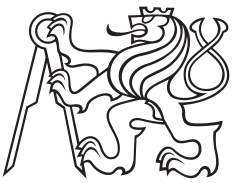
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra ekonomiky, manažerství a humanitních věd

Aspektově orientovaný vývoj kolaborativních Android aplikací využívající kontext

David Šindelář

Studijní program: Softwarové technologie a management
Obor: Manažerská informatika

Květen 2016
Vedoucí práce: Jiří Šebek

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra ekonomiky, manažerství a humanitních věd

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Šindelář David**

Studijní program: Softwarové technologie a management
Obor: Manažerská informatika

Název tématu:

Aspektově orientovaný vývoj kolaborativních Android aplikací využívající kontext

Pokyny pro vypracování:
1.Prostudujte existující nástroje pro podporu aspektově orientovaného programování (AOP).
2. Seznamte se s vývojem aplikací pro mobilní zařízení s platformou Android.
3. Zpracujte rešerši ohledně vývoje kolaborativních aplikací.
4. Navrhněte a implementujte framework, pomocí kterého se snadno budou vyvíjet kolaborativní aplikace pro Android.
5. Zohledněte AOP paradigma při návrhu frameworku.
6. Výsledná implementace musí být snadno rozšiřitelná.
Implementaci otestujte na demonstrační aplikaci.
7. Zhodnoťte výhody a možná omezení řešení.

Seznam odborné literatury:
1. ŠEBEK, J. and K. RICHTA. Aspect-oriented User Interface Design for Android Applications [online]. In: DATESO 2015. Databases, Texts, Specifications, and Objects 2015, Nepřívěc u Sobotky, Jičín, 2015-04-14/2015-04-16. Praha: MATFYZPRESS, vydavatelství Matematicko-fyzikální fakulty UK, 2015, pp. 121-130. CEUR Workshop Proceedings. vol. 1343.. Available from: http://www.cs.vsb.cz/dateso/2015/
2. Oficiální dokumentace: http://developer.android.com
3. ŠEBEK, J., M. TRNKA, and T. ČERNÝ. On Aspect-Oriented Programming in Adaptive User Interfaces. In: Proceedings of the 2nd International Conference on Information Science and Security. The 2nd International Conference on Information Science and Security, Seoul, 2015-12-14/2015-12-16. Piscataway: IEEE, 2015, pp. 147-151.

Vedoucí bakalářské práce: Ing. Jiří Šebek

Platnost zadání: do konce letního semestru 2016/2017

L.S.

Prof.Ing. Jaroslav Knápek, CSc.
vedoucí katedry

Prof.Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10.2.2016

Poděkování / Prohlášení

Rád bych poděkoval vedoucímu bakalářské práce Ing. Jiřímu Šebkovi za jeho rady a úpravy, které byly velmi podnětné.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 27. 5. 2016

.....

Abstrakt / Abstract

Tato bakalářská práce se zabývá vývojem efektivního kolaborativního frameworku pro mobilní systém Android, díky kterému mohou vývojáři velmi jednoduše vytvářet aplikace v krátkém časovém intervalu. Současné systémy, zabývající se stejnou problematikou, jsou zaměřené převážně na online kolaboraci, tedy přes internet. Tato práce navrhuje řešení jak pro online kolaboraci, tak i pro offline kolaboraci. Výsledkem práce je framework, ve kterém je dané řešení implementováno.

Klíčová slova: OS Android, Framework, Kolaborativní aplikace, Aspektově orientovaný přístup

My bachelor's thesis deals with the progression of effective collaborative framework for mobile system Android. Thanks to this framework developers create new applications. These applications are created easier and faster in comparison with other frameworks. Present systems, which deal with the same issues, are focused on online collaboration only. My bachelor thesis suggests solution for online collaboration, but also for offline collaboration. Finally, my bachelor thesis shows framework, in which is implemented the solution.

Keywords: aspect oriented approach, aspect driven design, entity inspection based approach, runtime aspect model, reduced maintenance and development efforts

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Hlavním cílem této bakalářské práce je vyvinout framework, který umožní vývojářům ulehčit a zefektivnit psaní kolaborativních Android aplikací. Programátor se tak nemusí starat o síťové propojení zařízení a nemusí se starat ani o to, jak odesílat objekty. Framework se o vše stará za něj.

V klasické konvenční aplikaci se nevyhneme duplicitám kódu např. pro logování a zabezpečení, které se musí řešit v každé části programu. Aspektově orientované programování přináší do vývoje aspekty, které nás těchto duplicit zbaví, zpřehlední kód a zajistí lepší znovupoužitelnost kódu.

První část je zaměřena na seznámení se základními pojmy využívanými v této bakalářské práci. Dále se v této části seznámíme s operačním systémem Android.

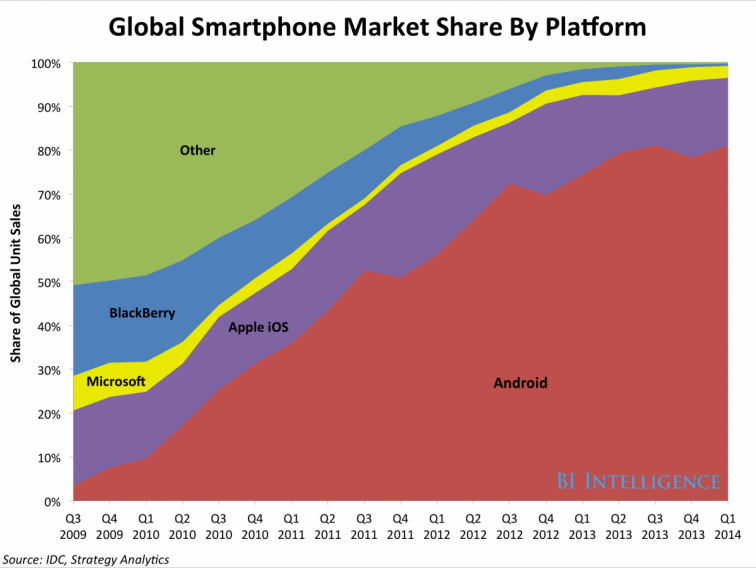
V následující části se seznámíme s konkurenčními řešeními.

Další část se zabývá analýzou, konkrétním řešením a požadavky na framework. Dané diagramy byly vytvořeny nástrojem Enterprise Architect.

V poslední části se porovnává konvenční a kolaborativní aplikace a vysvětluje se použití a instalace frameworku.

Kapitola 2
Rešerše

Na základě analýzy různých operačních systémů, jsem se rozhodl zaměřit na mobilní operační systém Android. Na obr. 2.1. si lže povšimnou, že operační systém Android je nejvíce využívaným operačním systémem na světě a jeho popularita a rozšiřitelnost stále roste. Aktuálně se Android nachází ve smartphonech[1], tabletech a nyní dokonce přechází i na notebooky a počítače.



Obrázek 2.1. Podíl mobilních operačních systémů na trhu od roku 2009 do roku 2014 [1]

2.1 Duplicity v kódu

I když se v dnešní době dbá na oddělení jednotlivých vrstev aplikace od sebe a využívá se správných principů, postupů a návrhových vzorů, stále se v kódu vyskytnou části, které se opakují. Tyto části jsou označovány jako duplicity v kódu. [2]

2.1.1 Odstranění duplicit

Existuje několik možností, jak se těchto duplicit zbavit. První z možností je využít principu Model driven architecture neboli Modelem řízená architektura (MDA). Jak název napovídá, základem této architektury jsou modely, například UML. Nejdříve se tedy vytvoří modely a z těch se již vygeneruje kód aplikace specifický pro určitou platformu. Další možností je využít princip aspektově orientovaného programování (AOP), o kterém se dočtete níže v kapitole 2.2.

[1] Mobilní telefon s OS.

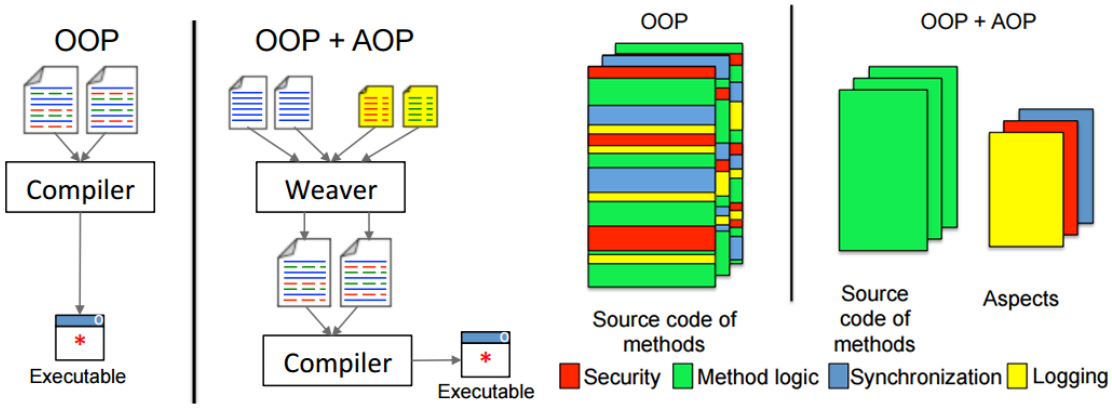
2.2 Aspektově orientovaný vývoj

Aspektově orientované programování (AOP) je programovací paradigma, tak jako Objektově orientované programování (OOP)[3]. Hlavním cílem AOP je snížení duplicit kódu a modularizování těchto duplicit do aspektů, viz obrázek 2.2., levá část.

Typickým příkladem je logování. V OOP, každá vytvořená třída, na kterou se má logování uplatnit, musí v každé své metodě zavolat tuto funkci logování. Vyvíjený kód se stává nepřehledný a přibývá mnoho duplicit. Pokud tedy budeme chtít udělat pozdější změnu funkce, budeme ji muset upravovat na všech těchto místech. AOP nám umožní dané logování nadefinovat v aspektu a při kompilaci kódu se vloží do všech metod automaticky viz obrázek 2.2., v pravé části.

Za nejznámější framework aktuálně považuji AspectJ, jehož vývoj v současnosti probíhá pod společností Eclipse Foundation. AspectJ má největší možnost definice pointcutů z aktuálně známých frameworků. Podporuje všechny typy aspektového proplétání až na proplétání za běhu programu. AspectJ upravuje i kompilátor, konkrétně jej rozšiřuje o nová klíčová slova.

Dalším velmi populárním a známým frameworkem je AspectFaces. Tento framework je zaměřen hlavně na uživatelské rozhraní. Ve frameworku AspectFaces se převážně využívají anotace u entit datového modelu. AspectFaces definuje vlastní sadu anotací.



Obrázek 2.2. Popis funkce AOP frameworku [4]

2.3 Kolaborativní systémy

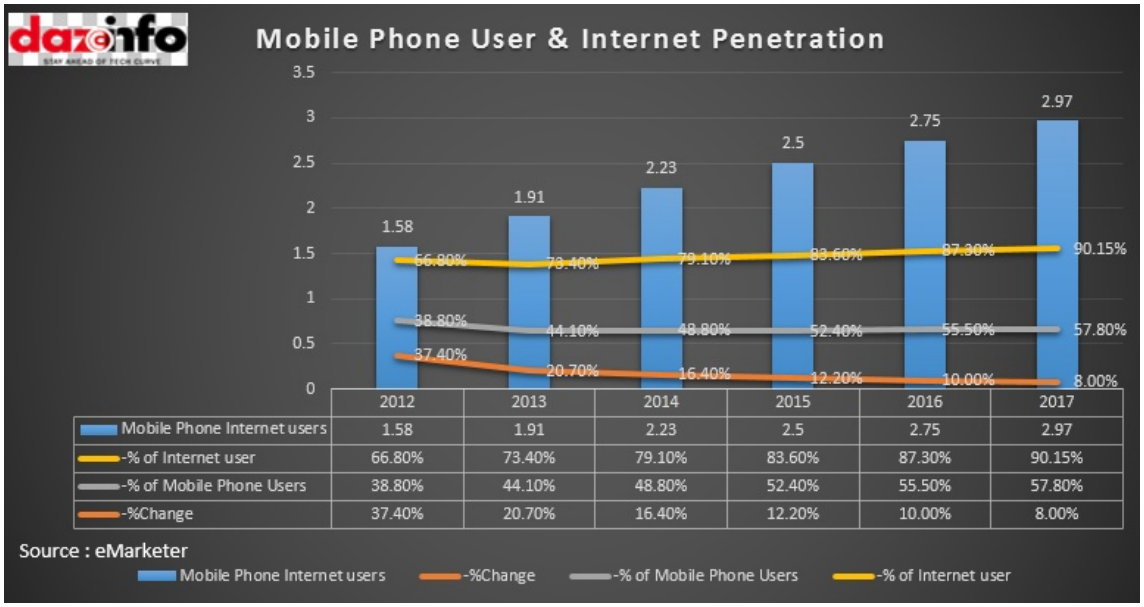
Kolaborativní systémy jsou určeny pro jeden základní účel, aby pomohl sjednotit lidi, kteří pracují na podobném nebo stejném úkolu a umožnit jim snadnější a efektivnější spolupráci. Jedná se o velmi široké téma počínaje správou projektů až po sdílení společných dokumentů či firemní sociální sítě.

2.4 Distribuované systémy

Distribuovaný systém lze definovat jako kolekci nezávislých počítačů, která se jeví svým uživatelům jako jediný systém. Nejznámějším příkladem distribuovaného systému je internet.

2.5 Využití internetu mobilními uživateli

V roce 2012 kolem 1,58 bilionu uživatelů využívalo jejich mobilní telefon k přístupu na internet, což tvoří zhruba 67% celkových internetových uživatelů. V roce 2014 to již bylo 2,23 bilionů uživatelů, což činí 79% internetových uživatelů. Z grafu lze vyčíst nárůst o celých 12,3%.



Obrázek 2.3. Graf zobrazující využití internetu v mobilních telefonech [5]

Jedním z hlavních úkolů vyvíjeného frameworku je zajistit komunikaci mezi zařízeními. Jedná se o zajištění stabilního propojení mezi zařízeními. Dle statistik na obrázku 2.3, stále přibývají uživatelé využívající internet ve svém mobilním telefonu. Z tohoto důvodu se v našem frameworku zaměříme na online propojení, avšak nebudeme mu věnovat hlavní pozornost, jelikož takovýchto řešení již několik existuje. Na druhou stranu stále je zde stále mnoho uživatelů, kteří internet v telefonu nevyužívají, ať už kvůli cenové dostupnosti anebo kvůli například vyčerpanému FUP limitu, po kterém se internet v telefonu opravdu na plno využívat nedá. Na tuto offline situaci se v tomto frameworku zaměříme a detailně popíšeme.

2.6 Online propojení zařízení

Online řešení představuje propojení zařízení přes internet. Při tomto připojení je potřeba, aby zařízení byla v online módu, resp. připojena k internetu. V tomto online módu je zařízení připojeno k serveru, přes který komunikuje s ostatními zařízeními. Hlavní výhodou tohoto připojení je, že zařízení spolu mohou komunikovat nezávisle na vzdálenosti. Takže není problém, aby byla zařízení spolu propojena, i přestože každé je na jiné straně zeměkoule. Hlavní nevýhodou je potřeba třetího zařízení (serveru), který tyto zařízení propojí, jelikož není možné, aby každé zařízení mělo veřejnou IP adresu, která je při komunikaci přes internet velmi důležitá.

2.7 Offline propojení zařízení

Opakem online řešením popsaným v kapitole 2.6 bude framework podporovat i offline řešení. Toto řešení bude podporovat technologii Bluetooth (BT) a Bezdrátovou síť

(WLAN). Oproti online řešení není v tomto případě zapotřebí třetího zařízení, které představuje server propojující zařízení. Zařízení jsou spolu propojeny přímo, což má jednu jasnou výhodu a tou je rychlost a bezpečnost.

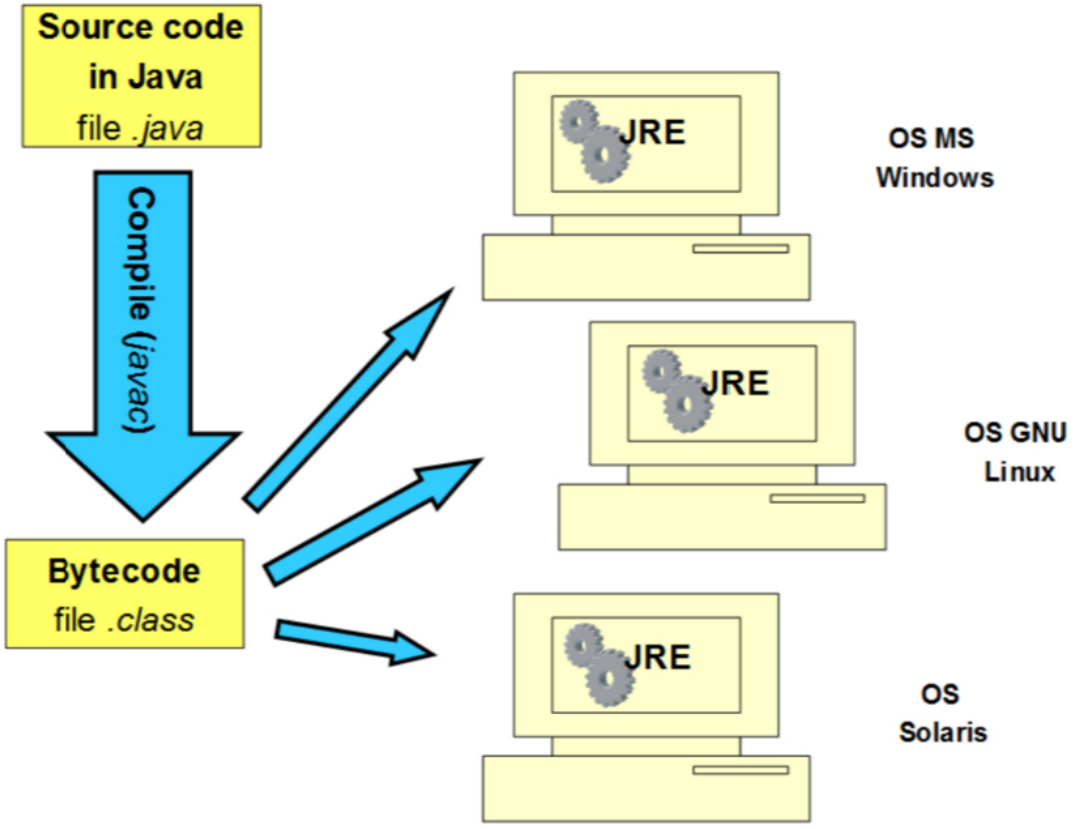
Kapitola 3
Android OS

3.1 Aplikace vyvíjené v Javě pro Android

Aplikace vyvíjené v jazyce Java mají základní neměnné kroky:

1. Napsat aplikaci v jazyce Java
2. Zkompilovat kód
3. Spustit program

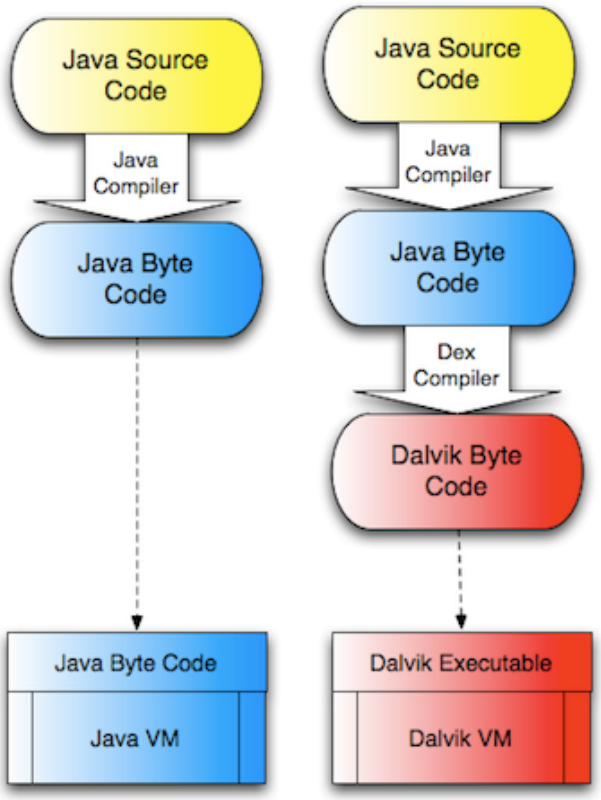
V prvním kroku je potřeba napsat jakoukoliv aplikaci v jazyce Java a uložit ji do souboru s příponou .java. Daný soubor zkompilujeme pomocí kompileru. Kompiler je zodpovědný za překlad java kódu do Bytekódu. Kompiler vytvoří z tohoto souboru nový soubor s příponou.class. Class soubor je již spustitelný na jakémkoliv zařízení nezávisle na platformě. Jediný požadavek je mít na zařízení Java Runtime Environment (JRE), obsahující Java Virtual Machine (JVM). Právě JVM se stará o spuštění .class souboru.



Obrázek 3.1. Kompilace Java kódu [6]

Aplikace vyvíjené v Javě pro Android mají velmi podobné kroky jako klasická Java. Hlavní rozdíl nastává při kompilaci spouštění aplikace. Kompilace začíná stejným procesem, soubor s příponou .java je zkompilován na soubor s příponou .class, avšak tím

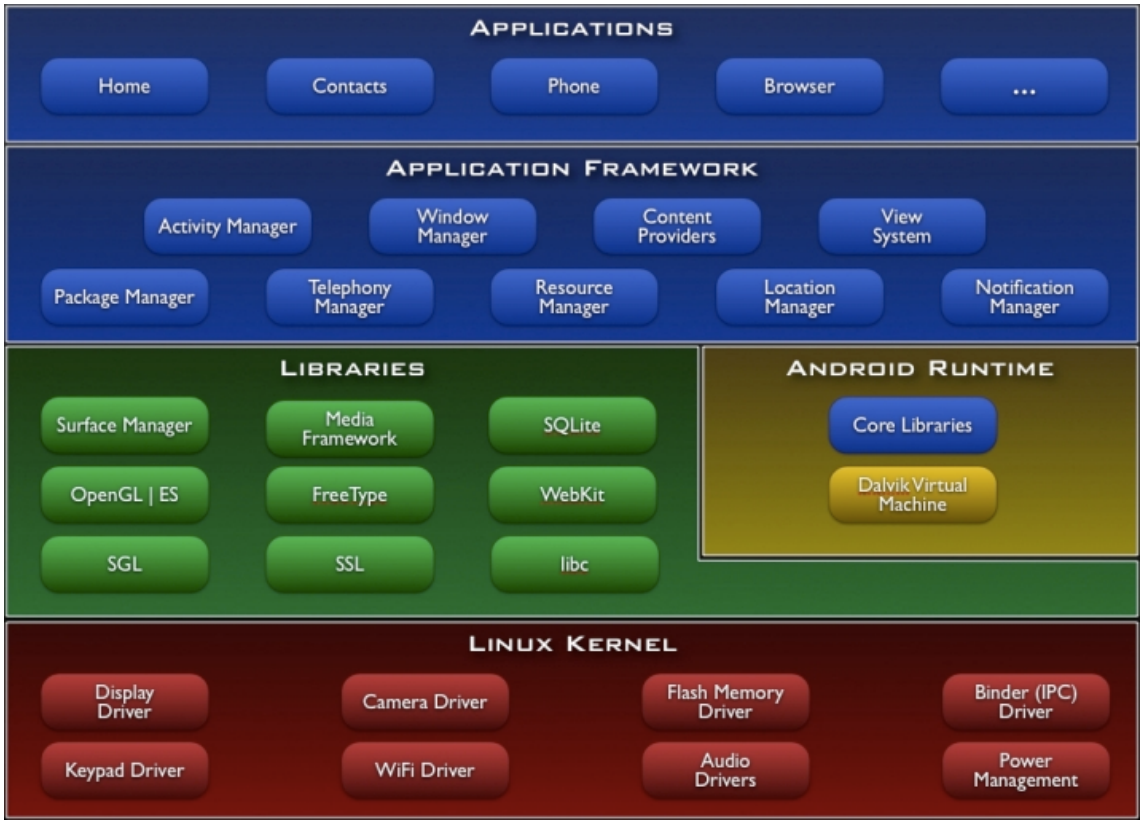
kompilace nekončí. Tento vygenerovaný soubor s Bytekódem je dále kompilován do souboru s příponou .dex obsahující Dalvik bytekód. Tento soubor lze spouštět na tzv. Dalvik Virtual Machine (DVM), který je od JVM odlišný. Tento rozdíl lze vidět na obrázku 3.2.



Obrázek 3.2. Obrázek znázorňující rozdíl mezi klasickým Java programováním a Java Android Programováním. [7]

3.2 Architektura OS Android

Android je mobilní operační systém běžící na jádře Linuxu a dostupný jako open-source.



Obrázek 3.3. Obrázek architektury OS Android Programováním. [8]

Na nejnižší základní vrstvě se nachází linuxové jádro (Linux kernel). Toto jádro se stará o veškerou komunikaci mezi hardwarem zařízení a softwarem programu. Obsahuje všechny důležité ovladače hardwaru jako klávesnice, displej, wifi ovladač, atd.

Na vrcholu linuxového jádra nalezneme knihovny (Libraries) starající se o základní funkce systému, které jsou napsány v programovacím jazyku C/C++. Mezi tyto základní funkce řadíme zobrazování aplikací a jejich vrstvení, práce s mediálními soubory, ukládání dat a práce s nimi, vykreslování webových stránek.

Za zmínku stoji Dalvik Virtual Machine (DVM). DVM se stará o překlad JAVA kódu do android instrukcí. Oproti známé Java Virtual Machine (JVM) je překladač DVM založený na práci s registry, čímž umožňuje lepší správu paměti.

Nejdůležitější vrstvou pro programátora je vrstva Aplikační framework (Application framework). Tato vrstva poskytuje několik služeb vyšší úrovně napsané v programovacím jazyce Java. Programátor již tyto služby využívá a používá je ve svém programu. Mezi klíčové služby patří Activity Manager (Řídí všechny aspekty životního cyklu aktivity a zásobníku aktiviy), Content Providers (Umožňuje aplikacím publikovat a sdílet data s jinými aplikacemi), Resource Manager (Poskytuje přístup k uloženým zdrojům jako soubory, UI layout, barvy, Stringy), Notifications Manager (Umožnuje aplikaci zobrazovat upozornění) a View system (Rozšířitelný soubor views k vytvoření uživatelských rozhraní).

Aplikace (Applications) je nejvyšší vrstvou architektury. Jedná se o již napsané aplikace, námi nebo jinými programátory. Příkladem takových aplikací jsou Kontakty, Prohlížeč, Hry, atd.

3.3 Životní cyklus android aktivity

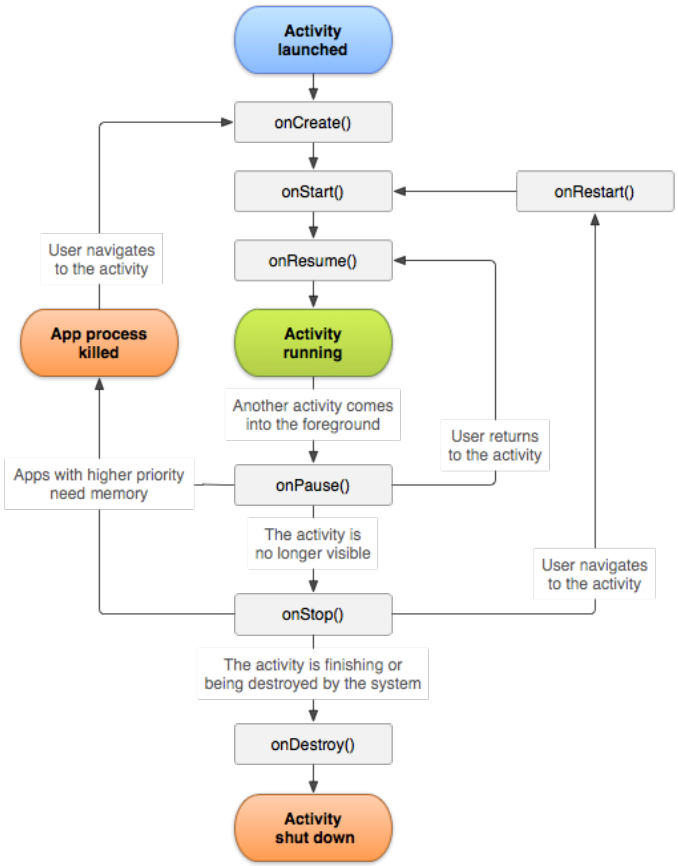
Nejdůležitější částí v Androidu je tzv. aktivita. Skoro všechny aktivity se starají o vytvoření okna, v kterém lze nastavit uživatelské rozhraní. Každá aktivita se řídí podle životního cyklu, viz obrázek 3.4.

Když se spustí nová aktivita, je umístěna na vrchol zásobníku a stává se z ní běžící aktivita. Předchozí aktivita zůstává pod novou aktivitou a nedostane se do popředí, dokud aktuální běžící aktivita existuje.

Každá aktivita má 4 základní stavy.

1. Aktivní - Tento stav nastane, když se aktivita dostane na popředí.
2. Pozastavena - Pokud zařízení přejde do režimu spánku, nebo pokud je aktivita částečně překryta, přejde do stavu pozastavena. Pozastavené aktivity jsou stále živé, takže si udržují všechny svoje informace.
3. Zastavena - Zastavené aktivity jsou kompletně zakryté jinou aktivitou, nebo jsou na pozadí. Zastavené aktivity jsou považovány za nejmenší prioritu.
4. Ukončena - Ukončené aktivity již v aplikaci ani v paměti neexistují.

Následující diagram zobrazuje důležité stavy Android aktivity. Čtvercové obdélníky představují zpětné volání metody, které lze implementovat pro vlastní obsluhu jednotlivých akcí při změně stavu. Tato zpětná volání je nutné znát, jinak není možné správně naprogramovat android aplikaci. Barevné ovály vyobrazují hlavní stavy, v kterých se aktivita může nacházet.



Obrázek 3.4. Obrázek popisující životní cyklus aktivity [6]

Metoda	Popis akce
onCreate	Toto je první zpětná metoda, která je volána jako první, kdy je aktivita vytvořena.
onStart	Tato metoda je volána když se aktivita zviditelní uživatelům.
onResume	Tato metoda je volána, když uživatel začne s aktivitou pracovat.
onPause	Pozastavená aktivita nemůže přijímat uživatelský vstup a nemůže vykonávat jakýkoliv kód.
onStop	Tato metoda je volána v případě, kdy se aktivita stává skrytou a není již uživatelem viditelná.
onDestroy	Tato metoda je zavolána, předtím než systém aktivitu odstraní.
onRestart	Tato metoda je volána v případě kdy se aktivita restartuje po zastavení.

Tabulka 3.1. Popis jednotlivých metod životního cyklu

Kapitola 4
Konkurenční řešení

Po podrobné analýze konkurence považuji za užitečné následující řešení:

- Google Realtime API
- Realtime Framework
- SocketCluster

4.0.1 Google Realtime API

Společnost Google není potřeba představovat, avšak s Google Realtime API[1] osobní zkušenost nemám. Po prostudování funkčnosti tohoto API jsem zjistil, že je poskytován pouze jako Javascript knihovna. Z toho lze soudit, že je určen primárně pro webové technologie.

Jeho architektura je založena na základním principu klient-server viz kapitola 5.1.1. Data jsou uložena na serveru, která jsou sdílena se všemi připojenými klienty. K přenosu dat využívá Operational transformation [9].

Hlavní výhoda tohoto řešení je samotná společnost Google. Ve společnosti Google jsou stovky kvalitních programátorů, kteří se této technologii věnovali a vyladili do detailů.

Avšak je zde jedna hlavní nevýhoda a tou je centralizace dat. Technologie vyžaduje přítomnost serveru, na kterém jsou data uložena.

4.0.2 Realtime Framework

Realtime Framework [10] poskytuje Messaging system. Poskytuje i cloud technologii, přes kterou jsou jednotlivé zprávy odesílány.

Hlavní výhoda tohoto frameworku je podpora většiny programovacích jazyků. Framework pro každý programovací jazyk poskytuje vlastní SDK (Software Development Kit), do kterého stačí nastavit application key, který se obdrží při registraci a lze odesílat zprávy.

Nevýhodou tohoto připojení je závislost na internetu. Odesílaná zpráva se ze zařízení odešle na cloud server Realtime Frameworku, odkud se již dále přepošle všem ostatním zařízením, která aktuálně kolaborují.

4.0.3 SocketCluster

Jedná se o open-source framework[2], určený pro Node.js[3]. Podporuje přímou klient-server komunikaci a hromadnou komunikaci (pub/sub channels).

SocketCluster je poskytován jako běžící služba na několika zařízeních, čímž zařizuje rychlou škálovatelnost.

Avšak ani SocketCluster nedokáže kolaborovat v offline režimu. Neustále je potřeba připojení k internetu.

[1] https://developers.google.com/google-apps/realtime/overview
[2] http://socketcluster.io/
[3] https://nodejs.org/en/

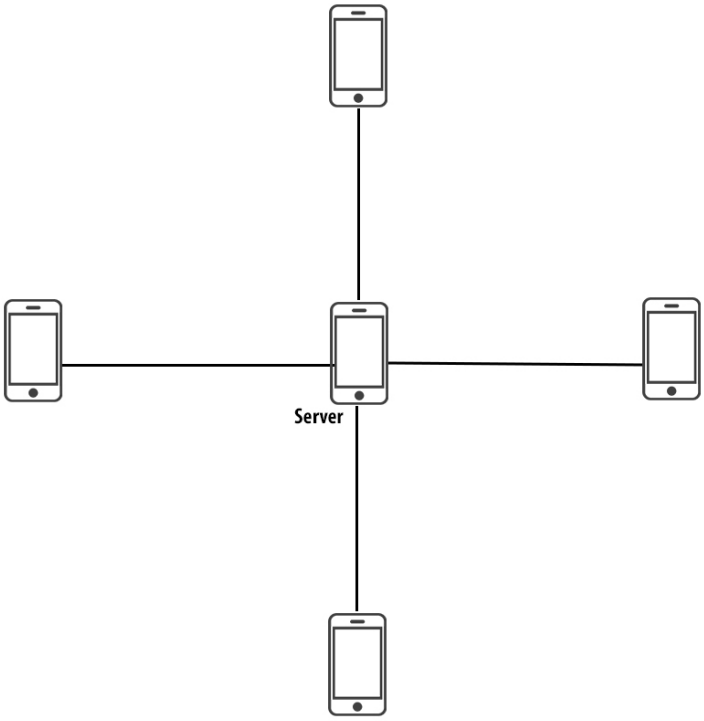
Kapitola 5
Analýza a design řešení

5.1 Architektury řešení

Framework řeší dvě různá připojení. První připojení online je přes internet a druhé je offline (Interní wifi a bluetooth). K připojení můžeme použít tyto architektury:

- Klient-server
- Full-mesh
- Zapojení do kruhu

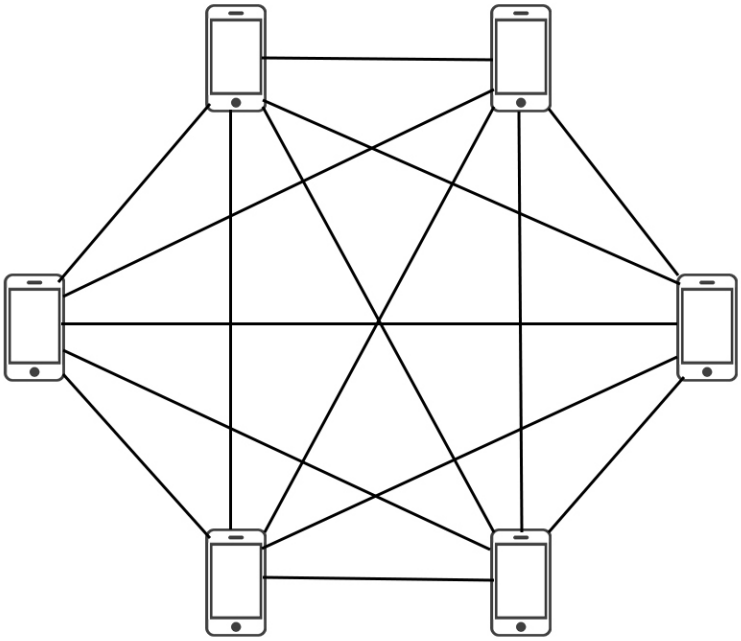
5.1.1 Klient-server (Zapojení do hvězdy)



Obrázek 5.1. Architektura klient-server

Jedná se o nejčastější typ připojení. Jak si lze povšimnou na obrázku 5.1., v centru je server, ke kterému se připojují jednotlivá zařízení (klienti). Velká nevýhoda této architektury je postavení serveru. Server musí zpracovávat požadavky všech klientů a to pro náš vyvíjený framework není vhodné, jelikož server by musel být vždy výkonný, aby tato propojení zvládl. Další velkou nevýhodou je odpojení serveru. Pokud se server odpojí, všichni klienti ztratí spojení a tím komunikace přestane fungovat.

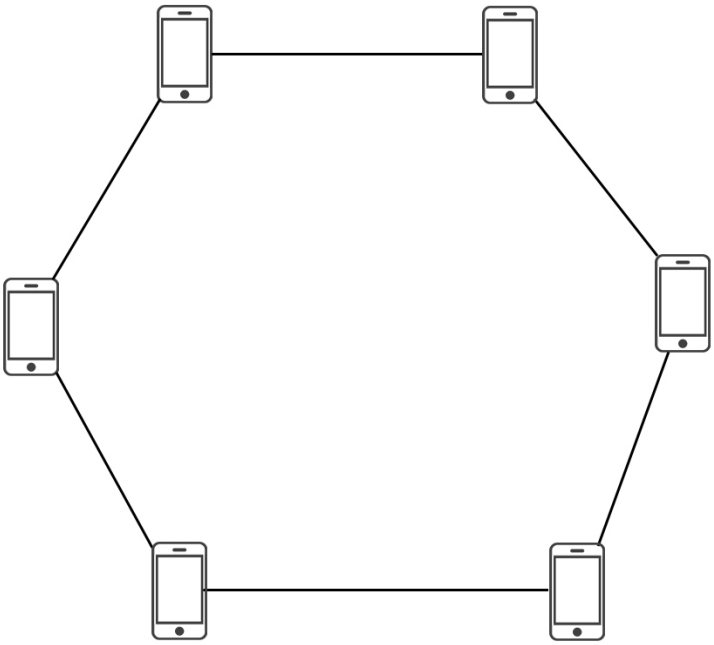
5.1.2 Full mesh zapojení



Obrázek 5.2. Architektura full mesh

Full mesh je typ zapojení, kdy každé zařízení je propojené s každým dalším. Hlavní výhodou je přímé propojení s každým zařízením. Pokud dojde k jakékoliv změně, jedno zařízení dokáže zkontaktovat všechny zařízení najednou ve stejný čas. Hlavní nevýhodou této síťové architektury je velké využití sítě. Při každém nově připojeném zařízení přibyde n+1 spojení, kdy n znamená počet zařízení. V případě většího počtu klientů se síť zahltí a komunikace bude velmi pomalá.

5.1.3 Zapojení do kruhu



Obrázek 5.3. Architektura zapojení do kruhu

V dané architektuře je každé zařízení propojené s následujícím. Poslední připojené zařízení je připojené k prvnímu a tím vzniká kruh. Nevýhodou tohoto zapojeni je odpojení zařízení. Při odpojení zařízení se ztratí celkové propojení a tak posílaná data neobdrží všechna zařízení. Naopak hlavní výhodou je odesílání dat skrze všechna. Další výhodou je nezatížení sítě a zařízení.

5.2 Design řešení

5.2.1 Online řešení

Pro online propojení využijeme architekturu Klient-server, kdy server musí mít přiřazenou veřejnou IP adresu, aby se k němu zařízení mohla připojovat. Výhodou této architektury v online módu je potřeba pouze jedné veřejné IP adrese oproti ostatním architekturám zmíněných v kapitole 5.1.

5.2.2 Offline řešení

Pro offline připojení vybereme zapojení do kruhu. Toto zapojení nejvíce vyhovuje naším požadavkům, jelikož nezatěžuje zařízení ani provoz v síti, takže se dosáhne velmi rychlého průchodu sdílených dat.

5.3 Kolaborativní systémy

5.3.1 Komunikace mezi zařízeními

V tomto vyvíjeném frameworku je využíván komunikační protokol TCP. TCP protokol spadá do Transportní vrstvy v OSI modelu a je využíván k odesílání dat (paketů) na hlavní server, který funguje jako zprostředkovatel přes internet (online zapojení), nebo

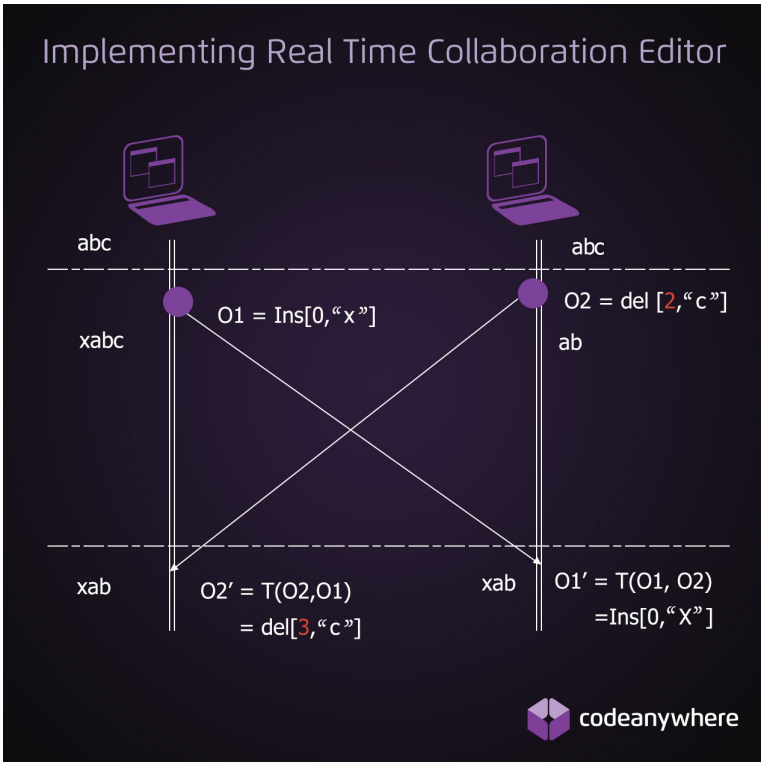
přímo na další zařízení (offline zapojení). TCP protokol pro komunikaci potřebuje nejprve vytvořit Socket. První zařízení si vytvoří tento Socket, a druhé se k němu připojí. Tímto propojením vznikne tzv. tunel, přes který zařízení již spolu mohou komunikovat. Výhoda v tomto TCP tunelu je zajištění pořadí dat, díky čemuž přijímací zařízení obdrží data v pořadí, v jakém byla odeslána. Tato volba TCP protokolu zajistí spolehlivost spojení a odesílání dat.

Jelikož Bluetooth nepodporuje TCP spojení, použijeme protokol RFCOMM. I přes to, že tento protokol byl navrhnut, aby emuloval RS-232 sériový port, je velmi podobný TCP a nabízí i stejné služby. Jediným zásadním rozdílem mezi těmito protokoly je rozsah portů. TCP protokol podporuje porty až do 65535, naproti tomu RFCOMM jen do 30. Tento protokol bude použit v Bluetooth režimu.

5.3.2 Sdílení mezi zařízeními

Jak je uvedeno v kapitole 5.3.1, zařízení po připojení vytvoří tunel, přes který spolu komunikují. V aktuální verzi frameworku spolu zařízení komunikují pomocí objektů rozšiřující objekt AbstractSendingObject. Tento objekt implementuje rozhraní Serializable, díky čemuž může Java při odesílání objekt serializovat a odeslat přes zmíněný protokol. Další zařízení, které tento objekt přijme, ho deserializuje a pracuje s ním jak je potřeba. Tento typ odesílání je vhodný, pokud se stále posílají nové objekty, například Chat mezi uživateli.

Pokud ale budeme pracovat na nějaké společné práci, např. sdíleném dokumentu, bylo by nevýhodné odesílat vždy celý dokument při jakékoliv změně. Zde ale nastává problém, jak by se měl framework zachovat, pokud změní dokument více uživatelů naráz. Každé zařízení by poslalo svůj upravený objekt a dokument by se změnil jen podle posledního přijatého objektu, které by zařízení obdrželo. V tomto případě je nutné použít odlišný typ sdílení dat. Pro tento typ použijeme technologii Operation tranformation (OT). Tato technologie je určena přímo pro problém kolaborativních aplikací, kdy se sdílí společná data, ve kterých dochází k částečným změnám. S touto technologií pracuje jak např. Apache Wave, tak i Google docs. Hlavní myšlenkou této technologie je posílat jen data, která se v aplikaci právě změnila, viz Obrázek 5.4.

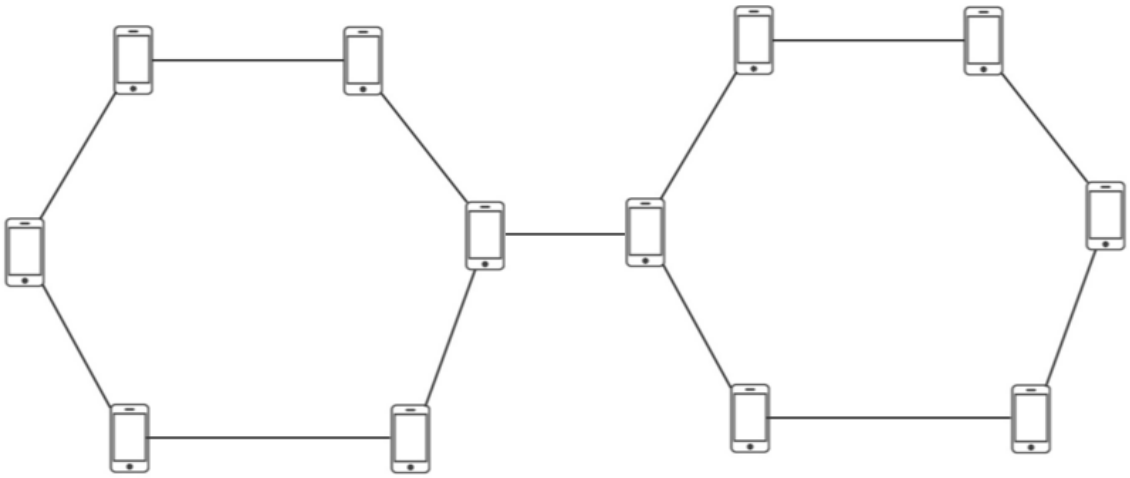


Obrázek 5.4. Operational transformation [9]

Na obrázku 5.4 je znázorněna funkčnost OT. Na počátku cyklu mají obě zařízení společný text abc. V jednom okamžiku zařízení provedou rozdílné změny. První zařízení (vlevo) vloží na začátek textu písmeno x. Druhé zařízení v ten samý okamžik vymaže písmeno c. U každé této operace (Vložit, Vymazat) si musíme pamatovat o jaké písmeno se jedná a na jakém indexu se nachází. Pokud by se neodesílalo v případě vymazat písmeno, nemohla by projít kontrola, zda opravdu mažeme správný znak, jelikož v tu chvíli než se dostal příkaz ke všem zařízením už mohlo dojít i k jiným změnám. Levé zařízení tedy má již text „xabc” a pravé zařízení „ab”. Levému zařízení nyní přišel požadavek na smazání písmene c na indexu 2. Tento požadavek ale musí být upraven vůči požadavku O1, takže se změní na požadavek smazat c na indexu 3. Výsledek tedy na levém zařízení je „xab” V ten samý okamžik přijalo druhé zařízení požadavek O1. Framework opět musí upravit požadavek vůči O2, který jej ale nezmění takže se na nultou pozici vloží znak X a výsledek na zařízení je „xab”. Obě zařízení mají tedy stejný text, a zatížení sítě je opravdu minimální.

5.4 Analýza problémů

5.4.1 Propojení dvou různých stromů



Obrázek 5.5. Propojení dvou různých stromů

Každé zařízení je omezeno pouze na max. 2 spojení, to znamená, že přijímání dat od předchozího zařízení je prvním spojením a druhé spojení je odesílání dat dalšímu zařízení v sítí. Díky tomuto přístupu není možné, aby se dvě různé sítě propojily a vznikalo tak větvení.

5.4.2 Odpojení zakladatele spojení

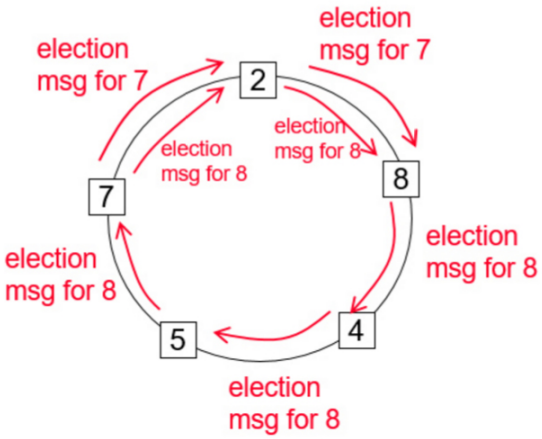
Zakladatel spojení je jediné zařízení, které může do vytvořené struktury přijímat další zařízení. V případě odpojení zařízení, není žádné zařízení, které by vysílalo do okolí, že tato vytvořená síť existuje, proto se spustí tzv. Leader election. viz kapitola 5.4.4.

5.4.3 Odeslání změny dat ve stejný čas

Tato situace je velmi nepravděpodobná. Při každé odeslané změně je u ní evidována časová značka s přesností nanosekund. Navíc každé zařízení má jiné parametry a tak i kdyby se uživatelé snažili provést operaci ve stejný čas, je velmi nepravděpodobné, že by se jim tato situace podařila. Proto tuto situaci ve frameworku řešit nebudeme a předpokládáme, že tato situace nenastane.

5.4.4 Leader election

V offline režimu je potřeba, aby jedno zařízení bylo stále viditelné pro nově příchozí zařízení. Tímto zařízením bude to, které server vytvoří. Co nastane, když se toto zařízení odpojí? Pak by vznikl uzavření kruh zařízení a nikdo jiný by se nemohl do kruhu nově připojit. V této situaci využijeme tzv. Leader election, konkrétně algoritmus Chang and Roberts leader election. Tento algoritmus počítá s tím, že každý procesor má své UID (unikátní id). Na začátku zařízení, které je připojeno v síti nejdéle vyšle election zprávu obsahující své UID. Další zařízení, které zprávu obdrží porovná toto UID se svým UID. Pokud přijaté UID je vetší, zprávu přepošle. Pokud přijaté UID je menší, nahradí UID svým UID a zprávu pošle dál. Pokud se přijaté UID rovná vlastnímu UID, zpráva obešla všechna zařízení a zařízení se stává leaderem. V této situaci, kdy se stane leaderem, se zařízení zviditelní a je možné do sítě připojit nové zařízení.



Obrázek 5.6. Obrázek znázorňující leader election

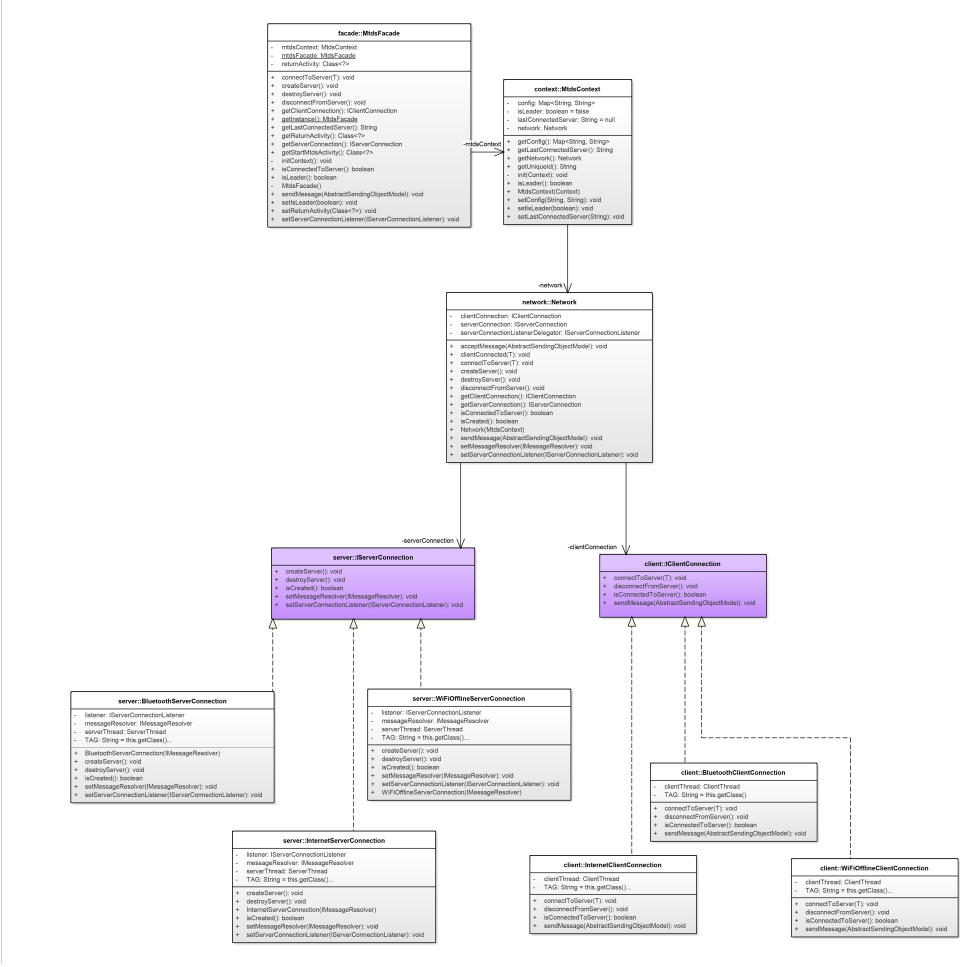
Na obrázku 5.6 je znázorněn příklad leader election. Zařízení s ID 7 vyšle leader election zprávu jako první. Zařízení 2 přečte zprávu, zjistí, že jeho ID je menší, tak zprávu jen přepošle dál. Zařízení 8 přečte zprávu a zjistí, že má větší ID. Tudíž zahodí tuto zprávu a vyšle novou leader election zprávu se svým ID. Jelikož zařízení s ID 8 je nejvyšší, tato zpráva projde celým kruhem znovu k zařízení 8. Toto zařízení zjistí, že dostal svoji vyslanou zprávu a tak se stává leaderem.

Případ	Časová složitost
Nejhorší případ	$O(n^2)$
Průměrný případ	$O(n * log(n))$
Nejlepší případ	$O(n)$

Tabulka 5.1. Časová složitost leader election

5.5 UML Diagram tříd

Základem vyvíjeného frameworku je třída MtdsFacade. Jedná se o třídu implementovanou pomocí návrhového vzoru Facade. Návrhový vzor Facade slouží ke zjednodušení komunikace se systémem. To znamená, že programátor, který bude využívat framework, bude komunikovat jen s touto třídou a nebude potřeba využívat jiné třídy.



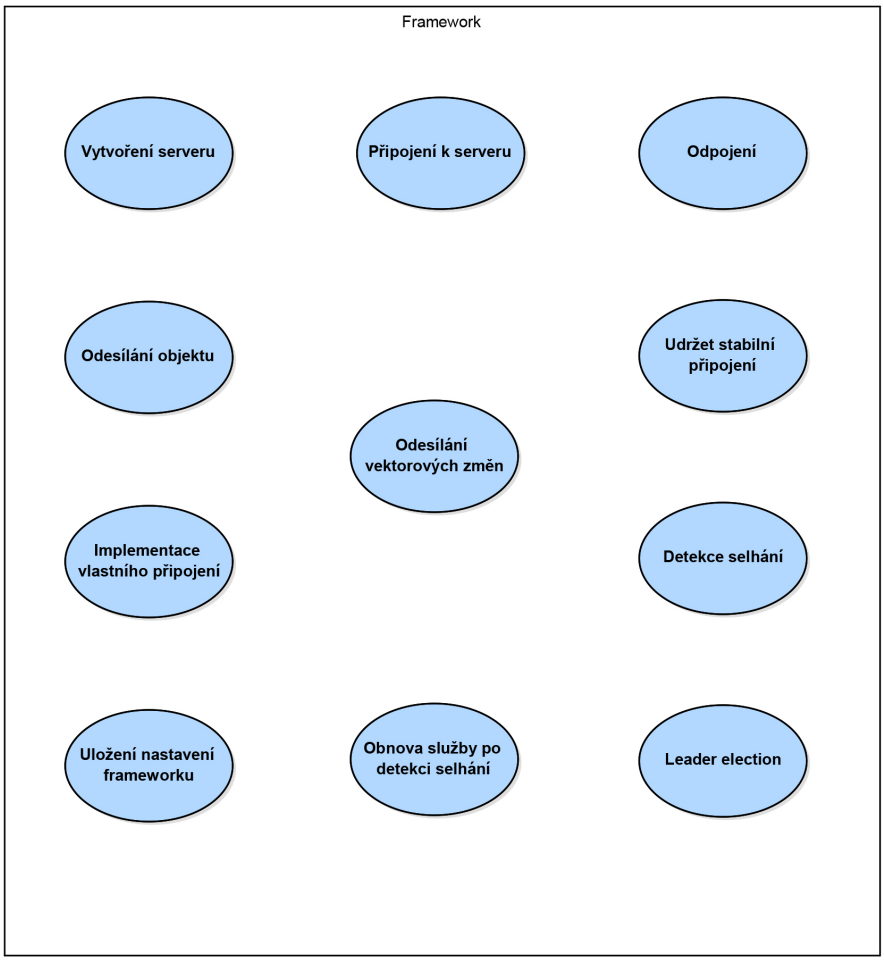
Obrázek 5.7. Náhled základního UML diagramu frameworku

Jednotlivé částí UML diagramu jsou podrobně rozebrány v následujících kapitolách.

Další důležitou třídou je třída MtdsContext. Tato třída si uchovává všechny důležité informace o aktuálním stavu, konkrétně si uchovává nastavení připojení. Také si v sobě uchovává instanci třídy Network. V této části diagramu se setkáme s návrhovým vzorem Strategy. Tento návrhový vzor se využívá v případě, kdy je potřeba za běhu programu vybrat správnou instanci objektu, v našem případě to bude atribut ServerConnection a ClientConnection, které se budou měnit na základě nastavení frameworku. Toto nastavení se ukládá přes Android třídu SharedPreferences. SharedPreferences je key-value uložiště, které pracuje jen s primitivními datovými typy, ale v našem frameworku více nepotřebujeme. Jak již z názvu vypovídá, uložiště je sdílené, takže je přístupné z jakékoliv aktivity. Objekt SharedPreferences se může získat metodou Context.getSharedPreferences(String name, int mode), kde name určuje daný preference objekt, takže jich můžeme mít v aplikaci více. Ve frameworku budeme využívat jednoduší volání přes PreferenceManager. Konkrétně zavoláme metodu PreferenceManager.getDefaultSharedPreferences(Context context), kdy dostaneme objekt s preferencemi společný pro celou aplikaci, díky čemuž nemusím řešit žádná jména, ani žádné módy.

5.6 USE CASE Diagram

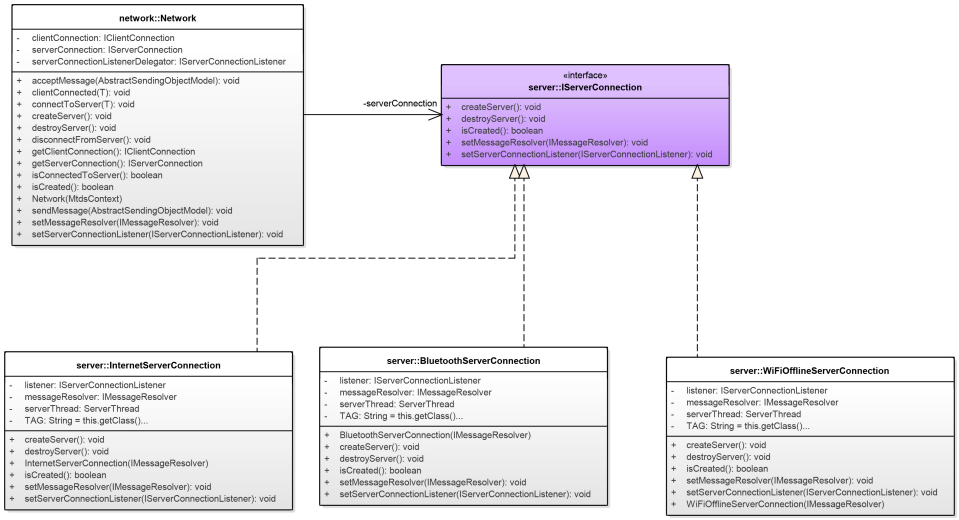
V následujícím use case diagramu je znázorněno, jaké všechny služby framework zajišťuje automaticky a o které se programátor využívající framework nemusí starat.



Obrázek 5.8. Diagram případu užití

5.6.1 Vytvoření serveru

Na základě nastavení frameworku se vytváří server. Jelikož framework podporuje několik typu připojení, zvolil jsem vhodnou implementaci pomocí návrhového vzoru Strategy

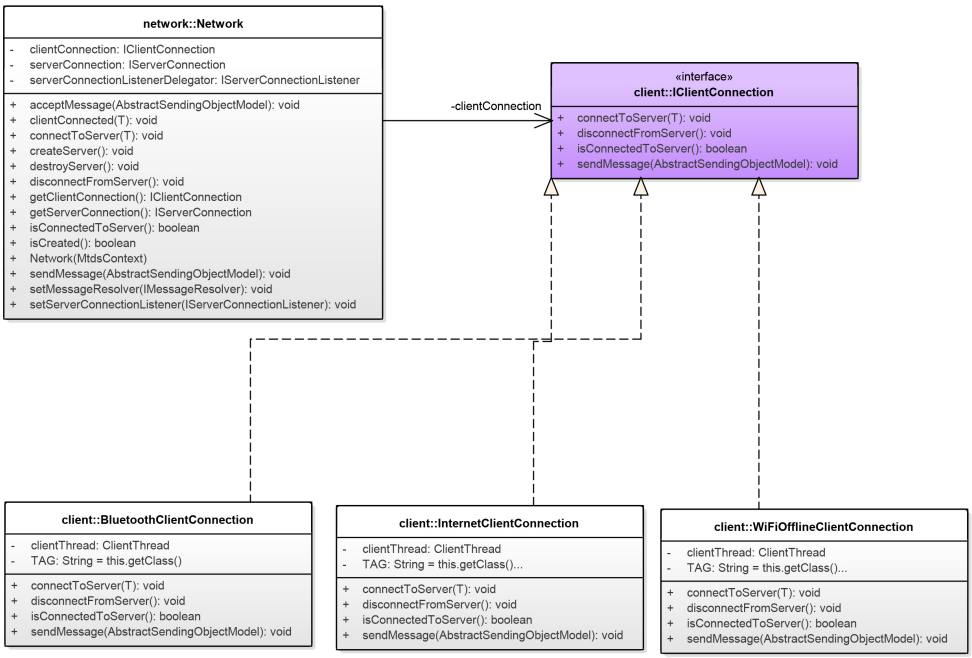


Obrázek 5.9. UML diagram server připojení

Na obrázku 5.9 vidíme, že třída Network představuje Kontext z hlediska návrhového vzoru Strategy. Uchovává v sobě atribut serverConnection s typem IServerConnection. Při inicializaci frameworku se vybere strategie (InternetServerConnection, BluetoothserverConnection, WiFiOfflineServerConnection) dle nastavení, a přiřadí se do atributu serverConnection. Ze třídy Network se poté příkazy createServer a destroyServer delegují do vybrané strategie, která už se stará o svůj druh připojení.

5.6.2 Připojení k serveru

Připojení k serveru funguje velmi podobně, respektive téměř stejně jako vytvoření serveru. Opět je využit návrhový vzor Strategy.



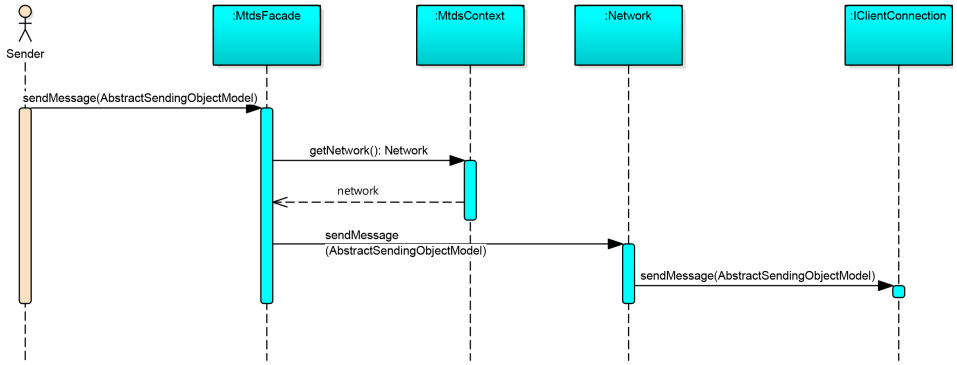
Obrázek 5.10. UML diagram client připojení

5.6.3 Odeslání objektu

Framework poskytuje odesílání jakéhokoli objektu rozšiřující objekt typu AbstractSendingObject. Programátor si tedy může odeslat jakýkoliv vlastní objekt, který projde celou sítí, až zpět k němu. Odeslání objektu funguje jednosměrně.

V offline režimu zařízení pošle objekt vždy dalšímu připojenému zařízení, což se vykonává v ClientConnection

V online režimu zařízení pošle objekt serveru, ke kterému je připojen, a ten již tuto zprávu rozešle všem dalším připojeným zařízení.



Obrázek 5.11. Sekvenční diagram odeslání objektu

Obrázek popisuje, jak odesílání objektu funguje. Programátor zavolá na MtdsFacade metodu sendMessage(AbstractSendingObjectModel object), která přijímá jakýkoliv objekt, rozšiřující třídu AbstractSendingObjectModel. Tuto metodu deleguje do třídy Network a ta metodu deleguje do vybrané strategie připojení. Ta se již stará o odeslání do dalších zařízení.

5.6.4 Operational tranformation

Toto odesílání framework ve své aktuální verzi řeší jen částečně. Poskytuje objekt OperationalTransformationObject, který v sobě uchovává atributy, o jakou operaci se jedná, index provedené změny a konkrétní změnu. Programátor tedy vytvoří tento objekt, a pošle ho přes metodu sendMessage ve třídě MtdsFacade. Operational tranformation by měl uživatel používat v případě, kdy uživatelé kolaborují na nějakém společném dokumentu, jelikož u tohoto dokumentu by byl nesmysl po každé jedné změně posílat celý dokument. Framework, který přijme objekt automaticky rozpozná, že se jedná o objekt typu OperationalTransformationObject a zavolá se vlastní metoda visit(OperationalTransformationObject o) v MessageResolveru, který již tento objekt zpracuje a dále pošle programátorovi přes listener. Více o této technologii se dočtete v kapitole 5.3.2.

5.6.5 Uložení nastavení frameworku

Framework k ukládání nastavení používá třídu z Androidu SharedPreferences. Android automatický zajistí uložení tohoto nastavení do xml souboru, takže po vypnutí a zapnutí aplikace zůstane toto nastavení zachováno. Díky přenechání zodpovědnosti na Androidu je kód načtení nastavení opravdu krátký.

```
private Map<String, String> config;

public Map<String, String> getConfig() { return config; }

public void setConfig(String key, String value) { this.config.put(key,value); }

private void init(Context context){
    this.config = new HashMap<>();
    SharedPreferences sh = PreferenceManager.getDefaultSharedPreferences(context);
    String connection = sh.getString("connection","wifi");
    this.setConfig("connection",connection);
    network = new Network(this);
}
```

Obrázek 5.12. Obrázek popisuje, jak ve frameworku probíhá načtení nastavení.

5.6.6 Leader election

Leaderem je vždy zařízení, které vytváří spojení. Pokud se toto zařízení odpojí, je potřeba vybrat nového leadera. K tomuto výběru slouží třída LeaderElection, která se stará o odesílání leaderelection zpráv a jsou do ní delegovány zprávy z message resolveru.

```
public class LeaderEletion {

    public void doElection() {
        try {
            MtdsFacade.getInstance().sendMessage(new LeaderElectionObject());
        } catch (ClientNotConnectedException e) {
            e.printStackTrace();
        }
    }

    public void accept(LeaderElectionObject o) {
        if (o.getSender().equals(Build.MODEL)) {
            //zpravu odeslalo toto zarizeni.. bude leader
            MtdsFacade.getInstance().setIsLeader(true);
        } else {
            try {
                if (o.getUid() < this.getDeviceUid()) {
                    MtdsFacade.getInstance().sendMessage(new LeaderElectionObject());
                } else {
                    MtdsFacade.getInstance().sendMessage(o);
                }
            } catch (ClientNotConnectedException e) {
                e.printStackTrace();
            }

        }
    }
```

Obrázek 5.13. Obrázek znázorňuje část kódu, který řeší odeslání a vybrání leadera.

5.6.7 Detekce selhání

V zapojení do kruhu je důležité, aby všechna zařízení byla připojena. Pokud se jedno zařízení odpojí, kruh se rozpadne a komunikace přestane fungovat. Framework s touto situací počítá a má zavedenou službu udržet stabilní připojení (sekce 5.6.10), která toto odpojené zařízení detekuje.

5.6.8 Obnova služby po detekci selhání

Tato služba navazuje na službu 5.6.10, kdy je potřeba opravit rozpojené kruhové zapojení. Zařízení se nevalidně odpojilo a framework nemá dostatek informací jak problém vyřešit. V této situaci framework odpojí všechna svá aktuální připojení a začne se připojovat k serveru. Server tato připojení obdrží a postupně se opět vytvoří kruhové spojení, ve kterém opět všichni mohou kolaborovat.

5.6.9 Odpojení

Odpojení lze provést dvěma způsoby:

- Validní odpojení
- Nevalidní odpojení

V případě validního odpojení, kdy uživatel vypne aplikaci, vyšle framework při zavírání objekt typu DisconnectObject. V tomto objektu je uchována informace, kdo se právě odpojuje a ke komu toto zařízení bylo připojeno. Jakmile tuto zprávu přijme zařízení, které bylo k nyní již odpojenému připojené, změní své klient připojení k zařízení z přijatého objektu.

V případě nevalidního odpojení, kdy se uživatel odpojí, aniž by framework stihl odeslat DisconnectObject (vypne se Bluetooth, vybije se baterie, apod.), vznikne v síti prázdné místo a komunikace v tuto chvíli nefunguje. Tuto situaci, ale detekuje služba v sekci 5.6.10, která za tuto chybu přebírá zodpovědnost.

5.6.10 Udržet stabilní připojení

Aby framework zajistil stabilní připojení, potřebuje vědět, že všechna zařízení jsou připojena, a nevznikla v síti díra, kvůli které by spojení přestalo fungovat. Z tohoto důvodu framework pravidelně v intervalu odesílá objekty typu FailureDetectionObjekt. V případě, že mu zpráva dojde, znamená to, že všechna zařízení jsou online. Pokud zprávu neobdrží, bylo detekováno selhání a spustí se obnova služby po selhání, sekce 5.6.8.

5.6.11 Implementace vlastního připojení

V mnoha případech se stává, že programátor si chce napsat vlastní obsluhu připojení anebo vznikne nové připojení, které ještě v dnešní době neexistuje. Při vývoji frameworku se s touto možností počítalo a programátor může vytvořit vlastní typ připojení. Jediné co musí udělat je, že vytvoří nové připojení, které implementuje interface IServerConnection pro server a IClientConnection pro připojování k serveru. Následně tyto nové připojení musí zaregistrovat, aby se objevili v nastavení frameworku ve výběru připojení. To provede pomocí metody v MtdsFacade registerNewConnection(iServerConnection serverConnection, iClientConnection clientConnection). Framework si tuto registraci přidá do seznamu existujícího připojení a uživatel již může toto připojení vybrat.

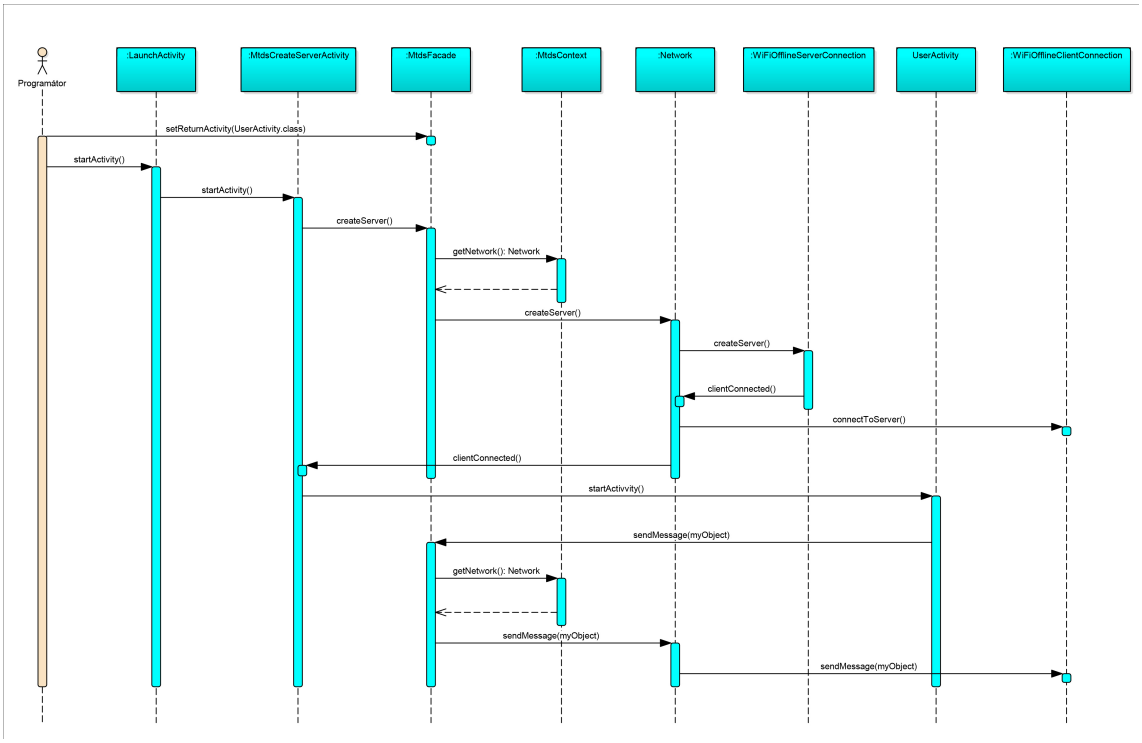
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main2);

    MtdsFacade.getInstance().setReturnClass(MainActivity.class);
    MtdsFacade.getInstance().registerNewConnection(new IrdaServerConnection(), new IrdaClientConnection());
    Intent intent = new Intent(this, MtdsActivity.class);
    startActivity(intent);
}
```

Obrázek 5.14. Obrázek znázorňující, jak přidat nové vlastní připojení do frameworku.

5.7 Sekvenční diagram

V následujícím sekvenčním diagramu je zobrazen proces, kdy uživatel zapne aplikaci, vytvoří server, ke kterému se připojí klient a následně je zapnutá aktivita, která již pracuje s odesíláním objektů. Celý proces začíná zapnutím aplikace. Po spuštění aplikace se ve frameworku nastaví, jaká aktivita po dokončení má být zavolána (pomocí metody setReturnActivity). Poté je spuštěna hlavní aktivita frameworku LaunchActivity, vybrána možnost vytvoření serveru, čímž se zapne aktivita MtdsCreateServerActivity. Tato aktivita volá ve třídě MtdsFacade metodu createServer, která je delegována až do třídy Network, odkud je dále delegována do serverConnection, v tomto případě do WifiOfflineServerConnection. Jakmile je server vytvořen, čeká se na připojení klienta. Po připojení klienta se zavolá ve třídě Network metoda clientConnected. V této metodě se zajistí následné připojení k připojenému klientovi, čímž vznikne kruhové zapojení o dvou zařízeních. Z této metody je zpráva o připojení klienta delegována až do MtdsCreateServerActivity, která spustí aktivitu, která se nadefinovala na začátku procesu (spustí se tedy UserActivity). Z této aktivity se již odesílají zprávy přes MtdsFacade, odkud se metoda deleguje do WifiClientConnection.



Obrázek 5.15. Sekvenční diagram aplikace.

Kapitola 6
Porovnání konvenční vs. kolaborativní aplikace

6.1 Konvenční přístup

Za předpokladu, že programátor bude celou aplikaci vyvíjet objektově orientovaným způsobem, bude muset obstarat všechna síťová připojení, stabilitu a spolehlivost. Pokud by programátor programoval jednoduchou chat aplikaci, obsahující kruhovou topologii s leader election a obsahující jen WiFi připojení, tak i zkušenému programátorovi tato práce zabere desítky hodin. Při pokusu o tento přístup jsem naprogramoval chat aplikaci na 612 řádků kódu.

6.2 Kolaborativní přístup

Díky využití našeho kolaborativního frameworku se celý kód síťové obsluhy vejde na tři řádky kódu. Programátor jen nastaví dle Holywood principu, jaká aktivita se má po připojení zavolat a poté spustí LauchActivity a jeho práce je dokončena. Toto napsání zabere programátorovi 5minut i s nastavením projektu a nahráním do telefonu.

6.3 Finanční srovnání nákladů

Pro srovnání finančních nákladů beru příklad napsání jednoduché aplikace, která má jedno textové pole, které lze editovat. Tuto editaci uvidí v reálném čase všichni kolaborující uživatelé.

	Konvenční přístup	Kolaborativní přístup
Obsluha síťového připojení	100h	0h
Chatovací aktivita	5h	5h
Celkem	5h	105h

Tabulka 6.1. Tabulka srovnání časové náročnosti napsání aplikace

Za předpokladu, že si najmeme Android programátora, který si účtuje 400 Kč za hodinu, dostaneme jednoduchý výsledek, že bez využití frameworku by aplikace vyšla na 42 000,- oproti tomu s využitím frameworku jen 2 000,- (framework bude poskytován zdarma jako open-source).

Kapitola 7
Framework

7.1 Použití frameworku

Framework využívá tzv. Hollywood Principle – Don't Call Us, We'll Call You![11]. Díky tomu programátorovi stačí nastavit jakou aktivitou mu aplikace začínat a při startu zavolat aktivitu LaunchActivity. Tato aktivita má své základní uživatelské rozhraní, ve kterém si uživatel může nastavit připojení, přes které se chce připojovat a pak může buď vytvořit server, a nebo se připojit již k existujícímu serveru. Po úspěšném připojení framework zavolá programátorem zvolenou aktivitu, ve které již bude fungovat kompletní síťové propojení mezi zařízeními, takže programátor může plně využívat funkce třídy MtdsFacade.

7.1.1 Startovací aktivita

Tato aktivita je zavolána ihned na začátku frameworku. Jedná se o startovací aktivitu frameworku. Na obrázku 7.1. si můžete povšimnout, že nám aktivita nabízí na výběr ze tří možností.

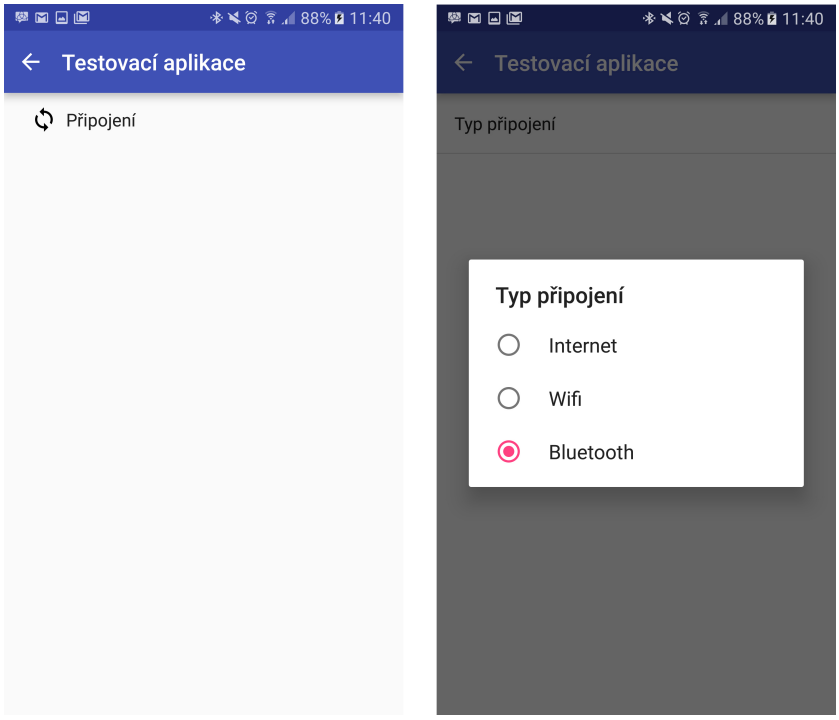
- Vytvořit server
- Připojit k serveru
- Nastavení

Jednotlivé možnosti jsou rozepsány v kapitolách 7.2, 7.3 a 7.4.

Obrázek 7.1. Obrázek zobrazující startovací aktivitu frameworku.

7.1.2 Nastavení

Aktivita nastavení rozšiřuje android třídu PreferenceActivity. V aktivitě stačilo nastavit pouze xml soubor, který definuje, jaké nastavení se v této aktivitě má zobrazit a ostatní důležitý kód vygenerovalo Android Studio. Po zvolení typu připojení (obrázek 7.2.,vpravo), se vybrané nastavení automaticky ukládá do SharedPreferences viz kapitola 5.6.5.



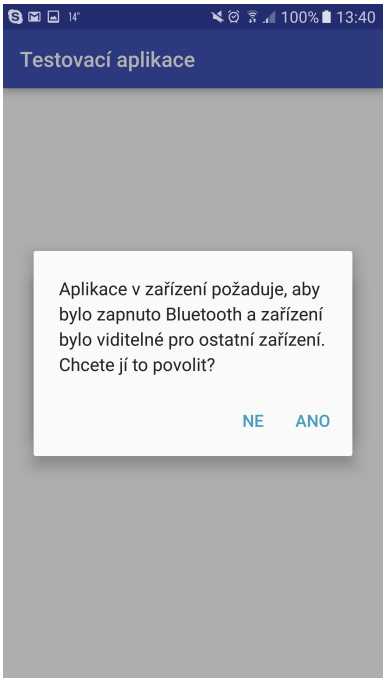
Obrázek 7.2. Obrázek zobrazující aktivitu nastavení.

7.1.3 Vytvořit server

V aktivitě vytvořit server rozlišujeme dva přístupy. První přístup je offline mód. Při vytváření serveru přes Bluetooth nebo Wifi framework nejprve ověří, zda je tato technologie zapnutá. Pokud je technologie vypnutá, požádá uživatele o zapnutí viz obrázek 7.3. Po této kontrole již framework vytvoří server a čeká na připojení alespoň jednoho klienta. V případě Wifi připojení, ještě zaregistruje vytvořený server jako službu pomocí Network Service Discovery (NSD).[1]

V druhém přístupu online módem, kdy se vytváří server pro online kolaboraci, se na zařízení vytvoří server, který čeká na připojení klientů. V tomto módu zařízení funguje pouze jako server, a nemůže s ostatními kolaborovat. Aby se k zařízení mohli klienti připojit, je třeba znát, na jaké IP adrese zařízení čeká na připojení.

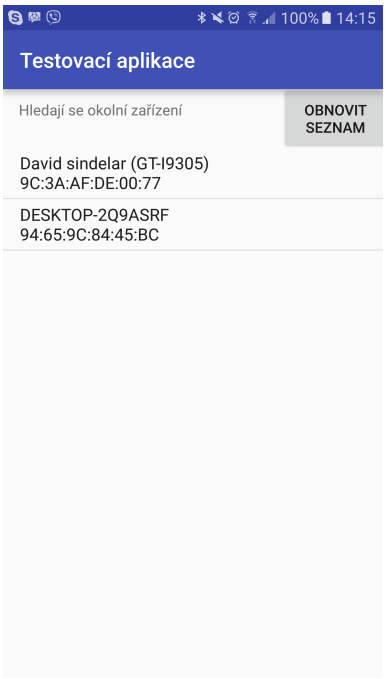
[1] https://developer.android.com/training/connect-devices-wirelessly/nsd.html



Obrázek 7.3. Obrázek zobrazující žádost o zapnutí bluetooth.

7.1.4 Připojit k serveru

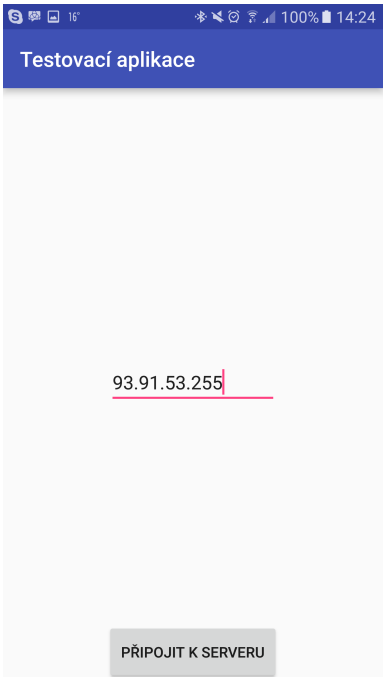
Podobně jako v aktivitě Vytvořit server (7.1.3) rozlišujeme dva přístupy. V offline přístupu pomocí Bluetooth nebo Wifi se ověří, zda jsou technologie zapnuté. Jakmile jsou technologie zapnuté, začne vyhledávat zařízení v okolí a vypisovat do ListView, jak si lze povšimnout na obrázku 7.4.



Obrázek 7.4. Obrázek zobrazující vyhledávání okolních zařízení.

Při kliknutí na položku seznamu, se framework začne připojovat k vybranému serveru. Při úspěšném spojení se zavolá programátorem nadefinovaná aktivita, která již může s ostatními zařízeními může přes framework komunikovat.

V případě online připojení, se zařízení v okolí nevyhledávají. Uživateli se zobrazí textové pole, do kterého je potřeba zadat adresu serveru, viz obrázek 7.5.



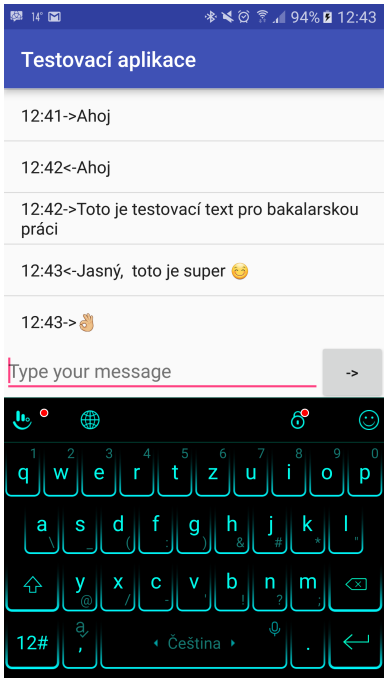
Obrázek 7.5. Obrázek zobrazující připojení k online serveru.

7.2 Testovací aplikace

Jako testovací aplikace jsem zvolil dvě jednoduché aplikace. První aplikací je klasická chat aplikace, kdy si uživatelé mezi sebou mohou odesílat zprávy. Druhá aplikace se zaměřuje na kolaboraci v reálném čase, kdy uživatelé vidí jedno textové pole, které mohou všichni naráz upravovat.

7.2.1 Chat aplikace

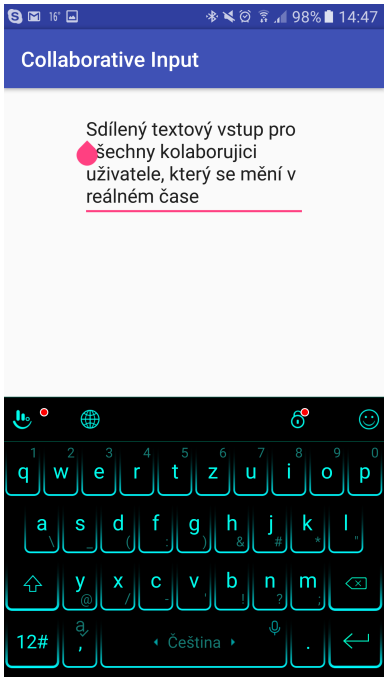
Jedná se o jednoduchou aplikaci o velikosti jedné aktivity a jejího layoutu. Nadefinoval jsem nový objekt typu ChatObject, který si mezi sebou zařízení odesílají. Tyto operce mi zabraly 30 minut času a funkční chatovací aplikace byla v provozu, jak pro online mód, tak i pro offline.



Obrázek 7.6. Obrázek zobrazující testovací chat aplikace, fungující na tomto frameworku.

7.2.2 Aplikace pro kolaboraci v reálném čase

Podobné jako chat aplikace z kapitoly 7.2.1 se tato aplikace skládá z jedné aktivity, jejího layoutu a objektu EditInputTextObject, který aplikace odesílá. Jak je vidět na obrázku 7.7. v aplikaci se nachází jedno textové pole, které lze libovolně upravovat. Každá tato úprava je ihned odeslána na všechna kolaborující zařízení, takže změna je okamžitá,



Obrázek 7.7. Obrázek zobrazující kolaborativní úpravu textového pole v reálném čase.

7.3 Instalace frameworku

Požadavky frameworku

- Minimální Android SDK verze: 19
- Vývojové prostředí s podporou Gradle

Instalace frameworku je velmi jednoduchá. Framework je poskytován jako Android knihovna v souboru s příponou aar. Programátor tento soubor importuje do svého projektu. V případě, že vývojové prostředí nepřidá závislost do build.gradle na tuto knihovnu, je potřeba ji přidat manuálně.

Postup pro Android Studio:

1. Vytvořit nový projekt
2. Pravým tlačítkem kliknout v projektech na app a zvolit možnost Open Module Settings
3. V levé horní části nového okna, kliknout na + (New Module)
4. Vybrat možnost Import .JAR/.AAR Package
5. Kliknout na Next
6. Ve File name zvolit cestu ke knihovně
7. Kliknout na Finish

Po úspěšném importování se přes Gradle stáhnou všechny důležité závislosti a programátor může začít s frameworkem pracovat.

Kapitola 8
Testování aplikace

8.1 Testování rychlosti technologií

Framework aktuálně poskytuje tří různé technologie připojení:

- Bluetooth
- Wifi v místní síti
- Internet

V následující tabulce 8.1. jsou porovnány rychlosti jednotlivých technologií. Každá technologie připojení a typ zprávy byl otestován na 100 odeslaných zprávách, z nichž byl vytvořen aritmetický průměr. V prvním sloupci je uveden počet znaků ve zprávě. Následující sloupce jsou již jednotlivé technologie a určují, za jak dlouho tuto zprávu obdrželo třetí zařízení.

Počet znaků ve zprávě	BT [ms]	WiFi [ms]	Internet [ms]
1	62,1	150,2	258,1
10	103,2	95,8	246,2
100	87,6	112,5	247,4
10 000	198,3	88,4	311
1 000 000	10101,1	1056	2562,7

Tabulka 8.1. Tabulka porovnání rychlostí, za jak dlouho třetí zařízení obdrží zprávu o různém počtu znacích

Testy byly prováděny za pomocí těchto zařízení: Samsung Galaxy S7 (Android 6.0.1), Samsung Galaxy S3 (Android 4.4.4) a HTC ONE M7 (Android 5.0.1). Dle tabulky 8.1. lze zjistit, že WiFi technologie v místní síti dopadla nejlépe. Avšak test byl prováděn v 5G pásmu, při signálu -65dBm. Žádná jiná síť se v 5G pásmu v okolí nenacházela, takže testování na této technologii proběhlo téměř v ideálních podmínkách. V klasické 2,4G síti, kde je větší využití se výsledky budou lišit.

Internetová technologie byla změřena následujícím způsobem. Telefon byl pomocí VPN připojen k VPS serveru o rychlosti 350Mbit/s download a 121,16Mbit/s upload. Komunikační tcp port frameworku byl z VPS serveru přesměrován na připojený telefon a tak vznikl veřejný server, na který se již dalo odkudkoliv připojit. Telefon byl na připojení o rychlosti 120Mbit/s download a 11Mbit/s upload.

Poslední technologie Bluetooth byla měřena ve chvíli, kdy zařízení od sebe byla vzdálena zhruba 70cm. U tohoto připojení stojí za zmínku situace, kdy bylo odesláno ve zprávě 1 000 000 znaků. Operace trvala celých 10 vteřin, než zprávu obdrželo třetí zařízení, což je 10x déle než ve WiFi připojení.

Kapitola 9
Závěr

Bakalářská práce byla vypracována na základě zadání katedry ekonomiky, manažerství a humanitních věd, fakulty elektrotechnické, Českého vysokého učení v Praze v rámci studijního programu Softwarové technologie a management, obor Manažerská informatika.

Cílem této bakalářské práce bylo prostudovat existující nástroje pro podporu AOP, seznámit se s Android OS a navrhnout řešení frameworku, podle kterého se budou vyvíjet kolaborativní aplikace a demonstrovat řešení na jednoduché aplikaci.

Práce se v první části zabývá rozborem řešení tohoto zadání a podrobně rozebírá jednotlivé architektury připojení. Z podrobné analýzy vyplynula dvě vhodná řešení, která nasměrovala řešitele k využití architektury klient-server pro online kolaboraci a zapojení do kruhu pro offline kolaboraci.

V následující části se práce zabývá návrhem řešení a problémy s ním vzniklé. Jednotlivé problémy jsou v dané částí podrobně popsány i s řešením, jak je vyřešit.

Výsledkem práce je framework v podobě Android knihovny, který velmi usnadňuje vývoj kolaborativních android aplikací. Framework umožňuje jak online, tak i offline kolaboraci. Framework poskytuje objekt MtdsFacade implementovaný pomocí návrhového vzoru Fasáda. Tento přístup zajistí programátorovi jednoduchou práci s frameworkem a tím i ušetří jeho čas.

9.1 Výhody řešení

9.1.1 Časová úspora při implementaci

Jednou z hlavních výhod je časová úspora při vývoji aplikace. Zatímco napsání klasické kolaborující aplikace trvá několik hodin, při použití frameworku je možné mít do hodiny naprogramovanou funkční aplikaci viz. Chat aplikace v kapitole 7.2.1.

9.1.2 Snadná rozšiřitelnost

Napsání frameworku jako Android knihovny umožňuje velmi snadnou rozšiřitelnost. Celý framework je totiž po zkompilování v jednom jediném souboru s příponou aar, který lze v jakémkoliv Android projektu importovat.

9.1.3 Výběr módu

Jako další velkou výhodou je přepínání mezi online a offline módem. Framework dokáže pracovat jak přes internet, tak i přes Bluetooth nebo Wifi (v místní síti).

9.2 Budoucí práce

9.2.1 Vzhled frameworku

Framework aktuálně poskytuje vlastní jednoduchý vzhled během vytváření serveru nebo připojování k serveru. Tento vzhled nemusí každému programátorovi vyhovovat. V budoucí práci plánuji možnost zvolení vlastního designu frameworku.

9.2.2 Online propojení

Zvolená architektura klient-server funguje dobře, avšak při připojení zařízení, které nemá dobrý a stabilní internet, docházelo k chybám a občas i přepisům. Z tohoto důvodu do budoucí práce plánuji server udělat i datovým uložištěm, aby klienti by k serveru přistupovali jen přes HTTP protokol, konkrétně přes REST API, který bude datově mnohem méně náročný.

9.3 Možná omezení

9.3.1 Operational transformation

Tato technologie funguje velmi rychle, avšak v jistých situacích vzniklo několik chyb, kdy každé zařízení mělo jiný text. Proto plánuji v budoucnu tuto technologii zdokonalit.

Literatura

[1] EDWARDS, Jim. The iPhone 6 Had Better Be Amazing And Cheap, Because Apple Is Losing The War To Android *IPhone v. Android Market Share - Business Insider*, 2015.
http://www.businessinsider.com/iphone-v-android-market-share-2014-5.

[2] BAŠEK, J. *Generování uživatelskžch rozhraní pomocí UIP a JPA.*,ČVUT, 2013. Diplomová práce
https://dip.felk.cvut.cz/browse/pdfcache/basekjin_2013dipl.pdf.

[3] CodeProject *Introduction to Object Oriented Programming Concepts (OOP) and More.*
http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep.

[4] Tomáš Černý *Software architectures [přednáška].*,Praha : ČVUT,21.11.2015

[5] DazeInfo *MOBILE 2 Billion Smartphone Users By 2015 : 83%of Internet Usage From Mobiles [Study]*
http://dazeinfo.com/2014/01/23/smartphone-users-growth-mobile-internet-2014-2017/.

[6] ŠEBEK, J. *Aspect-oriented user interface design for Android applications*,ČVUT, 2014. Diplomová práce
https://dip.felk.cvut.cz/browse/pdfcache/sebekji1_2014dipl.pdf.

[7] FORAYS IN SOFTWARE DEVELOPMENT *Stack based vs Register based Virtual Machine Architecture, and the Dalvik VM*
https://markfaction.wordpress.com/2012/07/15/stack-based-vs-register-based-virtual-machine-architecture-and-the-dalvik-vm/.

[8] FALATA, D. *Mobilní aplikace pro OS Android pro ovládání domovní automatizace a zabezpečení*,ČVUT, 2015. Bakalářská práce

[9] codeanywhere *Real-Time Collaboration Issues and Solutions*, 2014
https://blog.codeanywhere.com/real-time-collaboration-issues-and-solutions/.

[10] *Realtime Framework Messaging service.*
https://framework.realtime.co/.

[11] Fowler, Martin. *InversionOfControl*, 2005.
http://martinfowler.com/bliki/InversionOfControl.html.

Příloha A
Použité zkratky

A.1 Zkratky

AOP	Aspektově orientované programování
MDA	Modelem řízená architektura
OOP	Objektově orientované programování
BT	Bluetooth
WLAN	Bezdrátová síť
FUP	Fair User Policy
IP	Internet Protocol
OSI	Open Systems Interconnection
UID	Unique Identification Number
NSD	Network Service Discover

Příloha B
Obsah cd

```
./bakalarska-prace.pdf
./knihovna/
./knihovna/mtdslibrary.aar
./latex.zip
./vývoj.zip
```