv účely aplikace.

### 3.5.1 SSHControlMainActivity

Hlavní aktivita celé aplikace, která slouží jako rozcestník pro uživatele. Hlavním úkolem je zobrazit uživateli seznam všech ovládaných počítačů, které si přidal do aplikace. Nad tímto seznamem bude moci uživatel provádět základní editační úpravy, jako přidat nový počítač, editovat nebo smazat stávající počítače a hlavně se připojit k danému počítači a ovládat jej. Pomocí aplikačního menu bude moci uživatel přejít do nastavení, případně do správy pluginů. Aktivita spolupracuje s úložištěm počítačů a získává od něj seznam všech počítačů v databázi, které zpracovává pomocí cursoru.

### 3.5.2 AddServerActivity

Aktivita, která má za úkol zobrazit uživateli formulář pro přidání serveru a zpracovat a uložit data, která uživatel zadá. Aktivita spolupracuje s úložištěm počítačů a dodává mu data, která má uložiště vložit do databáze. Aktivita dále spolupracuje s třídou `SshHelper`, pomocí které nabídne uživateli možnost stáhnout SSH klíč do telefonu. Aktivita nejdříve zjistí jméno počítače, pokud už je stažen klíč stejného jména, použije jej, pokud není, pokusí se klíč stáhnout z počítače.

### 3.5.3 ControlActivity

Tato aktivita se skládá z dalších třech fragmentů, které reprezentují jednotlivé typy pluginů. Aktivita obdrží od SSHControlMainAktivity ID číslo serveru, ke kterému se má připojit. Dále požádá úložiště počítačů o kompletní informace o serveru s daným ID. Pokud tyto informace dostane, vytvoří pomocí PluginsManageru novou instanci pluginu, který má počítač obsluhovat a také připojí jeden ze třech fragmentů. Dále pomocí třídy SshHelper provede připojení ke vzdálenému počítači a vyčkává na stisknutí tlačítka od uživatele. Podle stisknutého tlačítka předá instanci pluginu konstantu, která reprezentuje příkaz pluginu a plugin vrátí objekt, který reprezentuje konkrétní příkaz pro ovládaný počítač. Aktivita dále předá objekt instanci třídy `SshHelper`. Tato třída pomocí dříve vytvořeného spojení příkaz odešle na ovládaný počítač.

Tato jediná aktivita má závislý stav na uživateli, konkrétně na vybraném serveru. Proto bude obsluhovat metody, které zajistí předání `Bundle` k uložení a následné vyzvednutí hodnot. Do `Bundle` ukládám ID serveru, který právě uživatel ovládá. Pokud `Bundle` bude obsahovat požadovaná data, tedy ID, aktivita provede všechny potřebné kroky ke spojení s počítačem a zobrazení fragmentu, který patří k pluginu pro ovládání. Uživatel se tak bude moci vrátit k ovládání i po přerušení činnosti aktivity nebo při jejím úplném vypnutí.

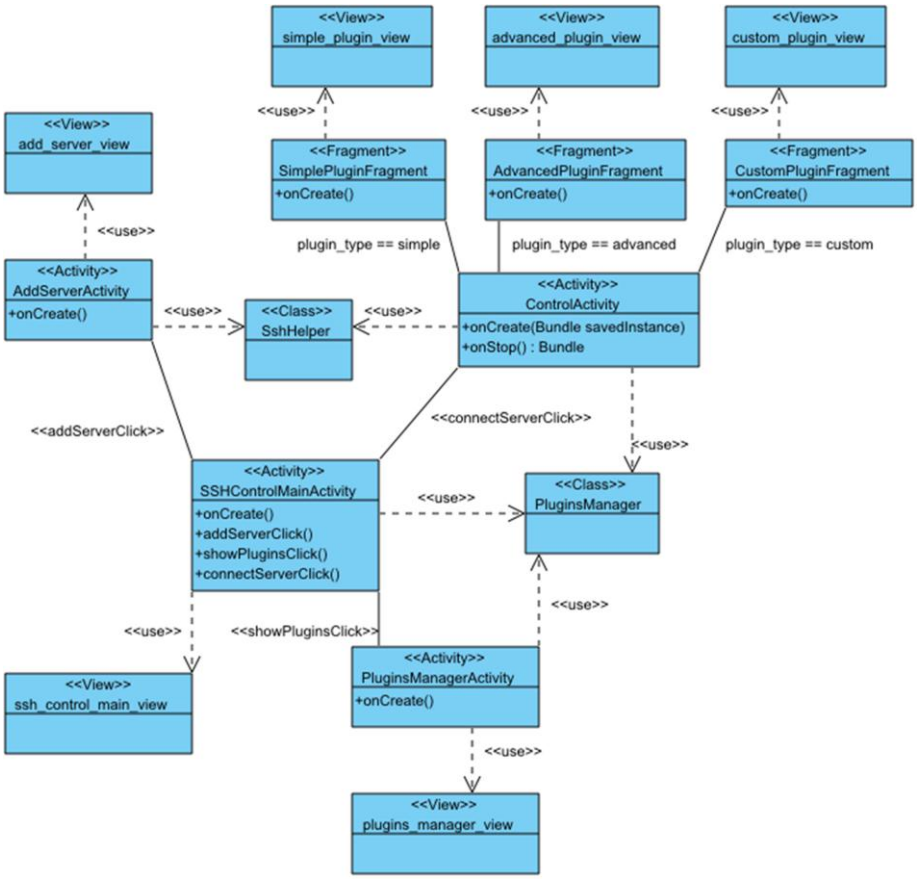
### 3.5.4 PluginsManagerActivity

Momentálně slouží aktivita pro zobrazení všech pluginů v zařízení a zobrazení informací o pluginu. Protože jsou pluginy uloženy na externím úložišti, které je dostupné uživateli, může uživatel jednoduše mazat, nahrávat i upravovat pluginy pomocí jakéhokoliv souborového manažera pro Android nebo může telefon propojit s počítačem a upravovat vše na počítači.

Do budoucna je počítáno s rozšířením aktivity o možnosti stahovat pluginy přímo z internetu, kde bude vytvořen repozitář pluginů, do kterého budou moci registrovaní uživatelé vkládat nové pluginy, hodnotit stávající pluginy ostatních uživatelů a všichni uživatele budou moci pluginy stahovat.

### 3.5.5 Diagram aktivit

Na následujícím diagramu v obrázku 40 jsou znázorněny hlavní prvky aplikace a jejich propojení a spolupráci. UML naštěstí umožňuje definovat vlastní stereotypy, které označují specifika daného objektu, případně spojení. Bylo tedy využito možnosti vytvořit vlastní stereotypy a označit všechny objekty v diagramu odpovídajícími značkami. Dále bylo užito dvou typů vazeb. První je typ „use“ který značí využití třídy nebo objektu. Takto je značeno spojení aktivit a view a také využití některé pomocné třídy aktivitou. Druhým typem spojení je označení přechodu z jedné aktivity na druhou. Tato spojení jsou zakončena příponou „Click“, která značí kliknutí na některý grafický prvek a přechod na novou aktivitu. Stejně tak značí metodu, která celý přechod vyvolá. U fragmentů má toto spojení stejný význam, ale jelikož se fragmenty nespouští kliknutím na UI element, ale splněním podmínky, byla vepsána, do parametru spojení, tato podmínku.




Obrázek 40: Diagram aktivit

# 4 Implementační detaily

Tato kapitola se věnuje implementaci aplikace. Na začátku je popsáno založení projektu ve vývojovém prostředí Eclipse s ADT pluginem (Android Development Tools plugin) a parametry, kterých je užito, při zakládání projektu aplikace SSH Control.

Poté bude vytvořeno grafické rozhraní podle wireframů a uzpůsobení bude takové, aby co nejvíce napodobovalo `ActionBar`, který nechceme, kvůli nedostatečné kompatibilitě se staršími zařízeními použít. Při vytváření budou popsány specifika definice UI v Android SDK.

Následně se implementují datové třídy nad databází SQLite a nad souborovým systémem na externím úložišti telefonu, kde budou uloženy XML soubory s pluginy. Tyto třídy získají většinu své funkcionality ze svých rodičů, konkrétně od třídy `SQLiteOpenHelper` dědí třída pro práci s databází a třídu `XmlPullParser` bude vhodně využívat třída pro práci s pluginy.

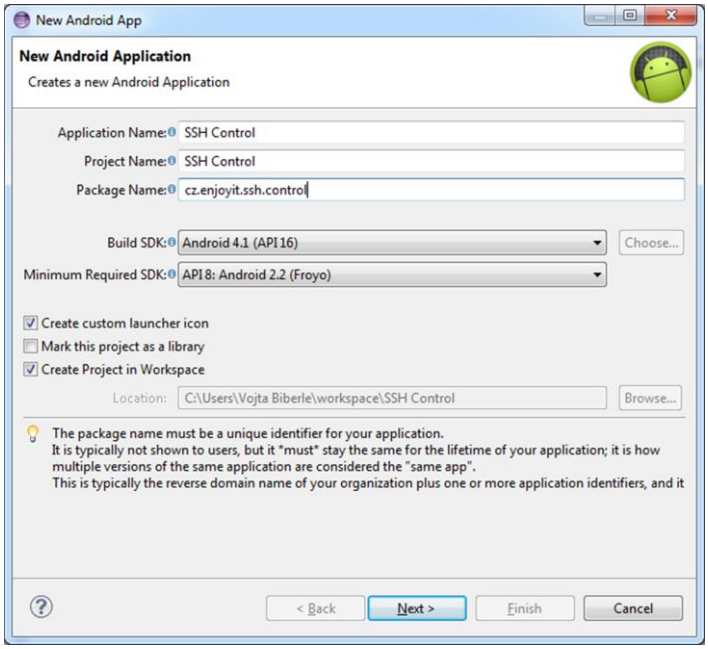
Po té budou vytvořeny pomocné třídy a to hlavně třída zajišťující komunikaci po SSH. Tato třída má také na starost stažení veřejného SSH klíče z počítače, pokud si to uživatel vyžádá. Tento klíč bude ukládán v interním úložišti telefonu. Interní úložiště je zvoleno z důvodu toho, že systém Android vyhradí prostor jen pro tuto aplikaci a žádná jiná aplikace nebude moci ke klíčům přistupovat. Stejně tak se veškeré klíče automaticky smažou, pokud uživatel odinstaluje aplikaci.

Poté všechny části budou spojeny dohromady prostřednictvím aktivit a budou uvedena některá specifika nastavení práv aplikací pro Android. Každá aplikace musí totiž uvádět, jaké zdroje hodlá využívat a až uživatel se rozhodne, jestli jí tyto zdroje povolí využívat.

## 4.1 Založení projektu

Založení projektu se skládá ze 3 kroků. V prvním kroku je potřeba nastavit jméno projektu a jméno aplikace. Jméno projektu se zobrazuje jen vývojáři v prostředí Eclipse. Jméno aplikace se použije při vytváření popisku v Launcheru zařízení s Androidem, ale jde velice jednoduše změnit. Další položkou je jméno balíčku. Balíček aplikace musí mít unikátní jméno na celém světě. Doporučuje se proto používat převrácenou webovou adresu, kterou vlastní osoba (firma) vyvíjející aplikaci. Poslední dvě položky, které je potřeba nastavit jsou sestavovací verze SDK a minimální verze SDK. Číslo sestavovací verse SDK se použije pro určení, s jakým SDK se má aplikace kompilovat a také určuje, jestli bude moci aplikace využívat třídy v daném SDK. Minimální verze SDK se použije při kompilování zpětné podpory nových věcí, do starších verzí Androidu ze „Support Library“. Android se totiž velice rychle vyvíjí a některé funkce rychle zastarávají. Proto je vhodné používat pro kompilování nejvyšší možné SDK (nejnovější), aby aplikace mohla využít nejnovější funkce. Zároveň některé funkce nejsou plně podporovány pomocí Support Library ve všech verzích, proto například nebude užito ActionBar. Použijeme nastavení stavovací SDK verze

16, což je označení pro Android 4.1 a minimální verzi SDK 8, což je Android 2.2. Toto nastavení doporučuje i Google, který uvádí, že s tímto nastavením aplikace bez problému poběží na 98 % všech zařízení s Androidem (samozřejmě při správném použití Support Library). Na obrázku 41 je první krok vytváření projektu.




Obrázek 41: Vytváření projektu - první krok

Druhým krokem je velice jednoduchý průvodce na volbu ikony aplikace. Umožňuje buď přímo vytvořit jednoduchou ikonu složenou z jednoduchého obrázku a dvou barev, nebo vložit svou vlastní ikonu aplikace.

Třetí krok nabízí možnost vytvořit „MasterDetail“ layout. Tento layout se používá pro zobrazení na tabletech, ale je možné jej použít až od API verze 11. Zde je nastavena verze 8 a tudíž tuto možnost není možné využít. Master detail layout se dá udělat s trochu práce i ručně a tak, aby fungoval i na nižších verzích Androida.

Posledním krokem je pojmenování první vygenerované Aktivity a prvního vygenerovaného View. Při pojmenování aktivit se držíme standardu, pojmenovávat aktivity s příponou Activity. Proto bude pojmenována hlavní aktivita „SSHControlActivity“. Pro pojmenování view použijeme vzor „nazev_aktivity_activity“, který zajistí abecední řazení layoutů ve složce projektu a zachová přehled, k jakému objektu layout patří.

Tímto je založení projektu hotové a v levé části je vidět strom projektu. V adresáři src jsou vlastní zdrojové kódy a v adresáři res jsou zdroje a layouty aplikace.

## 4.2 Definice UI

Android SDK definuje pro každý grafický prvek jednu třídu. Všechny třídy mají společného rodiče ve třídě `View`. Tato skutečnost umožňuje se všemi grafickými prvky pracovat velice podobně, některé prvky se dokonce liší jen v hodnotách svých parametrů pro vykreslování (textové editační políčka a tlačítka). V Android SDK tedy můžeme vytvořit jakýkoliv grafický prvek dynamicky, za běhu programu. Přesto se celkové UI definuje v Androidu pomocí XML. Android SDK definuje ke každé třídě grafického prvku odpovídající XML tag a zpřístupňuje všechny vlastnosti třídy jako atributy XML tagu. Typická definice UI vypadá takto (simple_control_fragment.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="horizontal" >

        <ImageButton
            android:id="@+id/btn_prev"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:contentDescription="@string/btn_desc_prev"
            android:src="@drawable/prev" />

        <ImageButton
            android:id="@+id/btn_next"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:contentDescription="@string/btn_desc_next"
            android:src="@drawable/next" />

    </LinearLayout>

    <ImageButton
        android:id="@+id/btn_play"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:contentDescription="@string/btn_desc_play"
        android:src="@drawable/play_pause" />

    <ImageButton
        android:id="@+id/btn_stop"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:contentDescription="@string/btn_desc_stop"
        android:src="@drawable/stop" />

</LinearLayout>
```

## 4.3 Datové třídy

Datová úložiště v aplikaci reprezentují dva odlišné typy objektů. Prvním je třída `Servers`, která obstarává rozhraní nad databází SQLite. Druhým je třída `PluginsManager`, která zapouzdřuje čtení XML souborů, reprezentujících pluginy, z externího úložiště a také mapování informací z XML do objektů, které reprezentují jednotlivé druhy pluginů. Tyto dvě třídy tvoří datovou vrstvu aplikace, ale ještě může být využito interní key-value úložiště pro ukládání stavu aplikace a pro uložení uživatelských nastavení aplikace, pro které se využívá nejčastěji.

### 4.3.1 Databázové třídy

Android SDK poskytuje plnou podporu SQLite databáze. Zajišťuje také zabezpečený přístup k tabulkám databáze. Každá aplikace vlastní svou databázi, ve které může vytvářet libovolný počet tabulek. Všechny tabulky, vytvořené v jedné třídě aplikace jsou přístupné i všem ostatním třídám v rámci aplikace. Neexistuje však způsob, jak by mohla jiná aplikace přistoupit k těmto tabulkám.

Doporučovanou metodou pro práci s databází je vytvoření třídy, která dědí z třídy `SQLiteOpenHelper` a přepsat minimálně metodu `onCreate(SQLiteDatabase db)`. V této metodě je potřeba zavolat metodu `execSQL` na předaném objektu `SQLiteDatabase` a vytvořit pomocí ní novou tabulku. Jelikož tuto pomocnou třídu bude používat jen třída `Servers`, je implementována jako vnitřní třída třídy `Servers`:

```
public class Servers {

	/* zde jsou statické finální konstanty, které obě třídy využívají */

	private SQLiteOpenHelper dbHelper;

	static class ServersDatabaseHelper extends SQLiteOpenHelper {

		ServersDatabaseHelper(Context context) {
			super(context, DATABASE_NAME, null, DATABASE_VERSION);
		}

		@Override
		public void onCreate(SQLiteDatabase db) {
			db.execSQL("CREATE TABLE " + TABLE_NAME + " ("
					+ COLUMN_ID + "INTEGER PRIMARY KEY,"
					+ COLUMN_TITLE + "TEXT NOT NULL, "
					+ COLUMN_HOST + "TEXT NOT NULL, "
					+ COLUMN_PORT + "INTEGER NOT NULL, "
					+ COLUMN_USER_NAME + "TEXT NOT NULL, "
					+ COLUMN_PASSWORD + "TEXT, "
					+ COLUMN_USE_SSH_KEY + "INTEGER NOT NULL, "
					+ COLUMN_SSH_KEY_PATH + "TEXT, "
					+ COLUMN_PLUGIN_NAME + "TEXT NOT NULL"
					+ ");");
		}
	}

	/* dále pokračují metody pro práci s objekty Server */
}
```

### 4.3.2 Souborové třídy

Vytvoření souborové třídy přináší tyto problémy:

- Prvotní nahrání pluginů, dodávaných s aplikací, do adresáře na externím úložišti při instalaci.
- Načtení souboru pluginu a převedení na objektovou reprezentaci pomocí parsování XML.

Protože Android neposkytuje notifikaci, že je aplikace nově nainstalována na zařízení, je potřeba kontrolovat, jestli složka pluginů existuje na externím úložišti a pokud ne, složku vytvořit a nahrát do ní všechny dodávané pluginy. Pro uchování a distribuci pluginů, spolu s aplikací, je potřeba uložit pluginy do adresáře projektu aplikace. Nejlépe pro uložení vyhovuje adresář `/assets`. Tento adresář obsluhuje třída `AssetManager` a usnadňuje manipulaci se soubory v tomto adresáři. Dále je potřeba zjistit cestu k externímu úložišti zařízení, která se může na různých zařízeních lišit. Pro tento úkol Android poskytuje metodu `Enviromental.getExternalStoragePublicDirectory()` která vrátí vždy správný adresář s připojeným externím úložištěm. Kopírování probíhá pomocí Java tříd `InputStream` a `OutputStream`.

Složitější a zajímavější je parsování XML souborů, pro to lze využít třídu `XmlPullParser`. Tato třída sekvenčně čte a zpracovává XML dokument a uživatele informuje o stavu XML dokumentu definovanými konstantami. Možné zprávy, na které může uživatel třídy zareagovat, jsou:

- START_TAG
- TEXT
- END_TAG
- END_DOCUMENT

V aplikaci je potřeba vytvářet různé objekty, na základě obsahu XML. pokud by třída `PluginsManager` měla být zodpovědná za parsování všech objektů, byla by zbytečně složitá a nepřehledná. Proto je přesunuta zodpovědnost za získání dat z XML na samotné objekty. Třída `PluginsManager` jen vytváří daný objekt podle informací z XML a dále předá parser vytvořenému objektu, který musí být schopen zbytek informací ze souboru získat sám.

```
public IPlugin getPlugin(String plugin_name) {
  IPlugin plugin = null;
  InputStream is;

    try {
      XmlPullParser parser = XmlPullParserFactory.newInstance().newPullParser();
      is = context.openFileInput(pluginDirectory + File.separator + plugin_name + ".xml");
      parser.setInput(is, null);
      int eventType = parser.getEventType();
      String tagName;

      while(eventType != XmlPullParser.END_DOCUMENT) {
        switch(eventType) {
        case XmlPullParser.START_TAG:
          tagName = parser.getName();
          if(tagName.equalsIgnoreCase(BasePlugin.PLUGIN)){
            String type = parser.getAttributeValue(null, BasePlugin.PLUGIN_TYPE);
            if(type.equalsIgnoreCase(BasePlugin.TYPE_SIMPLE)) {
              plugin = new SimplePlugin();
            }
            if(type.equalsIgnoreCase(BasePlugin.TYPE_ADVANCED)) {
              plugin = new AdvancedPlugin();
            }
            if(type.equalsIgnoreCase(BasePlugin.TYPE_CUSTOM)) {
              plugin = new CustomPlugin();
            }
            plugin.parse(parser);
          }
          break;
        }
      }
    }
    catch(XmlPullParserException e) {
      plugin = null;
    }
    catch(IOException e) {
      plugin = null;
    }
  return plugin;
}
```

Větší díl práce tedy zastává samotný objekt pluginu. Musí implementovat metodu `parse(XmlPullParser parser)`, pomocí které jej zavolá hlavní třída. Dále musí umět přečíst pomocí parseru zbytek XML souboru, který mu taktéž předá hlavní třída, a získaná data správně uložit do svých datových složek. Musí také hlídat událost konce XML tagu a v případě, že skončí XML tag, který vyvolal předání parsování do pluginu z hlavní třídy, musí plugin opět vrátit řízení metodě, která jej volala. Následuje ukázka metody parse pro SimplePlugin:

```
@Override
public void parse(XmlPullParser parser) throws XmlPullParserException, IOException {
  int eventType = parser.getEventType();
  String tagName;

  while(eventType != XmlPullParser.END_DOCUMENT) {
    switch(eventType) {
    case XmlPullParser.START_TAG:
      tagName = parser.getName();

      if(tagName.equalsIgnoreCase(BasePlugin.AUTHOR)) {
        this.parseAutor(parser);
      }
      if(tagName.equalsIgnoreCase(BasePlugin.TITLE)) {
        this.setTitle(parser.nextText());
      }
      if(tagName.equalsIgnoreCase(BasePlugin.DESCRIPTION)) {
        this.setDescription(parser.nextText());
      }
      if(tagName.equalsIgnoreCase(BasePlugin.REQUIRES)) {
        this.setRequires(parser.nextText());
      }
      if(tagName.equalsIgnoreCase(RUN)) {
        this.commands.put(RUN, new Command(parser.nextText()));
      }
      if(tagName.equalsIgnoreCase(PLAY)) {
        this.commands.put(PLAY, new Command(parser.nextText()));
      }
      if(tagName.equalsIgnoreCase(STOP)) {
        this.commands.put(STOP, new Command(parser.nextText()));
      }
      if(tagName.equalsIgnoreCase(NEXT)) {
        this.commands.put(NEXT, new Command(parser.nextText()));
      }
      if(tagName.equalsIgnoreCase(PREV)) {
        this.commands.put(PREV, new Command(parser.nextText()));
      }
      if(tagName.equalsIgnoreCase(QUIT)) {
        this.commands.put(QUIT, new Command(parser.nextText()));
      }
      break;
    case XmlPullParser.END_TAG:
      tagName = parser.getName();
      if(tagName.equalsIgnoreCase(BasePlugin.PLUGIN)) {
        return;
      }
      break;
    }
    eventType = parser.next();
  }
}
```

## 4.4 Pomocné třídy

Jedinou pomocnou třídou je momentálně SshHelper. Třída, která zapouzdřuje knihovnu JSch od firmy JCraft Inc. JSch je knihovna pro obsluhu SSH spojení pro jazyk Java (8). Protože aplikace využívá jen malou část celé knihovny, poskytuje SshHelper pouze nezbytně nutné metody, které jsou potřeba pro otevření spojení a spuštění příkazu na vzdáleném stroji pomocí SSH.

## 4.5 Třídy aktivit

Dále přichází na řadu implementace aktivit. Aktivity úzce spolupracují s pohledy. Každý pohled náleží k jedné aktivitě. Dohromady spolu tvoří jednu obrazovku aplikace. Aktivita samotná za určitých okolností může obsluhovat i více pohledů a aktivita bez pohledu nemá v Androidu smysl. Aktivity reprezentují kontrolní jednotky aplikace pro Android.

S příchodem zařízení, s větší obrazovkou, tedy zejména tabletů, bylo potřeba navrhnout flexibilnější jednotky pro návrh a práci s UI. Proto Google představil od API 11 Fragmenty. Fragment umožňuje UI rozčlenit do více oddělených celků, které se potom zobrazují pomocí aktivit. Fragment sám o sobě není schopen zobrazení a vždy potřebuje spolupracovat s aktivitou. Aktivita může rozhodovat, jaké fragmenty se zobrazí na základě velikosti obrazovky zařízení. Může také použít jinou definici layout souboru právě v závislosti na velikosti obrazovky.

Je doporučováno používat k definování UI fragmenty všude, kde je to možné. Aplikace se tak stane přehlednější a lépe přizpůsobitelnou pro různé velikosti obrazovky.

### 4.5.1 SSHControlActivity

Je hlavní aktivitou aplikace. Zobrazí se tedy jako první po spuštění aplikace, funguje jako rozcestník ke všem funkcím aplikace a navíc definuje, jaká ikona a jméno aplikace se mají zobrazit v Android Launcheru – tedy spouštěči aplikací. Tato aktivita sama definuje pouze kontextové menu, použité pro navigaci v aplikaci a jedno tlačítko, které spouští aktivitu pro přidání nového počítače do aplikace. Úzce spolupracuje s fragmentem `ServerListFragment`, který zobrazuje seznam všech počítačů, přidaných do aplikace.

### 4.5.2 ServerListFragment

Fragment pro zobrazení seznamu všech počítačů, které chce uživatel kontrolovat. Fragment je založen na třídě `ListFragment`, což mu umožňuje jednoduše zobrazit seznam položek, v tomto případě počítačů. Fragment dále nabízí možnost kliknutí na položku serveru a spuštění akce. Toto kliknutí sám neobsluhuje, ale předá informace o kliknutí rodičovské aktivitě. Dále nabízí kontextové menu pro smazání serveru. Fragment obsahuje dva zajímavé kusy kódu. Prvním je definice vlastního fragmentu. Třída `ListFragment` totiž definuje vlastní layout a není potřeba jej vytvářet. To ovšem neplatí v situaci, kdy chceme vzhled listu upravit. V aplikaci je tedy vytvořen pro tento fragment i view, který definuje položku `ListView`, ve které přenechá řízení zobrazování plně fragmentu a druhou položku typu `TextView`, které se zobrazí v případě, že uživatel nemá uloženy žádné počítače. Protože se u ListFragmentu nepočítá s definicí view, musíme jej fragmentu přiřadit na nestandartním místě a to v metodě `onCreateView()` místo obvyklého `onCreate()`.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstance) {
  View view = inflater.inflate(R.layout.servers_list_fragment, container, false);
  return view;
}
```

Druhým zajímavým místem je vynucení implementace posluchače v hlavní aktivitě pomocí interface. Zde definujeme interface s jednou metodou, která se má spouštět při kliknutí na položku seznamu a dále využijeme metody `onAttach()`, která se spouští ihned po připojení fragmentu k aktivitě a pokusíme se o přetypování aktivity na tento interface. Pokud přetypování selže, víme, že aktivita neimplementuje interface a tudíž neočekává kliknutí na položku seznamu, což by v tomto případě byla chyba.

```
public static interface OnServerClickedListener {
  public void onServerClicked(long id);
}

OnServerClickedListener listener;

@Override
public void onAttach(Activity activity) {
  super.onAttach(activity);
  try {
    listener = (OnServerClickedListener) activity;
  }
  catch (ClassCastException e) {
    throw new ClassCastException(activity.toString() + "must implement OnServerClickedListener");
  }
}
```

### 4.5.3 AddServerFragment

Definuje formulář, který umožnuje uživateli aplikace zadat data potřebná pro uložení nového serveru. Je definován jako fragment, aby mohl zaujmout na větších obrazovkách míst vedle seznamu serverů. Na menších obrazovkách jej bude obsluhovat samostatná aktivita, která jej pouze zobrazí.

### 4.5.4 SimpleControlFragment

Nejjednodušší fragment, který zajišťuje ovládání počítače. Ke svému fungování potřebuje znát ID serveru v databázi, které mu předá rodičovská aktivita. Dále si vyžádá instanci třídy `SshHelper`, kterou potřebuje pro spojení s počítačem a odeslání příkazů. Z databáze serverů zjistí všechny potřebné údaje o serveru a požádá instanci třídy `PluginsManager` o objekt typu `IPlugin`.

Dále přiřadí každému tlačítku ve svém layoutu nový onClickListener(), ve kterém požádá objekt Plugin o vrácení objektu třídy Command, která následně předá SSH k odeslání na server. Kombinací všech vytvořených tříd je potřeba k odeslání příkazu na server pouze jeden řádek. Všechny tyto akce jsou realizovány v metodě `onViewCreate()`:

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstance) {
	View root = inflater.inflate(R.layout.simple_control_fragment, container, false);

	Servers servers = new Servers(getActivity());
	Cursor s = servers.getServer(serverId);

	ssh = new SshHelper(getActivity());

	int host_index = s.getColumnIndex(Servers.COLUMN_HOST);
	int port_index = s.getColumnIndex(Servers.COLUMN_PORT);
	int user_index = s.getColumnIndex(Servers.COLUMN_USER_NAME);
	int pass_index = s.getColumnIndex(Servers.COLUMN_PASSWORD);
	int plugin_name_index = s.getColumnIndex(Servers.COLUMN_PLUGIN_NAME);


	if(s.getCount() < 0) {
		//TODO: error, nemame server a nemelo by se to stat
	}
	else
	{
		s.moveToFirst();
		String plugin_name = s.getString(plugin_name_index);
		String host = s.getString(host_index);
		String user = s.getString(user_index);
		int port = s.getInt(port_index);
		String pass = s.getString(pass_index);
		plugin = new PluginsManager(getActivity()).getPlugin(plugin_name);
		ssh.connectPassword(host, user, pass, port);¨
		ssh.execCommand(plugin.getCommand(SimplePlugin.RUN));
	}

	Button prev = (Button)root.findViewById(R.id.btn_prev);
	prev.setOnClickListener(new View.OnClickListener() {
		public void onClick(View v) {
			ssh.execCommand(plugin.getCommand(SimplePlugin.PREV));
		}
	});

	Button next = (Button)root.findViewById(R.id.btn_next);
	next.setOnClickListener(new View.OnClickListener() {

		public void onClick(View v) {
			ssh.execCommand(plugin.getCommand(SimplePlugin.NEXT));
		}
	});

	Button play = (Button)root.findViewById(R.id.btn_play);
	play.setOnClickListener(new View.OnClickListener() {

		public void onClick(View v) {
			ssh.execCommand(plugin.getCommand(SimplePlugin.PLAY));
		}
	});

	Button stop = (Button)root.findViewById(R.id.btn_stop);
	stop.setOnClickListener(new View.OnClickListener() {

		public void onClick(View v) {
			ssh.execCommand(plugin.getCommand(SimplePlugin.STOP));
		}
	});

	s.close();
	servers.close();
	return root;
}

@Override
public void onDestroy()
{
	ssh.execCommand(plugin.getCommand(SimplePlugin.QUIT));
}
```

Kombinací navrhnutých tříd jsme docílili velice jednoduché implementace fragmentu pro ovládání počítače. Je zde tedy jednoduchá cesta, jak definovat nový typ pluginů a vytvořit k němu fragment. Jsou sice vyžadovány programátorské schopnosti, ale celá aplikace nabízí jednoduše použitelný základ pro další rozšiřování.

Ostatní fragmenty pro ovládání fungují na stejném principu. Vyžádají si tedy data od databáze a manažera pluginů, pomocnou třídu pro komunikaci přes SSH a pouze definují, na která tlačítka ve view bylo kliknuto a které příkazy z pluginu se mají spustit.

### 4.5.5 Ostatní aktivity a fragmenty

Aplikace obsahuje i další aktivity a fragmenty, ale základní princip fungování je zde vysvětlen a proto nebudou postupně probírány všechny prvky aplikace. S malými změnami používají stejné principy a velice podobné kód, pouze pracují s jinými daty. Například `AdvancedControlFragment` se od `SimpleControlFragment` liší pouze definováním více tlačítek a obsluhou více metod pro posílání příkazů počítači přes SSH.

## 4.6 Nastavení aplikace

Systém Android vyžaduje o aplikaci množství informací, aby ji byl ochoten na zařízení spustit. Tyto informace se zapisují do souboru `AndroidManifest.xml`. Musí zde být uloženy informace o všech aktivitách, které aplikace obsahuje. Programátor dále musí definovat všechna požadovaná práva, která aplikace potřebuje ke svému běhu. V tomto případě právo pro přístup k internetu, použité pro připojení pomocí SSH k ovládaným počítačům a právo pro čtení a zápis na externí úložiště dat. Manifest pro tuto aplikaci tedy vypadá následovně:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cz.enjoyit.ssh.control"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".SSHControlActivity"
            android:label="@string/title_activity_sshcontrol" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="AddServerActivity" />
        <activity android:name="ControlActivity" />
        <activity android:name="PluginsActivity" />
        <activity android:name="SettingsActivity" />
    </application>

</manifest>
```

# Závěr

Při zpracování této práce jsem si procvičil jak návrh aplikací všeobecně, tak přímo návrh aplikací pro android. Použil jsem několik návrhových vzorů, které zjednodušily celkový programový kód aplikace a zpřehlednily funkčnost. Přesvědčil jsem se také, jak je dobré tyto vzory znát a rozumět jim, abych nemusel vymýšlet již jednou nalezenou věc.

Také jsem si procvičil programování v Android SDK. O prázdninách vyšla nová verze API s číslem 16, která pokračuje ve sjednocování dvou vývojových větví Androidu. Také přidává nové možnosti zobrazení hlavně v podobě nových layoutů. Tyto layouty jsou použitelné zvláště na moderních zařízeních s velkou obrazovkou.

Myslím, že jsem aplikaci navrhl podle zadání a splňuje základní požadavky, které na ni jsou kladeny. Aplikaci chci dále rozvíjet a již teď mám několik nápadů, které chci zahrnout do dalších verzí. Jedním z nich je vybudování online databáze pluginů, do které budou moci uživatelé nahrávat vlastní vytvořené pluginy a budou moci hodnotit pluginy ostatních uživatelů. S online databází se také pojí podpora v aplikaci a možnost stahovat pluginy přímo z internetu.

Dále chci vylepšit XML definice pluginů a přidat podporu psaní bashových funkcí. Nyní je možné napsat funkci přímo v tagu XML, ale určitě se najde lepší způsob, jak dále rozšiřovat pluginy a jejich schopnosti.

Jako poslední nápad je možnost jednoduše přidávat nová grafická rozhraní, která se budou ukazovat uživateli při ovládání pluginu. Tato funkcionalita se asi neobejde bez nutnosti znalosti programování v Androidu. Uvažuji nad možností použití instalovatelných pluginů přímo z obchodu Google Play, ale konkrétní návrhy zatím nemám.

Protože je Android velmi rozšířený a oblíbený systém i mezi uživateli Linuxu, domnívám se, že pro svou aplikaci najdu množství uživatelů. Značná část uživatelů Linuxu jsou i programátoři, proto doufám, že se k vývoji přidají i další programátoři, kteří budou chtít pracovat na vylepšování aplikace se mnou. Slibuji si od toho nové nápady na vylepšení a hlavně rychlejší vývoj celé aplikace. Protože se v Linuxu dá ovládat z příkazové řádky téměř všechno, je zde potenciál na napsání množství nových pluginů, ale právě díky takovému množství aplikací je napsání tolika pluginů nad moje síly.

# 5 Použitá literatura


1. UI Overview. GOOGLE INC. *Android developers* [online]. [cit. 2012-08-27]. Dostupné z: http://developer.android.com/design/get-started/ui-overview.html
2. Support Library. GOOGLE INC. *Android developers* [online]. [cit. 2012-08-27]. Dostupné z: http://developer.android.com/tools/extras/support-library.html
3. Action Bar. GOOGLE INC. *Android developers* [online]. [cit. 2012-08-27]. Dostupné z: http://developer.android.com/design/patterns/actionbar.html
4. Preference. GOOGLE INC. *Android developers* [online]. [cit. 2012-08-28]. Dostupné z: http://developer.android.com/reference/android/preference/Preference.html
5. Context (openFileInput). GOOGLE INC. *Android developers* [online]. [cit. 2012-08-30]. Dostupné z: http://developer.android.com/reference/android/content/Context.html#openFileInput%28java.lang.String%29
6. JSch. JCRAFT INC. *JCraft* [online]. [cit. 2012-08-30]. Dostupné z: http://www.jcraft.com/jsch/