VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

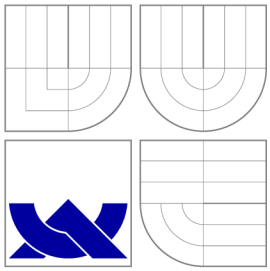
VZÁJEMNÁ NOTIFIKACE INTELIGENTNÍHO TELEFONU A SERVERU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

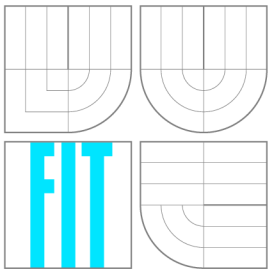
AUTOR PRÁCE
AUTHOR

Bc. MARTIN DOUDĚRA

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VZÁJEMNÁ NOTIFIKACE INTELIGENTNÍHO TELEFONU A SERVERU

MUTUAL NOTIFICATION OF SMART PHONE AND SERVER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MARTIN DOUDĚRA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN KOŘENEK, Ph.D.

BRNO 2015

Abstrakt

Tato diplomová práce se zabývá notifikacemi mezi chytrým telefonem a aplikačním serverem ovládající inteligentním domácnost. Zaměřuje se na mobilní platformu Android, na kterou je navržené řešení implementováno. Mobilní zařízení je ze strany serveru přímo neadresovatelné, a proto server nemůže zaslat zařízení zprávu přímo. Využívá se proto notifikační služby Google Cloud Messaging. Pro notifikaci serveru je využito aktuální polohy uživatele a jeho nadefinovaných oblastí. Server je pak informován o zónách, které uživatel překročí. S těmito informacemi může server vykonávat definované události automatizovaně. Implementované řešení bylo otestováno v reálném prostředí systému. Pravidelně byly vydávány alpha a beta verze aplikace a vyhodnocována zpětná vazba od uživatelů.

Abstract

This thesis deals with notification between smart phone and application server which controls intelligent home. It is focused on Android mobile platform and it is implemented for that platform. Notification service Google Cloud Messaging is used because it is not possible to directly address mobile phones from server. Based on actual position and user defined area the server is informed about moving to or from the area. The server can perform defined event automatically with these information. Implementation was tested in real system. Alpha and beta versions were regularly published.

Klíčová slova

Inteligentní domácnost, Android, push notifikace, Google Cloud Messaging, geofencing, Google Maps API, Location API

Keywords

Intelligent home, Android, push notification, Google Cloud Messaging, geofencing, Google Maps API, Location API

Citace

Martin Douděra: Vzájemná notifikace inteligentního
telefonu a serveru, diplomová práce, Brno, FIT VUT v Brně, 2015

Vzájemná notifikace inteligentního telefonu a serveru

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Kořenka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Douděra
26. května 2015

Poděkování

Na tomto místě chci zejména poděkovat svému vedoucímu, Ing. Janu Kořenkovi, Ph.D., a konzultantovi, Ing. Martinu Žádníkovi, Ph.D., za cenné rady a připomínky. Velký dík patří i celému týmu IoT za spolupráci a příjemné pracovní prostředí.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Ještě teprve pár desítek let zpět jsme inteligentní budovy viděli jen ve sci-fi filmech. Jen málokdo tenkrát věřil, že se jednou v něčem podobném ocitneme možná také. Nejprve se zcela výhradně inteligentní systémy dostaly do velkých firemních budov. Díky stále se zlevňujícím technologiím a moderním trendům se posledních několik let dostává automatizace i do menších staveb, a tedy i do rodinných domů a bytů. Inteligentní domácnosti pak obyvatelům slibují vyšší bezpečnost, úsporu energií, automatizování rutinních prací, vyšší míru komfortu a flexibilitu.

První inteligentní systémy byly řešeny samozřejmě pomocí drátového propojení s centrální jednotkou. To přinášelo často mnoho práce při návrhu a realizaci tohoto systému do nového obydlí. Pro majitele již fungujících "obyčejných" domů to znamenalo i menší stavební úpravy a nemalé částky vynaložené na to, přestavět svůj dům na inteligentní. Až v dnešní době, kdy se veškeré existující drátové systémy převádí do bezdrátových řešení pro větší komfort uživatele, se objevuje možnost velmi jednoduché instalace. V inteligentním bydlení to znamená přímé bezdrátové propojení senzorů a koncových prvků do sítě Internetu nebo s řídící jednotkou domu. To v praxi znamená, že například vypínač světel může být umístěn kdekoliv do pokoje pomocí oboustranné lepící pásky bez nutnosti přivádět elektrické vedení. Navíc umožní dynamicky definovat, které světlo nebo skupinu světel chce uživatel ovládat.

Jelikož v dnešní době má většina lidí chytrý telefon, odpadá tu nutnost ovládání domácnosti na vestavěných displejích v domě. Tato funkčnost se převádí do chytrých zařízení, které mají uživatelé většinu času po ruce. Jediné, co je tedy potřeba k ovládání, je aplikace nainstalovaná na chytrém zařízení. Domácnost pak může být ovládána a sledována odkudkoliv na světě, kde je zařízení připojeno k síti Internetu.

Tato diplomová práce se zabývá automatizací domácnosti na základě polohy uživatelova mobilního telefonu a notifikací uživatele na mobilní telefon. Jelikož není možné jednoznačně adresovat chytré zařízení a asynchronně zasílat zprávy ze serveru, práce se zaměří zejména na nativní podporu notifikací u jednotlivých platforem. Cílem je vymyslet způsob řešení pro reálný systém inteligentní domácnosti, následně ho implementovat, nasadit do existujícího systému a otestovat.

Na začátku práce je rozebrána obecně inteligentní domácnost, existující řešení a výzkumný projekt Fakulty informačních technologií VUT v Brně. V dalších dvou kapitolách jsou popsány možnosti služby Google Cloud Messaging a geolokace uživatele. Tyto znalosti jsou následně využity v kapitole 5 pro návrh. Následuje implementace a experimentální testování, kde je prověřena spolehlivost řešení a jeho nároky. Na závěr je práce vyhodnocena a jsou navrhnuta možná rozšíření.

Kapitola 2

Inteligentní domácnost

Člověk je vynalézavý a neustále zdokonaluje věci kolem sebe. A právě od dob, kdy byla vynalezena elektřina, prošla každá domácnost modernizací. V dnešní době si již málokdo dokáže představit život bez elektřiny, která nám v mnoha ohledech ulehčila každodenní rutinní práce a celkově zvýšila životní komfort. V předchozím století, kdy začali lidé používat technologie v domácím prostředí, došlo k třem důležitým milníkům:

1. Bylo třeba připojit domácnosti do externí infrastruktury, tzn. k elektřině, vodě, odpadům, plynu, telefonu, kabelové televizi apod.

2. Rozšířit připojení po domácnosti, tzn. teplá a studená voda, elektroinstalace, kroucená dvojlinka pro telefon apod.

3. Zavedení domácích spotřebičů a koncových zařízení.

Zatímco první dvě části jsou většinou zabudované technologie, třetí část považujeme za spotřební zboží.[5]

Automatizace domácnosti začala již před mnoha lety. I když první vynálezy nelze nazývat inteligentními, každý vynález, který usnadnil náš život, přispěl k rozvoji chytré domácnosti.

Jeden z prvních vynálezů byl dálkový ovladač, který si patentoval Nikola Tesla už v roce 1898. Mezi léty 1901 až 1920 vznikaly první domácí spotřebiče. Kolem roku 1925 byly správné podmínky pro rozkvět elektrických spotřebičů, a to díky levnému připojení elektřiny, spolehlivým spotřebičům a elektroinstalaci. Zasloužila se o to asociace *The Electrical Development Association* (EDA).

Mezi léty 1918 až 1939 se procento připojených domácností zvýšilo z 6% na 66%. Zpočátku měly domácnosti jen jednu 5A zásuvku. Asi jen jedna ze tří domácností měla více než dvě zásuvky. Lidé se báli zavedení elektřiny do svého obydlí.[5]

V roce 1966 Jim Sutherland vynalezl systém pro automatizaci domácnosti, který se jmenoval *ECHO IV*. Jméno pochází ze zkratky z anglického spojení *Electronic Computing Home Operator*. Tento systém umožnil skladovat recepty, vytvořit nákupní lístek, sledovat výdaje a příjmy rodiny, zapínat a vypínat spotřebiče, kontrolovat teplotu v domácnosti a předpovídat počasí. Byl to autorův soukromý projekt a nebyl nikdy uveden na trh. [13]

Ke zlevnění elektroniky přispěl vynález mikrokontroléru v roce 1971. V roce 1975 byl vyvinut protokol X10 pro komunikaci mezi elektronickými zařízeními a umožnil tak spolupracovat více zařízením. Ke komunikaci využíval elektrické vedení.

Pojem inteligentní dům (v originálním znění "Smart House") byl rozšířen americkým sdružením *American Association of Housebuilders* v roce 1984.[4] Vznikl projekt *Smart*

House Project vedený *National Research Center of the National Association of Home Builders* (NAHB). Tato asociace popisuje chytrou domácnost jako propojenou jedním kabelem, který se skládá z napájecího vodiče, komunikačního vodiče pro telefon i video a dalšího vodiče, které propojují spotřebiče a světla s elektronickými zařízeními, které zajišťují napájení a spínání.[3]

I když se většina z nás s pojmem "inteligentní domácnost"již setkala, už pouze zlomek z nás má přesnou představu o tom, co to přesně znamená. Neexistuje přesná definice a pravděpodobně za několik desítek let bude pojem inteligentní budova znamenat již mnohem modernější technologie.

Jak řekl ve své knize Richard Harper[10], domácnost není chytrá, protože je dobře navrhnuta a postavena, efektivně využívá prostor, je ekologická, čerpá energii ze solárních panelů nebo recykluje odpadní vodu. Chytrá domácnost může zmíněné technologie obsahovat, a také často obsahuje. Ale to, co jí dělá chytrou, jsou právě interaktivní technologie, které tyto části propojují a umožní jim tak spolupracovat.

Miroslav Valeš (2006, s. 1)[14] definoval inteligentní dům následovně: "*Inteligentní dům v nejširším možném smyslu slova je budova vybavená počítačovou a komunikační technikou, která předvídá a reaguje na potřeby obyvatel s cílem zvýšit jejich komfort, pohodlí, snížit spotřebu energií, poskytnout jim bezpečí a zábavu pomocí řízení všech technologií v domě a jejich interakcí s vnějším světem.*"

Inteligenci můžeme rozdělit do následujících pěti stupňů podle míry inteligence[14]:

1. **Obsahující inteligentní zařízení a systémy** – Obsahuje fungující zařízení a systémy pracující samostatně.

2. **Obsahující inteligentní komunikující zařízení a systémy** – Obsahuje inteligentní zařízení, které si spolu z důvodu zdokonalení své činnosti vyměňují informace a zprávy mezi sebou.

3. **Propojený dům** – Domácnost je propojena pomocí vnitřní a vnější sítě, což umožňuje vzdálené ovládání systému, přístup ke službám a informacím z domu i mimo něj.

4. **Učící se dům** – Zaznamenává aktivity v domácnosti a podle nasbíraných dat samočinně ovládá domácnost.

5. **Pozorný dům** – Neustále je vyhodnocována aktuální poloha a aktivita lidí a technologie jsou ovládány podle předvídatelných potřeb.

Pojem inteligentní domácnost je často zaměňován za chytrá domácnost, automatizace domácnosti nebo digitální domácnost. Existuje i moderní vědní obor nazývaný *domotika*.

V inteligentní budově lze najít několik různých zařízení. Mohou být buď drátově nebo bezdrátově propojené. Koncové monitorovací prvky jsou nazvány senzory. Ty mají za úkol sledovat jednu nebo více veličin a v určitých intervalech posílat naměřené hodnoty kontroléru. Kontrolér je centrální prvek v domácnosti, který shromažďuje data od všech senzorů, řídí domácnost na základě aktuálních dat a může také odesílat data na server. Celou domácnost je potřeba řídit, a proto je nezbytné mít ovládací prvek. U starších řešení to je vestavěná řídící jednotka. Dnešní doba si však žádá flexibilnější řízení pomocí mobilních telefonů a tabletů.

Podobná zařízení jako senzory jsou aktory. Ty neměří žádnou veličinu, ale vykonávají nějakou činnost. Často je senzor i aktor obsažen v jednom zařízení. Jedná se tedy o multifunkční zařízení (dále jen "senzor-aktor"), které je schopno snímat i ovládat stav. Většina

zařízení ovládajících stav něčeho je senzor-aktor, jelikož je nezbytná i zpětná vazba o stavu provedení. Například pro systém ovládání oken je využit pouze aktor, který má dva stavy – otevřeno a zavřeno. První problém je, že nelze rozeznat aktuální stav okna. Pokud by systém informoval uživatele o aktuálním stavu, musel by vědět počáteční stav, ukládat si změny stavu a předpokládat, že nikdo jiný manuálně nezasáhne do otevření nebo zavření okna. Z toho plyne, že inteligentní domácnost by nebylo možné ovládat manuálně. Problém by nastal i při nepodařeném zavření nebo otevření okna (např. překážka bránící v zavírání okna). Samostatné aktory je tedy možné použít pouze tam, kde není důležitá informace o aktuálním stavu.

I když se někteří lidé chytrých domácností a vzdáleného přístupu prozatím obávají z důvodu bezpečnosti a nemají k němu důvěru, v následujících letech to bude samozřejmostí. Dle průzkumu[1] na území USA bylo v roce 2012 nainstalováno 1,5 miliónů inteligentních systémů do domácností a v roce 2017 se odhadují více než 8 miliónů instalací.

Internet věcí je označení pro propojení vestavěných zařízení s Internetem. Podle výzkumu[2] společnosti Gartner Inc. bude do roku 2020 připojených 26 biliónů zařízení do Internetu. Výzkum[3] od společnosti Allied Business Intelligence dokonce tvrdí, že do roku 2020 bude připojených více než 30 bilionů zařízení. V každém případě bude lidstvo v následujících letech čelit masivnímu nárůstu síťové komunikace.

2.1 Stávající řešení

Tato kapitola je zaměřena na známé řešení bezdrátových inteligentních domácností. Jelikož je těchto řešení hodně a tato práce se zabývá zejména ovládáním na chytrých zařízeních, výběr je omezen na nejznámější řešení, která již využívají k ovládání chytrá zařízení.

2.1.1 ELKO EP

Společnost ELKO EP[4] je česká firma zabývající se elektroinstalací již dvacet jedna let. Je jedním z předních výrobců elektroinstalace v ČR. Veškerý sortiment má rozdělený do tří skupin:

- **relé** – klasické modulové přístroje
- **INELS RF Control** – bezdrátový systém
- **INELS BUS System** – sběrnicový systém

Posledních sedm let společnost rozvíjí oblast inteligentních elektroinstalací. Konkrétně to je systém iNELS Smarthome Solutions, který sdružuje bezdrátové řešení a sběrnicovou inteligentní elektroinstalaci.

Domácnost lze ovládat z mobilního telefonu, tabletu nebo chytré televize. Podporovány jsou dvě mobilní platformy, a to Android a iOS. Aplikace na chytré zařízení jsou zdarma ke stažení na stránkách Google Play a iTunes. Zajímavostí z pohledu implementace je, že pro obě platformy mají oddělené aplikace pro mobilní zařízení a pro tablet. Kompatibilita aplikace s mobilním telefonem a tabletem se standardně řeší v rámci jedné aplikace. Každá aplikace komunikuje s jiným prvkem systému. Najdeme zde i promo aplikace, které jsou

[1] https://www.abiresearch.com/press/15-million-home-automation-systems-installed-in-the
[2] http://www.gartner.com/newsroom/id/2636073
[3] https://www.abiresearch.com/press/more-than-30-billion-devices-will-wirelessly-conne
[4] http://www.elkoep.cz/

(a) Home Control Mobile (b) Home Control RF Mobile (c) Home Control IR Mobile

Obrázek 2.1: Ukázka obrazovek aplikací od firmy ELKO – převzato z Google Play

reálně nasazené v showroomech a lidé si můžou vyzkoušet, jak aplikace reaguje. Výsledek je vidět na on-line kameře, které je součástí systému.

Pro napojení bezdrátové chytré domácnosti je potřeba zakoupit chytrou RF krabičku, která se bezdrátově spojí až s 40 prvky v domácnosti. K ovládání pak slouží aplikace pojmenovaná *iNELS Home Control* (obrázek 2.1a) nebo *iNELS Home Control RF* (obrázek 2.1b). Aplikace uživateli umožňuje regulaci teplovodního nebo elektrického podlahového vytápění, měření teploty bezdrátovými senzory, spínání spotřebičů (garážová vrata, žaluzie atd.), stmívání světel, časové spínání, integrace kamer a definici scén.

Zajímavostí je aplikace *iNELS Home Control IR Mobile*, která komunikuje s chytrou IR krabičkou. Ta umí emulovat až sto různých povelů běžných ovladačů domácích spotřebičů (TV, video, DVD přehrávač apod.). I samotným vzhledem (obrázek 2.1c) chtěli vývojáři napodobit klasické ovladače, na které jsme zvyklí. Aplikace podporuje i funkci hromadného zaslání příkazů, což může ulehčit rutinní práce.

2.1.2 Jablotron

Česká společnost Jablotron[5] působící od roku 1990 zejména v oblasti zabezpečovací techniky. Firma se specializuje na tato odvětví:

- **Alarmy** – Zabezpečení objektů, které pokryje jak malé objekty, tak i velké firmy. Alarmy i kamery jsou bezdrátové. V bezpečnostním centru Jablotron je možné platit měsíční paušál, kdy se hlášení o alarmu posílají na centrálu a podle výše ceny služby na alarm zareagují.
- **Autotechnika** – Pokrývá klasické autoalarmy, GPS[6] a GSM[7] alarmy, monitoring,

[5] http://www.jablotron.com/
[6] Globální polohový systém
[7] Globální systém pro mobilní komunikaci; původní zkratka z francouzského spojení *Groupe Spécial Mo-*

parkování i dálkové ovládání. Alarmy mohou informovat uživatele o stavu baterie auta, nechtěném odtahu policií nebo zapomenutí zamknutí. Opět lze napojit reportování na bezpečnostní centrálu Jablotron.

- **Komfort systémy** – Toto odvětví pokrývá regulace topení, ovládání prvků a měření různých veličin. Většina prvků v těchto systémech je bezdrátová a lze ji ovládat vzdáleně.

- **Zdravotní alarm** – Obsahuje dva produkty. První se nazývá Nanny. Je to zařízení pro monitoring dýchání novorozenců a případný alarm v případě zástavy dechu. Druhý je mobilní telefon Granny, který se dodává s tísňovým tlačítkem pro přivolání pomoci. Mobil se podobá vzhledu pevné linky. Prvotně byl určen pro staré lidi. Avšak vzbudil zájem nejen u starých lidí, ale i u mobilních operátorů při konkurenčním boji s pevnými linkami. Na toto odvětví nakonec v roce 2005 založili dceřinou společnost Jablocom[8]. Nyní mají v nabídce stolní telefon Raven s operačním systémem Android.

Zabezpečovací systém objektů lze ovládat i přes Internetové rozhraní nebo pomocí chytrého telefonu. Mobilní aplikace je k dostání zdarma na stránkách Google Play pod názvem MyJABLOTRON (obrázek 2.2). Aplikace umožňuje:

- Ovládat několik bezpečnostních systémů z jednoho místa.
- Zastřežit nebo odstřežit systém.
- Zapínat předprogramované úkoly.
- Sledování aktuální situace v systémech a zobrazení jejich historie.
- Nastavení SMS, emailu nebo push notifikací pro různé události v systému.
- Nastavení oprávnění pro jiné uživatele.

Aby uživatel mohl využívat aplikaci v chytrém zařízení, musí se nejprve zaregistrovat v Jablotron Cloud Service, odkud dostane přidělené bezpečnostní údaje.

2.1.3 Belkin WeMo

Americká společnost Belkin si s projektem WeMo[9] zakládá na jednoduchých produktech, které používají k ovládání Internet. Do sítě se připojují přímo přes WiFi.

Aktuální nabídka prvků firmy Belkin je:

- **Žárovka** – Ovládání a plánování svícení. Může se ovládat jako separátní žárovka nebo se mohou vytvořit skupiny žárovek a ovládat je jako celek.

- **Kamera NetCam** – Kamera například umožňuje nastavení sepnutí žárovek, pokud zaznamená pohyb. Záběry z kamery je možné sledovat odkudkoliv na chytrém zařízení.

- **Chytré zařízení do kuchyně** – Například chytrý hrnec Crock-Pot, u kterého se může nastavovat délka vaření, teplota a samozřejmě na dálku zapínat a vypínat. Nebo například Mr.Coffee, který uvaří kávu na dálku. Lze vytvořit i plán vaření káv.

bile

[8] http://www.jablocom.cz/

[9] http://www.belkin.com/us/Products/home-automation/c/wemo-home-automation/

(a) Seznam zařízení (b) Detail zařízení

Obrázek 2.2: Ukázka obrazovek aplikace MyJABLOTRON – převzato z Google Play

- **Zásuvka** – Lze na dálku ovládat zásuvku a sledovat její spotřebu.
- **Různé senzory** – Senzory pohybu, polohy, oken, dveří a jiné.

WeMo spolupracuje se službou IFTTT (viz sekce 2.1.5). Lze si tedy pro všechny prvky nadefinovat různé scénáře, kdy prvky v domácnosti mohou reagovat na nejrůznější podněty z ostatních prvků v domácností nebo z úplně jiných systémů a služeb. Propojení s IFTTT umožní definování geolokačních oblastí nebo push notifikací. Aplikace je volně ke stažení na Google Play. Na obrazovkách 2.3 je možné pozorovat jednoduché grafické zpracování aplikace.

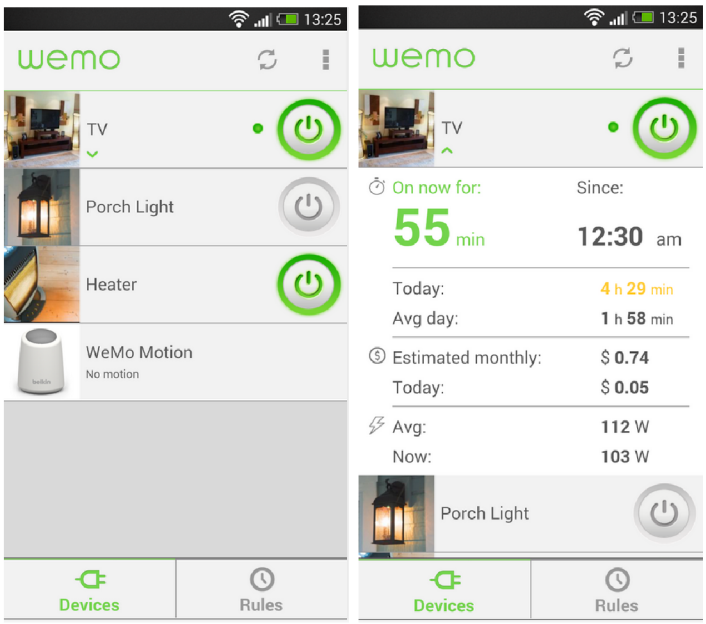
2.1.4 Insteon

Insteon[10] je řešení vedené společností Smartlabs, Inc. založenou v roce 1992. Projekt Insteon byl vypuštěn do světa v roce 2005. Toto řešení má velké množství různorodých koncových prvků.

Jako centrální prvek pro koncové zařízení slouží Insteon Hub, ke kterému se všechny prvky připojí. Pro koncové prvky je to brána do Internetu. Systém obsahuje prvky jako vypínače (pouze se přilepí na zeď), žárovky, ovladače zásuvky, senzory nejrůznějších typů, termostaty, zámek dveří, kamery apod.

Pro vývojáře je zde možnost integrovat své aplikace přes rozhraní REST[11]. Aplikace je opět zdarma a umožní ovládat celou domácnost, plánovat automatické akce, přijímat notifikace, sledovat kamery a provádět nastavení celého systému. Z výše rozebíraných aplikací má tato nejhorší uživatelské rozhraní. Nespokojenost s aplikací vychází i z uživatelských hodnocení.

[10] http://www.insteon.com/
[11] Representational State Transfer



(a) Seznam zařízení (b) Detail zařízení

Obrázek 2.3: Ukázka obrazovek aplikace Belkin WeMo – převzato z Google Play

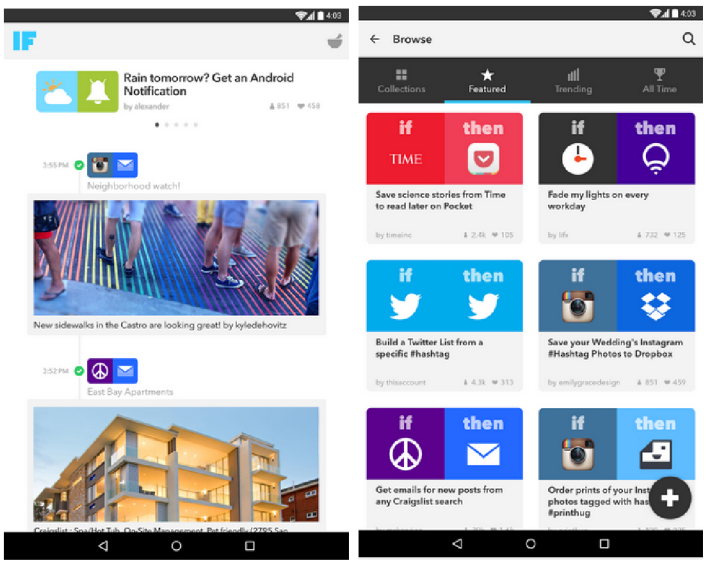
2.1.5 IFTTT

Služba IFTTT[12] je trochu odlišná od předchozích aplikací. Nezprostředkovává přímo chytrou domácnost, ale propojuje různé systémy a umožňuje jim vzájemně spolupracovat. Odtud je i odvozen název služby. Název pochází s anglického spojení "If This Then That." Stejně jako název napovídá, služba umožňuje definovat podmínku a příslušnou akci neboli "když se něco stane, pak něco udělej."

Služba pracuje s úkoly a kanály. Jeden úkol je složen z kanálů. Jeden kanál představuje vstup nebo výstup podmínky. Každý úkol je tedy vytvořen dvěma kanály ve tvaru podmínky: "když *vstupní kanál* pak *výstupní kanál*." Služba také nabízí spoustu přednastavených úkolů, které nazývá "recepty". Uživatel tedy může využít již nadefinovaných úkolů, což usnadňuje práci pro nezkušeného uživatele, kterého by nemuseli i některé možnosti využití napadnout. Jako kanály zde můžeme využít nejrůznější sociální sítě, cloudové úložiště, aktuální hodnoty (počasí, měnové kurzy apod.) a různé zařízení. Například zařízení WeMo (viz sekce 2.1.3) umí spolupracovat s touto službou. Dalšími systémy spojené s chytrou domácností jsou termostat Nest, SmartThings, žárovka Philips Hue, Life360 a další méně známější. Do budoucna se předpokládá nárůst připojených systémů, což umožní propojit tyto systémy a vytvářet tak zajímavé úkoly.

Ke službě existuje i mobilní verze pro platformu Android a iOS. Aplikace má název IF a je k dostání zdarma. Na obrázcích 2.4 jsou příklady obrazovek pro platformu Android. Na levé z nich je vidět historie provedených podmínek. Vpravo si uživatel vybírá ze seznamu receptů nebo si může definovat úkol svůj vlastní.

[12] https://ifttt.com/



(a) Historie podmínek (b) Výběr podmínek

Obrázek 2.4: Ukázka obrazovek aplikace IF – převzato z Google Play

2.2 Projekt IoT

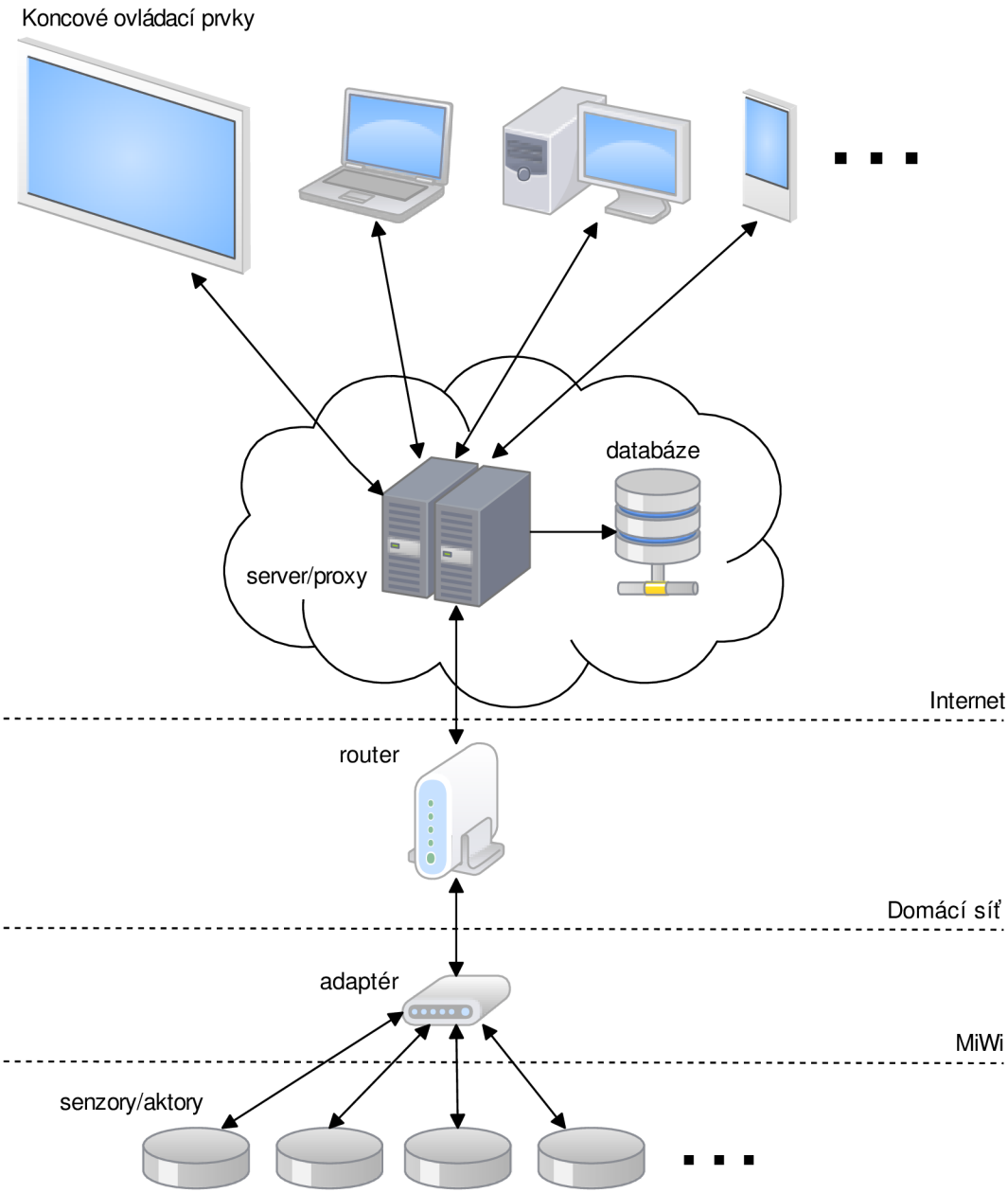
Tato diplomová práce vzniká při spolupráci s výzkumnou skupinou na Fakultě informačních technologií VUT v Brně a týmem IoT, který se zabývá inteligentní domácností. Projekt nese název BeeeOn. V rámci projektu se vyvíjí i primitivní jednoúčelové senzory a aktory. Nicméně se projekt primárně zabývá propojením bezdrátových prvků třetích stran, a tím zapouzdřením různých řešení od různých prodejců, čímž poskytne novou abstraktní vrstvu pro řízení a automatizaci domácnosti.

Na obrázku 2.5 je znázorněna architektura celého systému. Architektura je rozdělena do několika vrstev. Spodní úroveň tvoří senzory a aktory. Senzory monitorují měřenou veličinu a posílají ji bezdrátově na adaptér. Ten přibalí další informace a odesílá jako celek serveru. Server informace zpracuje a uloží do databáze. Koncové ovládací prvky komunikují se serverem při požadavku na data nebo na změnu stavu aktoru. Při požadavku na změnu server šíří informaci přes adaptér až k aktoru.

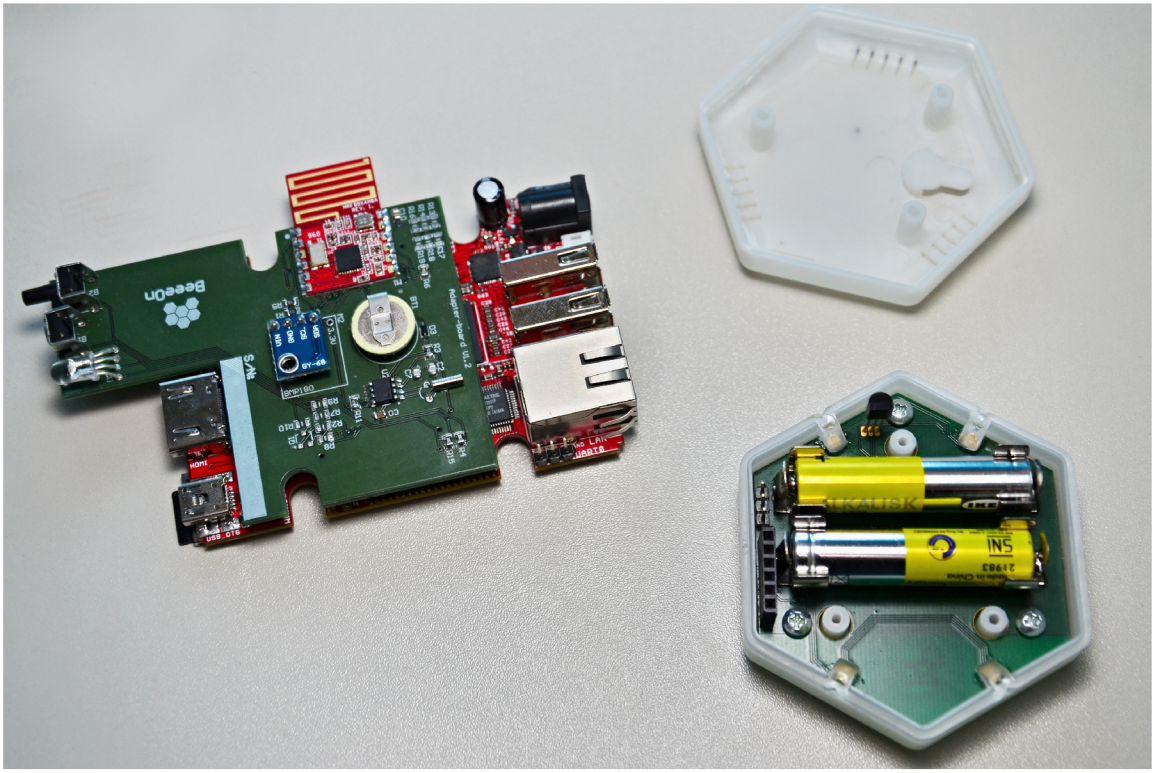
2.2.1 Senzory a aktory

Senzory snímají nějakou veličinu nebo naopak aktory vykonávají určitou činnost. Otevřený teplotní senzor je zobrazen na obrázku 2.6 vpravo. Senzory komunikují bezdrátově s adaptérem (viz kapitola 2.2.2). Komunikace je kvůli prevenci odposlouchávání šifrovaná. Každý prvek je schopen registrace k adaptéru po zatřepání, pokud se adaptér nachází v párovacím režimu. Po úspěšném spárování prvky vysílají jednoznačnou identifikaci a základní údaje o sobě – počet, typ senzorů/aktorů apod. O rozeznání a napojení prvků třetích stran se stará adaptér.

Pokud se senzor úspěšně spáruje, uspává se na určitý čas. Tento čas lze měnit. Při každém probuzení senzor odesílá nově naměřená data a ze serveru (viz kapitola 2.2.3) dostává nový čas, na který se uspává. U senzor-aktorů je tento přístup podobný, nicméně aktor musí stále naslouchat pro případný příchozí požadavek na provedení akce. Odeslání nově naměřených dat se pak provádí ve stejném intervalu jako u senzoru. Odeslání nových



Obrázek 2.5: Architektura BeeeOn systému složená z několika vrstev



Obrázek 2.6: Adaptér a senzor teploty

hodnot může způsobit i nový požadavek na aktor. V tomto případě se odesílají hodnoty jen z přidružených senzorů k aktoru.

Jelikož je většina senzorů napájena pomocí baterie, uspávání senzoru značně prodlužuje celkovou výdrž senzoru. Každý senzor posílá informace o stavu baterie, aby mohl být uživatel upozorněn na slabou baterii a následnou výměnu baterie. První výjimkou jsou prvky, které ovládají nebo snímají zdroj elektrické energie. Z tohoto zdroje mohou být i napájené. Druhou výjimkou jsou aktory, které vykonávají činnost, která nelze být napájena z baterie. V tomto případě je nutnost mít aktor připojený do elektrické sítě.

2.2.2 Adaptér a směrovač

Adaptér se do vyšších vrstev (tj. na server) registruje pomocí jednoznačné identifikace. Adaptér bez obalu je zobrazen na obrázku 2.6 vlevo. Hlavní funkcí je převod bezdrátových protokolů do IP Ethernet protokolu. Adaptér má na starosti prvotní spárování bezdrátových prvků v domácnosti. Po spárování dokáže komunikovat s každým prvkem. Adaptér nikdy nekomunikuje s nespárovanými prvky. Na rozdíl od senzorů se nesmí uspávat, a tedy musí být dostupný pořád. Adaptér totiž neví, kdy mu přijde další aktualizace ze senzorů.

Jelikož se postupně budou přidávat podporovaná zařízení třetích stran, adaptér je schopný vzdáleně aktualizovat firmware. To umožní reagovat na změny za plného nasazení.

Při výpadku připojení do Internetu je potřeba zachovat určitou logiku v adaptéru, aby domácnost byla schopna fungovat i nadále. Navíc si ukládá neodeslaná data na SD kartu a při opětovném připojení odešle i zbylé hodnoty.

Ke směrovači může být adaptér připojen několika způsoby:

1. **Vestavěný do směrovače** – Vyžaduje zásah do hardwaru i do softwaru, což je většinou těžké nebo nemožné. Vyřešit se může úplnou změnou OS směrovače, ale tím se může ztratit původní funkcionalita a také není vždy změna umožněna.

2. **USB** – Nevýhodou je, že levnější směrovače stále nemají rozhraní USB a stále vyžadují zásah do softwaru směrovače.

3. **WiFi** – Nepotřebuje zásah do hardwaru ani do softwaru, ale je tu nutnost nastavení SSID sítě a případně nakonfigurování hesla k zabezpečení sítě. Řešením by mohlo být prvotní připojení přes Ethernet kabel a poté fungování přes WiFi. Nicméně tento proces značně komplikuje jednoduché zapojení pro neznalé uživatele. Oproti předešlým řešením by se také muselo přidat externí napájení adaptéru.

4. **Kabel Ethernet** – Z pohledu implementace a zapojení je tato možnost nejjednodušší. Stejně jako WiFi připojení nepotřebuje zásah do softwaru ani hardwaru a odpadá i problém s SSID a heslem. Napájení lze vyřešit buď externím zdrojem nebo přes PoE[13], což stále není standardní výbavou směrovačů. Při absenci existují PoE injektory.

Směrovač nemusí mít veřejnou IP adresu a tedy není z Internetu jednoznačně identifikovatelný. Pokud by se uživatel chtěl připojit přímo na zařízení, potřeboval by na směrovači veřejnou IP adresu, aby se mohl připojit. V dnešní době většinou uživatel dostává od poskytovatele Internetového připojení soukromé adresy. Směrovač je tedy skrytý za zařízením NAT, a tím pádem není adresovatelný. Aby byl uživatel schopný komunikovat i v tomto případě, je potřeba proxy server, který zajistí propojení se směrovačem. Server má veřejnou IP adresu a koncové ovládací prvky ho mohou jednoznačně adresovat. Využití proxy serveru přináší jednoduchost v konfiguraci.

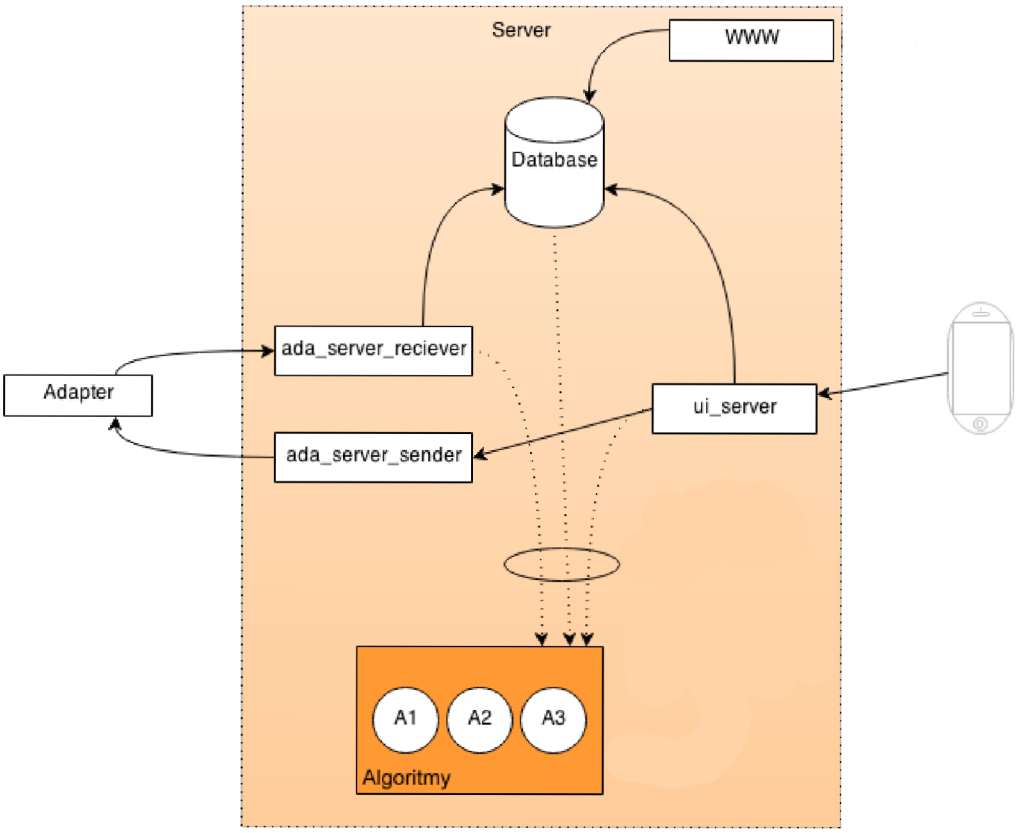
2.2.3 Server

V proxy serveru je možné ukládat veškerá data. Nad těmito daty pak lze vytvořit celou logiku a aplikace. Nicméně tohle může některé zákazníky odradit, protože nechtějí mít data o své domácnosti uložené v cloudu. Je proto třeba řešit zabezpečení dat, autentizaci, autorizaci a šifrované komunikace. Další nevýhodou tohoto přístupu je, že při výpadku Internetu je celá domácnost nedostupná. Jelikož některé části logiky jsou kritické, je třeba část logiky přemístit do adaptéru, aby byla domácnost schopna fungovat i bez připojení do Internetu.

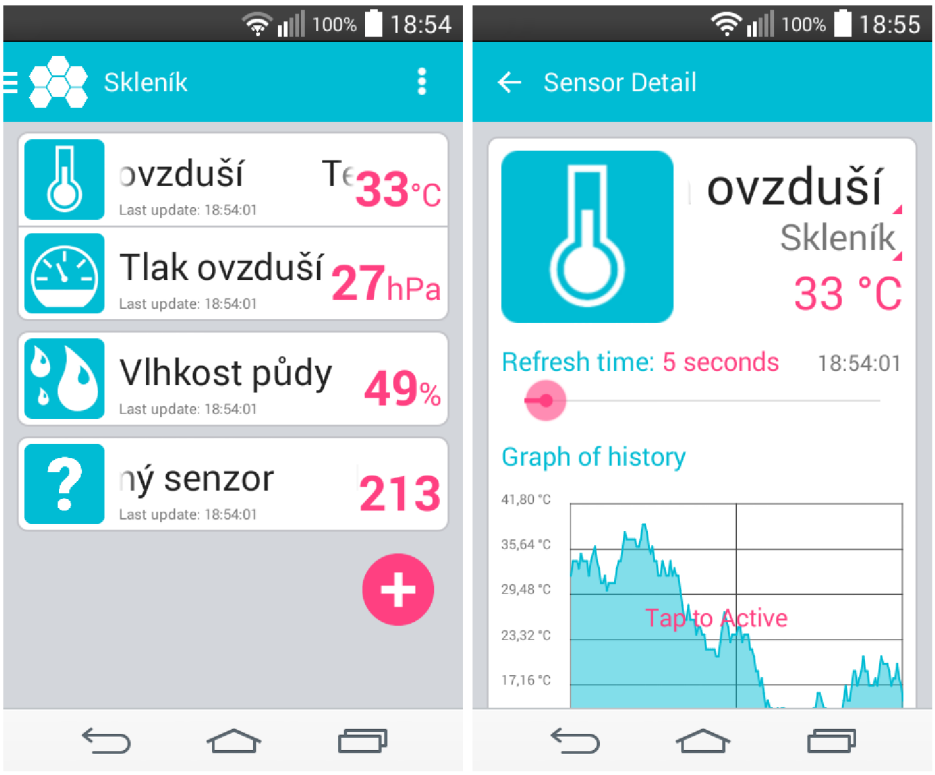
Na obrázku 2.7 je vidět, že samotný server se skládá z několika částí. Z pohledu koncových zařízení je nejdůležitější *ui_server*. Tato část serveru musí mít veřejnou IP adresu a port, ke kterému se koncová zařízení připojují. *Ui_server* přijímá požadavky od koncových zařízení, komunikuje s databází a posílá zprávy adaptéru za pomocí *ada_server_sender*. Tato část serveru se stará o zabezpečenou komunikaci s adaptérem. Adaptér přeposílá aktualizace ze senzorů na *ada_server_reciever*, který musí být taktéž adresovatelný v síti Internetu. *Ada_server_reciever* se stará o zpracování hodnot a uložení do databáze.

Algoritmy pak dostávají zprávy od *ada_server_reciever* a *ui_server* nebo si data získávají přímo z databáze. Nad daty pak vyhodnocují nadefinované podmínky nebo složitější výpočty.

[13] *Power over Ethernet*; napájení přes datový síťový kabel



Obrázek 2.7: Rozdělení jednotlivých částí serveru – převzato z Wiki projektu IoT.



(a) Seznam senzorů a aktorů (b) Detail senzoru

Obrázek 2.8: Obrazovky z Android aplikace BeeeOn

2.2.4 Koncové ovládací prvky

Uživatel ovládá celou domácnost pomocí těchto koncových zařízení. Koncové ovládací prvky představují běžně užívaná zařízení, jako třeba chytrý mobil, počítač, notebook nebo chytrá televize. Je třeba, aby komunikovaly se serverem zabezpečeně. Využívají tedy zabezpečeného přípojení TLS[14] a ověření serverového certifikátu. Uživatel se autentizuje vůči serveru a poté dostává přiřazený token, který využívá dále ke komunikaci se serverem.

Android aplikace je rozdělena do tří úrovní. Síťová vrstva zajišťuje zabezpečenou komunikaci se serverem, převádí objekty aplikace do XML[15] formátu a naopak. Dále kontrolér, který je mezivrstvou mezi síťovou a prezentační vrstvou. Stará se o dočasné uložení dat přijatých ze serveru, o správu uživatelů a zprostředkovává komunikaci do prezentační vrstvy. Ta po získání potřebných dat přehledně zobrazuje data, reaguje na různé stavy a převádí požadavky uživatele do kontroléru. Na obrázcích 2.8 jsou náhledy obrazovek.

2.3 Požadavky

Jelikož se systém navrhuje nejen pro platformu Android, musí být návrh jednoduše rozšiřitelný na ostatní mobilní platformy. Zejména to znamená mít jednoduše rozšiřitelný návrh serverové části, která je společná pro všechny platformy. Jakákoliv komunikace musí probíhat zabezpečeně, aby bylo znemožněno podvržení nebo odposlouchávání zprávy. Navržené

[14] *Transport Layer Security*; kryptografický komunikační protokol
[15] *Extensible Markup Language*; http://www.w3.org/TR/REC-xml/

řešení musí být co nejméně náročné na baterii zařízení, aby se předešlo rychlému vybíjení po nainstalovaní a používání aplikace.

Doručení notifikace na chytrý telefon musí být rychlé a spolehlivé. Každá zpráva má název a typ, podle kterého je zpracována. Na každou notifikaci musí být mobilní telefon schopný zareagovat. Pokud konkrétní notifikaci nerozpozná, zahodí ji. Musí být zaručeno, že nikdy není doručena zpráva jinému uživateli než pro koho byla původně určena. Je třeba zajistit jednoduché propojení se stávajícím serverem.

Pro notifikování serveru využijeme schopnost lokalizace chytrého telefonu. Uživateli je umožněno nadefinovat si oblasti, které později využije pro definování podmínek pro automatizaci domácnosti. Vytváření oblastí probíhá interaktivně přímo v mapě, kde jsou zobrazeny i již existující zóny. Při překročení hranice oblasti (vstup nebo výstup) je server informován o překročení hranice. Pokud server najde související podmínky, zareaguje definovanou akcí.

Kapitola 3

Notifikace mobilních aplikací

U všech mobilních platforem je možné využít procesy na pozadí a opakované dotazování serveru (tzv. *pull technologii*). Aplikace by musela mít ovšem spuštěný proces na pozadí a opakovaně v definovaných časových intervalech se musela dotazovat na server, zda nemá nová data pro aktualizaci. V praxi se tato metoda pro účel notifikací nepoužívá kvůli velké spotřebě baterie. Pokud se zvětší časový interval dotazování pro úsporu energie, pak dochází k velkým zpožděním.

Dalším univerzálním řešením pro všechny platformy je mít otevřené spojení se serverem na pozadí. Pokud to však typ aplikace nevyžaduje a není potřeba nepřetržitá komunikace, je toto řešení velmi neefektivní. Zejména se tím zvyšuje výrazně spotřeba baterie. Pokud by si své otevřené spojení udržovala každá aplikace v systému, spotřeba by byla značná.

Nejefektivnějším řešením jsou tedy technologie určené přímo pro jednotlivé platformy. Ve všech případech se jedná o push notifikace. Je to způsob asynchronní komunikace, kdy komunikaci začíná server. Výhodou tohoto přístupu je nízká náročnost na baterii, jelikož si neotevírá každá aplikace své spojení, ale systém si udržuje pouze jedno spojení s notifikačním serverem, přes které komunikují všechny aplikace v systému. Pokud využíváme push notifikace, ušetříme i množství přenesených dat oproti dotazování. Při dotazování je většina zpráv poslaná zbytečně, protože server při většině dotazů na nová data odpovídá záporně. Tento zbytečný objem dat roste s požadavkem na menší zpoždění mezi odesláním zprávy a doručením (maximální doba pro doručení je doba mezi dvěma dotazy na server), protože musíme zvyšovat frekvenci dotazování. S tím stoupá i zátěž baterie. U push notifikací je doručení téměř okamžité.

Jelikož se tato diplomová práce zabývá řešením pro platformu Android, zaměří se na to, jaké jsou možnosti pro tuto platformu, jak fungují a co je potřebné vědět pro návrh a implementaci. U zbylých dvou majoritních platforem (iOS, Windows Phone) se práce zaměří pouze na princip fungování, aby byla notifikační služba navrhnuta dostatečně obecně a umožnila lehkou rozšířitelnost na další platformy.

3.1 Google Cloud Messaging (GCM)

Google Cloud Messaging je služba, kterou zajišťuje společnost Google a která pomáhá vývojářům odesílat a přijímat data ze svých serverů do zařízení Android a do aplikací v prohlížeči Chrome. Tato služba je bezplatná.

První zavedení do systému Android došlo ve verzi 2.2 Froyo. Služba se zde nenazývala ještě Google Cloud Messaging, ale nesla název *Android Cloud to Device Messaging* (C2DM).

Převedení na službu Google Cloud Messaging došlo na Google I/O[1] v roce 2012. Rozšíření pro prohlížeč Chrome představila společnost na Google I/O v roce 2013.

Pomocí GCM lze posílat jednoduché zprávy o maximální velikosti 4KB. Z tohoto důvodu lze rozdělit zprávy na dva typy podle sémantiky.

Prvním typem je synchronizační zpráva, která oznámí aplikaci, že má na serveru přichystaná nová data ke stažení. Samotná zpráva tedy nenese žádná aktuální data, ale jen žádá aplikaci o stažení nových dat ze serveru. Tento přístup je zejména vhodný v případě, kdy aktualizace obsahuje velké množství dat, která by se do GCM zprávy nevešla. V případě potřeby můžeme tento princip využít i pro přenos tajných informací, kdy odstraníme z komunikace službu třetí strany. Pošle se tedy opět pouze požadavek na stažení tajné informace a samotný přenos pak proběhne po zabezpečeném kanále přímo mezi aplikací a serverem.

Druhým typem je zpráva, která již nese samotná data. Tento přístup je vhodný pouze pro přenos dat, kde lze zajistit maximální velikost dat 4KB.

Cílová aplikace nemusí být vůbec spuštěná ani na pozadí. Systém probudí aplikaci pomocí *Intent Broadcast*[2], když obdrží novou zprávu adresovanou aplikaci. Musí však být správně nastaveno povolení v *Manifestu*[3] aplikace a *Broadcast Reciever*[4]. Samotné GCM neposkytuje žádné rozhraní pro automatické zpracování zpráv. Vývojáři poskytne dvojice jméno a hodnotu. Je tedy v plné režii vývojáře, jestli například zobrazí zprávu do notifikační lišty, změní nastavení v aplikaci nebo jen bez vědomí uživatele na pozadí sesynchronizuje data.

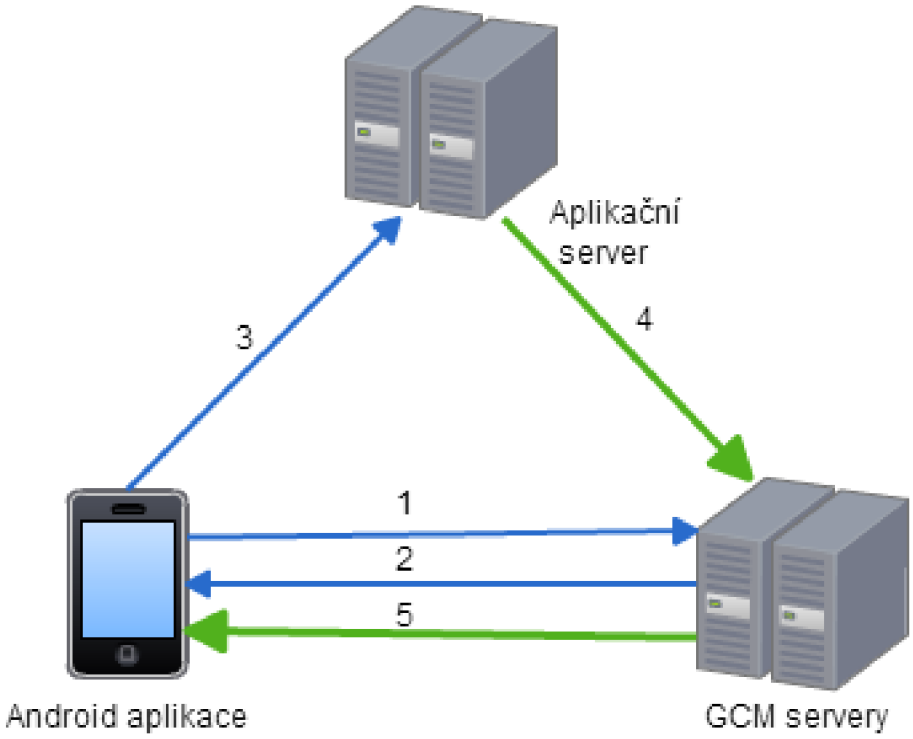
Na obrázku 3.1 je znázorněno schéma komunikace. Prvním prvkem je **Android aplikace**, která má povolené GCM. Musí to být zařízení s Androidem verze 2.2 nebo vyšší a nainstalovaným *Google Play Services*. Pokud je verze Androidu nižší než 4.0.4, musí mít uživatel přihlášený alespoň jeden Google účet na svém zařízení. Od vyšších verzí to není podmínkou. Další důležitou částí je **aplikační server**, který chce komunikovat s aplikací. Tato část je v plné režii vývojáře. Známe jen rozhraní komunikace s GCM serverem a zbytek je v režii programátora. Posledním prvkem je **GCM server** poskytnutý Googlem, který přijímá zprávy od aplikačního serveru, skladuje a přeposílá zprávy do Android zařízení (respektive do konkrétní aplikace).

Komunikace mezi aplikačním serverem, GCM server a mobilním zařízením je zabezpečená. Nicméně GCM server má přístup k datům v otevřené podobě. Řešením je buď šifrování dat (identifikátorů i hodnot nebo alespoň hodnot). Nebo neposílat citlivá data přes tento kanál a poslat pouze synchronizační zprávu. Je potřeba si ale uvědomit, že se jedná o krajní případ, kdy by musel samotný Google data na serveru zneužít nebo někdo kompromitovat GCM server. Odposlechnutí mezi serverem a koncovým zařízením není možné.

Vysvětlení identifikátorů:

- **Sender ID** – Číslo projektu, které je získáno z *Google Developers Console*[5]. Používá se v procesu registrace pro identifikování našeho serveru.
- **Application ID** – Android aplikace, která žádá o registraci pro příjem zpráv, je identifikovaná pomocí jména balíčku v Manifestu. Tím je zaručeno, že zprávy jsou

[1] každoroční konference pořádaná společnosti Google v San Franciscu
[2] http://developer.android.com/guide/components/intents-filters.html
[3] http://developer.android.com/guide/topics/manifest/manifest-intro.html
[4] http://developer.android.com/reference/android/content/BroadcastReceiver.html
[5] https://console.developers.google.com



Obrázek 3.1: Schéma Google Cloud Messaging komunikace

adresovány správné aplikaci. Pro vývojáře transparentní.

- **Registration ID** – Identifikátor, který užívá GCM server. Jeden identifikátor se váže k jedné konkrétní aplikaci na konkrétním zařízení. Musí být drženo v tajnosti. Maximální velikost může být až 4KB. Nicméně v dnešní době se velikost přidělených identifikátorů pohybuje kolem 200 znaků. Do budoucna se ale bude délka identifikátoru zvětšovat.
- **Sender Auth Token** – API klíč, který si uchovává aplikační server. Google servery podle toho autorizují přístup ke službám.

3.1.1 Postup zasílání notifikací

Na obrázku 3.1 jsou vidět dvě různé komunikace (modrá a zelená). Postup při komunikaci je následující:

1. **uložení *registration ID* na server** (modrá komunikace)
   (a) Pokud má aplikace přístup k Internetu, žádá GCM server o přidělení *registration ID*.
   (b) GCM server přidělí *registration ID* aplikaci.
   (c) Aplikace si *registration ID* uloží do interní perzistentní paměti, aby příště věděla, že je zaregistrovaná a neptala se zbytečně opětovně GCM serveru.
   (d) Aplikace odešle *registration ID* na aplikační server.
   (e) Aplikační server si uloží *registration ID* do databáze.

2. **poslání zprávy na Android zařízení** (zelená komunikace) – předpokládá se validní provedení bodu 1

   (a) Pokud chce aplikační server kontaktovat Android zařízení, najde si požadované *registration ID* (nebo seznam více identifikátorů).

   (b) Pomocí jednoho z protokolů ze sekce 3.1.2 pošle požadavek na GCM server.

   (c) GCM server odpoví, zda se podařilo zprávu pro příjemce zpracovat, případně pro které ne.

   (d) Pokud je cílové zařízení dostupné, GCM server doručí zprávu ihned. Pokud se nepodaří doručit ihned, uloží ji na serveru. Pomocí atributu ve zprávě se může nastavovat doba, po kterou GCM server udržuje zprávu při neúspěšném okamžitém doručení (více v sekci 3.1.2). Pomocí atributů lze zajistit i sdružování zpráv apod.

   (e) Zařízení přijme zprávu. Systém ji přesměruje do cílové aplikace pomocí *Intent broadcast* s příslušným povolením. Tím zajistí, že žádná jiná aplikace není schopna zprávu přečíst. Aplikace zpracuje zprávu.

3.1.2 Protokoly pro komunikaci s GCM serverem

Pro předání zprávy z aplikačního serveru na GCM server můžeme použít protokol HTTP[6] nebo XMPP[7]. Lze nasadit jen jeden z nich nebo oba najednou. Pro předání informací využívají formát JSON[8].

Protokol HTTP poskytuje synchronní, jednosměrné propojení. U tohoto protokolu je možné využít mimo JSON formátu i obyčejný text. Odeslání zprávy se provádí pomocí požadavku POST na GCM server.

Protokol XMPP zajišťuje trvalé, asynchronní, obousměrné připojení s GCM serverem. JSON data jsou zaobaleny v XMPP zprávě. Jelikož je komunikace obousměrná, může i Android komunikovat se serverem po stejném komunikačním kanále.

HTTP

GCM HTTP server poskytují synchronní, jednosměrné (cloud-to-device) připojení. Kromě reprezentace dat v JSON formátu lze použít i plain text.

Odeslání zprávy se provádí pomocí požadavku POST na adresu `https://android.googleapis.com/gcm/send`. Zpráva se skládá z HTTP hlavičky a těla. Hlavička musí obsahovat `Authorization: key=SENDER_AUTH_TOKEN` a typ reprezentace dat (JSON nebo plain text). Pro JSON `Content-Type: application/json` a pro plain text `application/x-www-form-urlencoded;charset=UTF-8`. Pokud je tento údaj vynechán, implicitní je plain text. Dále uvažujeme prezentaci dat pomocí JSON formátu.

V těle HTTP požadavku musí být cíl, komu zprávu chceme zaslat, definovaný jedním ze způsobů:

- `registration_ids` – Seznam *registration ID* oddělených čárkami. Tímto způsobem je možno odeslat jednu zprávu na jedno až tisíc zařízení. Pokud se odesílá zpráva na více zařízení než na jedno, nazývá se multicastovou zprávou.

[6] *Hypertext Transfer Protocol*; http://www.ietf.org/rfc/rfc2616.txt
[7] *Extensible Messaging and Presence Protocol*; http://www.ietf.org/rfc/rfc3920.txt
[8] *JavaScript Object Notation*; http://www.ietf.org/rfc/rfc4627.txt

- `notification_key` – Slouží pro odesílání zprávy více zařízením, které vlastní jeden uživatel. GCM server zasílá *notification key*, který zapouzdří všechny *registration ID* jednotlivých zařízení. Uchovává se tedy jen jeden klíč a aplikační server si nemusí pamatovat všechny *registration ID* všech zařízení, ale pamatuje si je GCM server. S ním komunikujeme pomocí tří zpráv: vytvoř nový *notification key*, přidej do stávajícího *notification key* nový *registration ID* a nebo odstraň *registration ID* z *notification key*.

HTTP požadavek bez nepovinných parametrů tedy vypadá následovně:

```
Content-Type:application/json
Authorization:key=AIzaSyB-1uEai2WiUapxCs2Q0GZYzPu7Udno5aA
{
  "registration_ids" : ["REGISTRATION_ID_1", "REGISTRATION_ID_2", ...],
  "data" : {
    ...
  },
}
```

Statusy odpovědi:

- **200** – Zpráva byla zpracována.
- **400** – Chyba při zpracování JSON formátu.
- **401** – Chyba při autentizaci.
- **5xx** – Interní chyba na GCM serveru nebo dočasně nedostupný.

Pokud tedy přijatá zpráva obsahuje status kód 200, pak se jedná o zprávu o úspěšném přijetí požadavku. Neznamená to však, že byla zpráva zpracována a doručena. Tyto informace obsahuje až odpověď. Tělo zprávy může vypadat například takto:

```
{
  "multicast_id": 216,
  "success": 3,
  "failure": 3,
  "canonical_ids": 1,
  "results": [
    { "message_id": "1:0408" },
    { "error": "Unavailable" },
    { "error": "InvalidRegistration" },
    { "message_id": "1:1516" },
    { "message_id": "1:2342", "registration_id": "32" },
    { "error": "NotRegistered"}
  ]
}
```

Vysvětlení jednotlivých elementů:

- `multicast_id` – Unikátní ID identifikující multicast zprávu.
- `success` – Počet zpráv, které byly úspěšně zpracovány.

- `failure` – Počet zpráv, které nebyly zpracovány.
- `canonical_ids` - Počet *canonical ID*, které musí být nahrazeny.
- `results` – Pole JSON objektů, které jsou seřazeny ve stejném pořadí jako odeslané *registration ID*. Reprezentují status zprávy.
  - `message_id` – Řetězec reprezentující zprávu, která byla úspěšně zpracována.
  - `registration_id` – Aktuálnější *registration ID*, které musí být použito jako náhrada za původní identifikátor. Více v sekci 3.1.3.
  - `error` – Řetězec popisující chybu, která nastala při zpracování.

Pokud hodnoty `failure` i `canonical_ids` jsou 0, nemusí se zpráva dále zpracovávat. Jinak se prochází následovně:

1. Pokud je nastaveno `message_id`, zkontroluje se, zda neobsahuje i `registration_id`. Pokud ano, nahradí se s ním původní ID.
2. Jinak se zkontroluje hodnota `error`.
   - `Unavailable` – GCM server je zaneprázdněn, zkusí se opětovně odeslat později.
   - `NotRegistered` – Aplikace je již odinstalovaná nebo má špatný identifikátor. Musí se smazat z databáze.
   - `MissingRegistration` – Nebylo zadané ani jedno *registration ID*.
   - `InvalidRegistration` – Neplatný identifikátor. Zkontroluje se formátování *registration ID*.
   - `MismatchSenderId` - *Registration ID* je spojené s povolenými odesílateli. Došlo k odeslání z nepovoleného zdroje.
   - `Unregistered Device` – Aplikace se odregistrovala z GCM serveru, vypršela platnost *registration ID* nebo na zařízení není nakonfigurován *broadcast reciever*.
   - `MessageTooBig` – Zpráva překročila hranici 4096 bajtů.
   - `InvalidDataKey` – V datové části požadavku bylo použito rezervované slovo, které je využívané GCM serverem.
   - `InvalidTtl` – Porušené požadavky na číslo v rozmezí 0 až 2419200.
   - `InternalServerError` – Chyba serveru.
   - `DeviceMessageRateExceeded` – Překročen limit zpráv na zařízení v určitém čase.
   - `InvalidPackageName` – Zpráva byla adresována *registration ID*, které se neshoduje se zadaným jménem balíku.

XMPP

GCM Cloud Connection Server (CCS) poskytuje trvalé, asynchronní, obousměrné připojení na GCM server. Pomocí XMPP se posílají i přijímají zprávy mezi aplikačním serverem a propojenými zařízeními.

Výhodou tohoto přístupu je asynchronní povaha XMPP, která umožňuje posílat více zpráv s menším nárokem na zdroje. Komunikace je obousměrná, takže i Android zařízení

může kontaktovat server. To lze zejména uplatnit tehdy, když chceme oboustrannou komunikaci mezi dvěma zařízeními a ani jedno nemá veřejnou IP adresu. Například se tedy jedná o komunikaci z chytrého zařízení na jiné chytré zařízení. Další výhodou tohoto přístupu je, že zařízení může poslat zprávu pomocí stejného připojení pro příjímání zpráv. Tím šetří baterii oproti tomu, když si otevírá další připojení.

Pro připojení pomocí XMPP je třeba splnit požadavky. Klient musí zahájit komunikace pomocí *Transport Layer Security* (TLS). CCS vyžaduje *PLAIN Simple Authentication and Security Layer* (SASL)[9]. Využívá se odkaz `<Sender_ID>@gcm.googleapis.com` a heslo je *Sender Auth Token*.

Jakmile je spojení navázáno, využívají se XMPP tagy `<message>` pro zapouzdření JSON zprávy. Samotná JSON zpráva je stejná jako u HTTP protokolu až na následující výjimky. Nepodporuje více příjemců v jedné zprávě a místo parametru `registration_ids` je použit parametr `to`. Dále může využívat ještě parametr `message_type`, který určuje typ zprávy. Může nabývat hodnot "ack", "nack"nebo "control". Navíc CCS přidává povinný parametr `message_id`, který jednoznačně identifikuje zprávu v daném XMPP připojení. ACK a NACK zprávy využívají tento parametr.

Požadavek na server bez nepovinných parametrů tedy vypadá následovně:

```
<message id="">
  <gcm xmlns="google:mobile:data">
  {
    "to":"REGISTRATION_ID",
    "message_id":"MESSAGE_ID",
    "data": {
      ...
    },
  }
  </gcm>
</message>
```

Parametry zpráv

V následující tabulce jsou použity zkratky pro zmenšení rozměrů a tedy přehlednost.
Prot. = Protokol, C/H = CCS i HTTP, P = Povinný parametr

Prot.	**Parametr**	**Popis**	**P**
HTTP	`registration_ids`	Pole *registration ID* v počtu 1 až 1000.	✓
HTTP	`notification_key`	Mapuje několik *registration ID* vlastněných jedním uživatelem do jednoho identifikátoru. Maximum mapovaných identifikátorů je 20.	×
C/H	`collapse_key`	Určuje řetězec, podle kterého jsou shlukovány zprávy, pokud je zařízení offline. Zařízení dostane vždy jen poslední přijatou zprávu. Často nazývané *send-to-sync* zprávy. Maximálně čtyři klíče na zařízení v jeden čas.	×

[9] http://www.ietf.org/rfc/rfc4616.txt

HTTP	`restricted_package_name`	Řetězec reprezentující jméno balíku v Manifestu.	×
HTTP	`dry_run`	Testování pro vývojáře bez doručení na koncové zařízení. Implicitně false.	×
C/H	`data`	Specifikuje JSON objekt, který obsahuje dvojice klíč-hodnota. Celková velikost dvojic musí být menší než 4kb. Doporučuje se používat řetězce, protože na GCM serveru se stejně převádí vše do řetězců. Nesmí obsahovat názvy, které jsou použity v této tabulce, nebo názvy začínající na `google.`	×
C/H	`delay_while_idle`	Pokud je `true`, server čeká se zasláním, až je zařízení v aktivním stavu. Tzn. pokud je například mobil zamčený, zpráva je doručena až po odemčení. Implicitně false.	×
C/H	`time_to_live`	Sekundy, jak dlouho je zpráva uchována na GCM serveru, pokud je zařízení offline. Hodnota 0 až 2419200 sekund. Implicitní je maximum, tzn. 4 týdny.	×
CCS	`to`	Nahrazuje `registration_ids` z HTTP.	✓
CCS	`message_id`	Jednoznačná identifikace zprávy v rámci XMPP spojení.	✓
CCS	`message_type`	Typ zprávy. Nabývá hodnot "ack", "nack"nebo "control". Z toho první dvě použijeme i na našem serveru při potvrzování GCM serveru.	×
CCS	`delivery_receipt_requested`	Požadavek na potvrzení přijetí zprávy. Až zařízení potvrdí přijetí, CCS pošle potvrzení serveru. Implicitně false.	×
CCS	`message_status`	Nachází se uvnitř parametru `data`. Jediná možná hodnota je `MESSAGE_SENT_TO_DEVICE`, která indikuje přijetí zprávy na zařízení.	
CCS	`original_message_id`	Nachází se uvnitř parametru `data`. ID originální zprávy, kterou GCM server poslal na zařízení.	
CCS	`device_registration_id`	Nachází se uvnitř parametru `data`. Seznam *registration ID*, na které byla zpráva odeslána.	

Tabulka 3.1: Parametry zprávy zasílané na GCM server

3.1.3 Canonical registration ID

Pokud nedojde k opakovaným registracím *registration ID*, nenastane žádný problém. Nicméně při aktualizaci aplikace nebo celého systému je nutné zažádat o nové *registration ID*. V tomto případě může dojít k nekonzistenci identifikátoru registrovaného v GCM s identifikátorem uloženým na aplikačním serveru.

GCM server pro tento případ poskytuje službu zvanou *canonical registration IDs*, která umožňuje tuto situaci vyřešit. *Canonical registration ID* je identifikátor poslední registrace zařízení na GCM serveru. Když tedy aplikační server pošle zprávu na *registration ID*, které už není aktuální, GCM server i přesto dokáže doručit zprávu. Avšak v odpovědi od GCM serveru dostaneme v poli `registration_id` aktuální validní *registration ID*. Tímto novým identifikátorem musíme přepsat identifikátor v databázi, aby příští požadavek již byl poslán na správné *registration ID*.

3.1.4 Odregistrování ze služby

Jelikož *registration ID* adresuje konkrétní aplikaci na konkrétním zařízení, nesmí se pomocí odregistrování řešit situace, kdy aplikaci používá více lidí (tzn. více účtů). *Registration ID* není nijak spojené s aktuálně přihlášeným uživatelem. Zde nám odregistrování nepomůže a pravděpodobně (není to zaručeno) při příští registraci bude přiděleno stejné *registration ID*. Tuto situaci lze nejlépe řešit pomocí odstranění *registration ID* z aplikačního serveru při odhlášení uživatele. Při následném přihlášení uživatele se již jen pošle uložené *registration ID* na aplikační server a ihned může server komunikovat se zařízením.

Situací, kdy je potřeba odregistrovat *registration ID* přímo z GCM serveru, je málo. Jeden z důvodů odregistrace může být situace, kdy je podezření, že *registration ID* bylo zkompromitováno. Odregistrování lze provést i v případě, když je nutné úplně zakázat zprávy z GCM. V obou případech musíme brát v potaz, že odregistrování může trvat až 5 minut kvůli propagaci změn na všechny GCM servery. Opětovná registrace po odregistrování může trvat až dalších 5 minut. Během této doby mohou GCM servery odmítat požadavky na zaslání zpráv.[6]

K odregistrování aplikace musí dojít také při odinstalování aplikace. Nicméně Android neposkytuje callback při odinstalování aplikací, takže na tuto událost nelze reagovat. *Registration ID* tedy zůstane aktivní na GCM serverech i na aplikačním serveru.

Scénář při odinstalování aplikace je tedy následující[6]:

1. Uživatel odinstaluje aplikaci, která je zaregistrována u GCM serveru pro příjem zpráv.
2. Aplikační server odešle požadavek na zaslání zprávy.
3. GCM server přeposílá zprávu na zařízení.
4. GCM klient[10] přijme zprávu a pomocí *Package Manageru* zjistí, zda existuje alespoň jeden *Broadcast receiver*, který je schopný přijmout tuto zprávu. Zjistí, že nikoliv.
5. GCM klient informuje GCM server, že aplikace byla odinstalována.
6. GCM server označí *registration ID* k odstranění.
7. Aplikační server posílá další požadavek na zaslání zprávy.

[10] GCM framework na Android zařízení

8. GCM server vrací chybu `NotRegistred`.

9. Aplikační server si musí *registration ID* odstranit z databáze.

3.2 Apple Push Notification service (APNs)

APNs je služba zajišťující push notifikace pro iOS a OS X. Každé koncové zařízení navazuje se službou šifrované spojení a přes toto stálé spojení přijímá notifikace.

Apple poprvé tuto službu představil v iOS 3.0 v červnu roku 2009. V iOS 5 změnil Apple pracování s lokálními i push notifikacemi pomocí nástroje *Notification Center*. Od Mac OS X v10.7 "Lion"najdeme APNs i v ostatních zařízení od Apple. Od října roku 2014 je jakékoliv propojení s APNs uskutečněno pomocí TLS.[15]

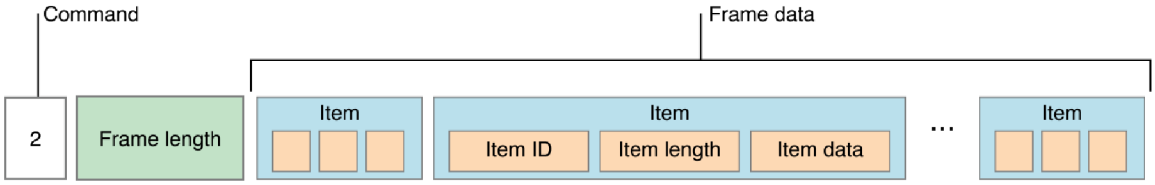
Princip je velice podobný jako u GCM (viz sekce 3.1). Pouze místo *registration ID* se zde využívá pojmenování *device token*. Každá notifikace pro iOS 8 a novější musí mít maximálně 2kb. Pro starší systémy než iOS 8 a OS X je maximum 256 bajtů. Data jsou reprezentována ve formátu JSON. Každá notifikace musí obsahovat JSON klíč `aps` s objektem. V `aps` je obsaženo jedno nebo více vlastností, které specifikují typ:

- `alert` – Zpráva, která je zobrazena uživateli. Buď řetězec nebo další JSON objekt, který specifikuje jména proměnných ve slovníku nebo definuje název tlačítka.
- `badge` – Číslo, které je zobrazeno na štítku v ikoně aplikace. Pro zrušení předchozích štítků je třeba nastavit číslo na 0. Pokud není definováno, ikona zůstává nezměněna.
- `sound` – Řetězec, který nese název zvuku, který se po přijetí přehraje. Zvuk musí být umístěný v aplikaci. Pokud není nalezen, přehraje se implicitní zvuk upozornění.
- `content-available` – Pokud je hodnota 1, jedná se o tichou notifikaci. Uživateli se nezobrazí nic a na pozadí se například pouze sesynchronizují data.

Mimo klíč `aps` si může vývojář přidat svoje definované proměnné.
Příklad JSON:

```
{
  "aps" : {
    "alert" : "Nový email!",
    "badge" : 3,
    "sound" : "audio.aiff"
   },
   "sync" : true,
   "email_id" : 123456
}
```

Pokud není aplikace zapnutá, zobrazí zprávu, přehraje zvuk nebo připne štítek k ikoně. Na předchozím příkladu je ukázána kombinace všech možností notifikace uživatele. Při přijetí této zprávy se přehraje zvukový záznam s názvem *audio.aiff* (musí být lokálně dostupný v zařízení), u ikony aplikace se zobrazí značka s číslem tři a uživateli se objeví upozornění v dialogovém okně. Navíc jsou ve zprávě obsažené dvě volitelné proměnné. První nese název *sync* a může být využit jako příznak pro synchronizaci dat ze serveru. Druhá proměnná se jménem *email_id* předává identifikaci nového emailu. Pokud je aktuálně aplikace na popředí, systém notifikaci deleguje do aplikace.



Obrázek 3.2: Formát požadavku na notifikaci (MPNs) – převzato ze zdroje [2]

ID	Název	Délka	Popis
1	*Device token*	32 B	*Device token*, který byl zaregistrovaný zařízením na APNs. V binární podobě.
2	JSON data	různá	Samotná JSON data, která jsou popsána výše.
3	ID notifikace	4 B	Jednoznačná identifikace notifikace. Nepovinné.
4	Datum expirace	4 B	UNIX formát data v sekundách (UTC), kdy může být notifikace vymazána ze serveru APNs. Pokud je 0, neukládá notifikaci vůbec.
5	Priorita	1 B	Pokud je 10, notifikace je odeslána ihned. Pokud je 5, notifikace čeká na nejlepší čas vzhledem k úspoře baterie.

Tabulka 3.2: Přehled položek *item data* v notifikační zprávě APNs

Produkční APNs naslouchá na adrese `gateway.push.apple.com` a portu 2195. Ke komunikaci využívá binární protokol zobrazený na obrázku 3.2. *Command* je jeden bajt, který nabývá vždy hodnoty dva. Další čtyři bajty nesou velikost následujícího bloku jednotlivých položek. Každá položka *item data* je složena z identifikátoru, velikosti dat a samotných dat. Přehled položek *item data* je popsán v tabulce 3.2.

Pokud APNs správně zpracuje požadavek, žádná zpráva není navrácena. V opačném případě je nutné rozeznat chybu a zareagovat na ni podobně jako je popsáno u GCM v sekci 3.1.2.[1]

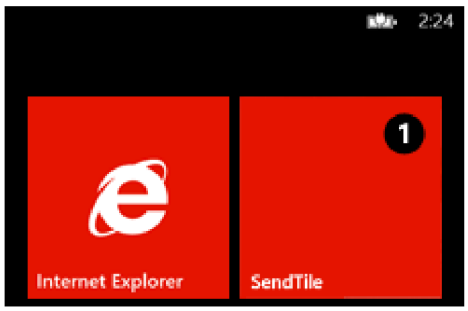
3.3 Microsoft Push Notification Service (MPNS)

MPNS je asynchronní služba pro push notifikace pro Windows Phone (WP). Princip je opět stejný jako u GCM (viz sekce 3.1). Na rozdíl od APNs a GCM nedostává identifikátor, ale přímo URI[11]. Na tuto adresu se požadavek odesílá. Adresace zařízení ze strany serveru tedy probíhá přímo zasláním požadavku na uloženou URI oproti předešlým metodám, kde se vše posílalo na jednu adresu a až obsah zprávy adresoval zařízení.

Můžeme vytvořit tři typy notifikací[11]:

- **Toast** – Krátká zpráva, která se zobrazí uživateli v notifikační liště. Kliknutí na notifikaci pak způsobí přesměrování do aplikace. Pokud je aplikace, pro kterou je notifikace určena, zapnutá, pak se do notifikační lišty nezobrazuje a je přímo předána do aplikace.
- **Tile** – Tile notifikace mění vzhled dlaždice hry nebo aplikace, kterou má uživatel připnutou na hlavní obrazovce. Je možné zde specifikovat nápis, číslo (například počí-

[11] Uniform Resource Identifier



Obrázek 3.3: Ukázka *tile* notifikace na WP doručená přes MPNS – převzato ze zdroje [12]

tadlo nových zpráv) a pozadí dlaždice (obrázek může být lokální i externí). Jestli je aplikace aktuálně zapnutá nebo ne, nemá vliv na zpracování notifikace.

- **Raw** – Podobné předchozím, avšak nemá předdefinované chování a formát zprávy. Je v plné režii vývojáře, aby aplikace nebo hra rozuměla zprávám zaslaných ze serveru a uměla na ně patřičným způsobem reagovat.

U každé z typů můžeme definovat interval dávkování v hlavičce požadavku pomocí parametru `X-NotificationClass`. Jsou tři typy: ihned, do 450 sekund a nebo do 900 sekund.

Můžeme pracovat ve dvou režimech[11]:

- **Neautorizovaný** – V tomto režimu je počet a frekvence zasílaných notifikací omezena. Aktuálně to je 500 notifikaci za den pro jeden kanál.
- **Autorizovaný** – Tento režim vyžaduje po vývojáři registraci certifikátu přes Windows Phone Marketplace. Certifikát musí být vydán důvěryhodnou Microsoft kořenovou certifikační autoritou. Tento certifikát se následně využije pro bezpečné připojení mezi aplikačním serverem a MPNS. Autorizované notifikace nejsou omezeny v počtu ani ve frekvenci.

Žádosti na MPNS jsou posílány pomocí POST požadavků na přidělenou adresu. Data jsou reprezentována v XML formátu. Příklad požadavku na *tile* notifikaci:

```
<?xml version="1.0" encoding="utf-8"?>
<wp:Notification xmlns:wp="WPNotification">
  <wp:Tile>
    <wp:BackgroundImage>cervena.jpg</wp:BackgroundImage>
    <wp:Count>1</wp:Count>
    <wp:Title>SendTile</wp:Title>
    <wp:BackBackgroundImage>modra.jpg</wp:BackBackgroundImage>
    <wp:BackTitle>BackNazev</wp:BackTitle>
    <wp:BackContent>Ahoj svete!</wp:BackContent>
  </wp:Tile>
</wp:Notification>
```

Tato notifikace způsobí zobrazení červené dlaždice (viz obrázek 3.3) se značkou jedna a nápisem "SendTile". Od verze Windows Phone 7.5 lze specifikovat i druhou stranu dlaždice. Tyto parametry jsou tedy nepovinné. Lze zde definovat barvu dlaždice, nadpis a delší popis.

Kapitola 4

Geolokační oblasti

Geolokační zóny neboli geofencing je vymezení geografické oblasti na zóny. Tvoří se tedy virtuální bariéry v mapě a při překročení některé z nich se může vyvolat nějaká akce.

Tato kapitola se podrobně věnuje geolokaci pro platformu Android. Součástí *Google Play Services* je *Location APIs*[7], které usnadňuje práci pro vytváření aplikací založených na lokalizaci chytrého zařízení. *Location APIs* odstiňuje vývojáře od použité technologie pro lokalizaci. Není tedy potřeba definovat, zda zařízení musí čerpat lokalizační data pomocí mobilní sítě, Wi-Fi signálu nebo GPS modulu. Naopak se na základě požadavků na přesnost a spotřebu baterie vybere aktuálně dostupná a nejvhodnější technologie. Využije se tedy aktuálně nejvýhodnější možnost.

V dřívějších verzích musel vývojář definovat, jakou lokalizační techniku chce využít pro lokalizaci. V aktuální verzi 6.5 se pouze specifikuje, jak přesná má být poloha na úkor baterie. Pokud není vyžadováno přesné určení polohy, je dopad na baterii minimální.

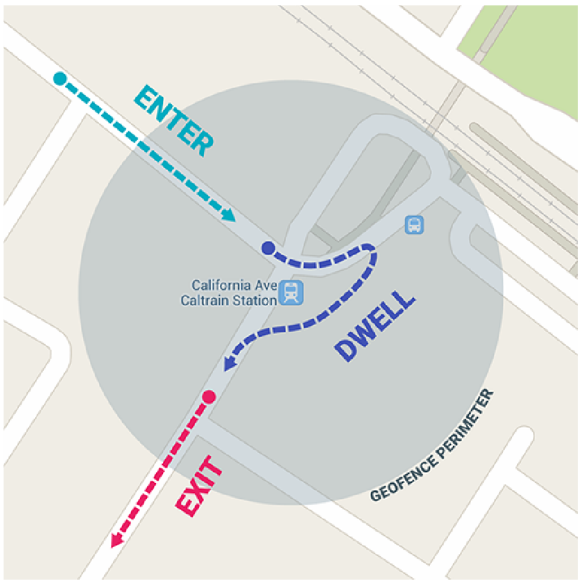
Součástí nové verze je také rozpoznání aktuální aktivity člověka. Rozlišuje čtyři stavy, a to v klidu, chůze, na kole a v autě. Využívá k tomu akcelerometr, který je z pohledu náročnosti na baterii minimální.

Detekci aktivity využívá i lokalizační služba. Jelikož má informaci o tom, zda uživatel stojí na místě, chodí, jezdí na kole nebo jede v autě, může tak přizpůsobit frekvenci dotazů na polohu. Například automobilní navigace má informaci o tom, že uživatel se nachází v autě a pohybuje se tedy rychleji než při chůzi, a proto se musí na aktuální polohu ptát častěji.

Nová verze využívá efektivněji WiFi sítě pro lokalizaci a v kombinaci s ostatními technologiemi dosahuje mnohem lepších výsledků v lokalizaci uvnitř budov než u starší verze.

Pro definování kritérií určení polohy je možné využít následující priority[8]:

- `PRIORITY_BALANCED_POWER_ACCURACY` – Při tomto nastavení nejpravděpodobněji použije lokalizaci pomocí WiFi a BTS triangulace. Přesnost odpovídá městskému bloku, tj. přibližně 100m.
- `PRIORITY_HIGH_ACCURACY` – Nejpřesnější určení místa. Nejpravděpodobněji bude použita GPS. Při zapojení GPS do výpočtu pozice dochází k velké spotřebě baterie.
- `PRIORITY_LOW_POWER` – Detekuje pouze přibližné určení polohy na úrovni měst. Tato priorita spotřebuje nejméně baterie.
- `PRIORITY_NO_POWER` – Nemá přímý vliv na spotřebu baterie vlivem určování pozice, protože aplikace dostane aktualizaci polohy jen tehdy, pokud si jiná aplikace v systému



Obrázek 4.1: Typy událostí při geofencingu – převzato ze zdroje[9]

vyžádá polohu. Sama aplikace si tedy aktivně o polohu nežádá. Je vhodná v situacích, kdy určení polohy není kritickým místem aplikace a kdy není potřeba v určitý moment mít aktuální pozici.

Uživatel si v aplikaci může definovat oblasti, které lze využít na navázání dalších akcí. Jedna oblast je definována přesnou GPS souřadnicí a rádiusem. Jeden uživatel může mít definován více oblastí. Maximum aktivních oblastí však je sto. Dle specifikace [9] existují tři typy přechodů, které jsou znázorněny v obrázku 4.1. První přechod *ENTER* znamená vkročení do definované zóny. Stane se tak ihned po vstupu do oblasti, pokud je poloha dostatečně přesná, aby bylo spolehlivě určeno, zda se uživatel nachází v dané oblasti. Druhý typ *DWELL* navazuje na předešlý přechod. Aktivuje se po vkročení do zvolené oblasti, ale až po uplynutí požadované doby. Zajistíme tím, že se uživatel v oblasti již nachází nějakou dobu. Poslední typ *EXIT* nastává při opuštění oblasti.

Jelikož uživatel bude interaktivně definovat zóny v mapě, potřebuje mít mapové podklady. K tomu slouží služba *Google Maps Android API v2*, která se také nachází v *Google Developers Console*.

Kapitola 5

Návrh

Cílem této kapitoly je navrhnout způsob pro vzájemnou notifikaci při událostech generovaných na serveru či telefonu s využitím v chytré domácnosti. Návrh musí zohlednit spotřebu baterie chytrého telefonu a veškerý přenos mezi telefonem a serverem musí být šifrovaný.

Od této kapitoly se práce zaměřuje pouze na platformu Android a znalosti o ostatních platformách jsou využity pro efektivní a obecný návrh, který umožňuje jednoduché rozšíření. Návrh pro server i mobilní platformu je objektově orientovaný.

5.1 Google Cloud Messaging

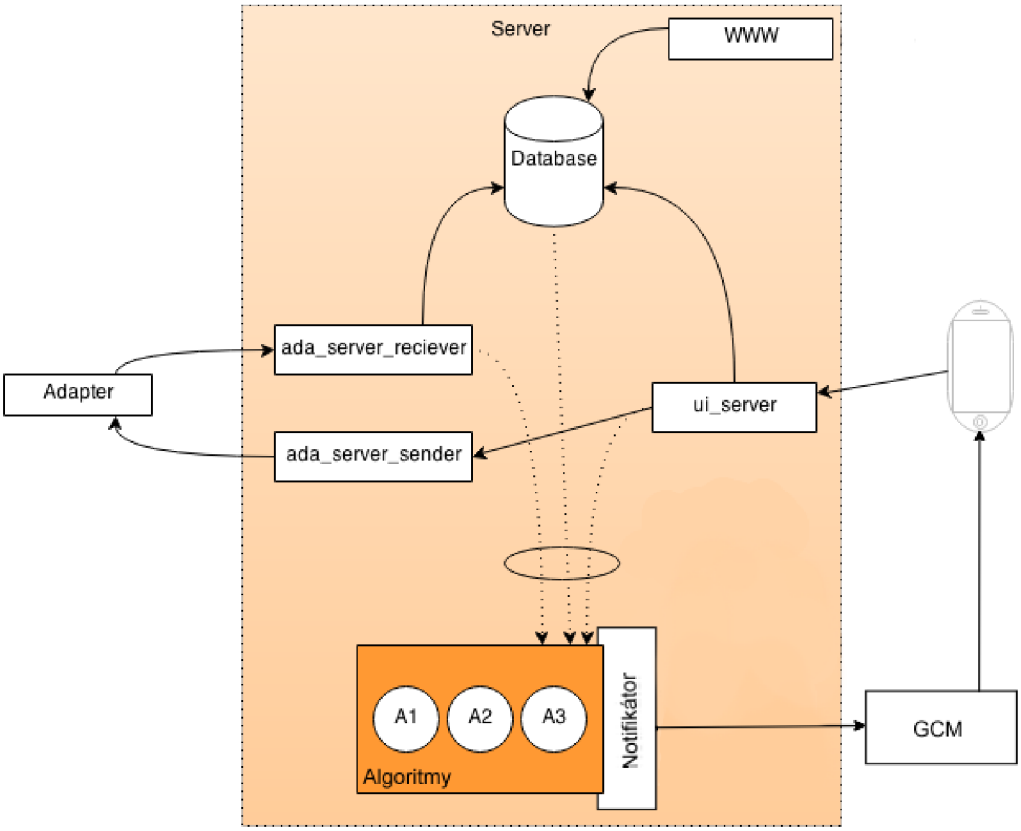
Serverová část je reprezentována vlastní knihovnou, která slouží jako rozhraní ke komunikaci s koncovým mobilním zařízením. Vývojáře serveru odstíní od implementace a vymezí přesné rozhraní pro jednotlivé zprávy.

Na obrázku 2.7 vidíme rozdělení serveru bez použití notifikačního modulu (dále jen "notifikátor"). Notifikátor je napojen přímo na komponentu algoritmů, která dostává veškeré příchozí informace od všech částí systému. Může tedy vyhodnocovat nejrůznější podmínky, které mohou být definované jak uživatelem, tak i systémově. Právě tato část serveru může jako jediná inicializovat komunikaci směrem ke koncovým zařízením. Algoritmy jsou tedy propojeny přes dynamickou knihovnu, která je s ní v době překladu zkompilována. Po připojení notifikátoru vznikne schéma 5.1.

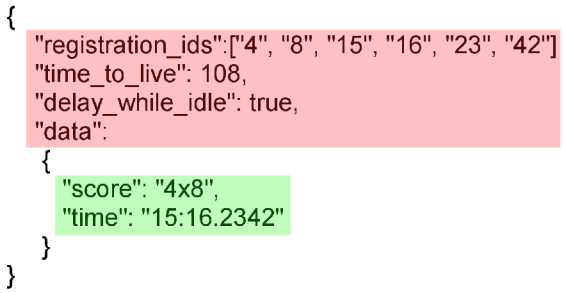
Pro komunikaci s GCM serverem je využita varianta s HTTP požadavky (viz sekce 3.1.2). Při rozhodování hrálo hlavní roli to, že aplikační server je umístěný na veřejné adrese a koncová zařízení zabezpečeně komunikují se serverem pomocí TLS. Proto implementace protokolem XMPP je zbytečná, protože není potřeba stejným kanálem odpovídat. Zejména jsou z komunikace telefon-server vyloučeny servery třetích stran, což zaručí vyšší míru bezpečnosti při přenosu tajných informací. Komunikace v rámci GCM probíhá zabezpečeně, nicméně GCM servery mají k datům přístup. Z tohoto důvodu je vhodné mít alespoň jeden nezávislý kanál, který nám zaručí bezpečný přenos i pro nejcitlivější data.

Jak již bylo zmíněno v úvodu této kapitoly, notifikátor je navrhován pro objektově orientovaný jazyk. Proto každá zpráva je reprezentována jednou třídou. Je vytvořena dědičnost tříd pro zavedení hierarchie různých notifikací a stejně tak i jednotlivé typy notifikací (viz sekce 5.1.1). Toto řešení je jednoduše rozšiřitelné a jednotlivé typy notifikací mají pevně definované parametry. Nové notifikace si pouze dodefinují nové parametry a mají zaručené vlastnosti určitého typu.

Na aplikačním serveru se uchovává i historie notifikací, která je v omezeném počtu



Obrázek 5.1: Rozdělení jednotlivých částí serveru a napojení notifikačního rozhraní – převzato z Wiki projektu IoT a upraveno.



Obrázek 5.2: Rozdělení zpracování zprávy mezi GCM serverem (červená část) a Androidem (zelená část)

zobrazena uživateli na mobilním zařízení. Jednotlivé notifikace musí obsahovat i metodu pro převod do XML formátu, který odpovídá specifikaci protokolu mezi serverem a mobilním zařízením. Iniciátor pak zodpovídá za uložení XML do databáze.

Důvod, proč je notifikace uložena do databáze ve formátu XML, je následující. Každá z notifikací má jiné parametry a v databázi by se musela vytvořit jedna tabulka, která by pokrývala všechny parametry všech notifikací. Při uložení jedné určité notifikace by pak často zůstala velká část hodnot prázdná. Proto volitelnou část notifikace uložíme jako řetězec ve formátu XML, který musí odpovídat specifikaci protokolu mezi serverem a koncovým zařízením. Jako atributy v databázi budeme uchovávat pouze povinné parametry, které jsou u všech zpráv stejné.

V systému Android jsou vytvořeny třídy odpovídající jednotlivým notifikacím. Po přijetí zprávy pomocí *broadcast recieveru*, se zpráva validuje, zda obsahuje všechny povinné parametry (viz sekce 5.1.1) a zda jsou validní. Zejména se musí zkontrolovat ID uživatele s aktuálně přihlášeným uživatelem. Pokud je vše v pořádku, vytvoří se instance třídy, která již bude mít rozhraní pro zpracování dle požadované akce.

5.1.1 Typy zpráv

Na obrázku 5.2 je znázorněno, kdo zodpovídá za zpracování jaké části zprávy. GCM server zpracovává celou část kromě sekce `data` (na obrázku červená barva). Jsou to většinou parametry z tabulky 3.1.2, které říkají GCM serveru, komu mají zprávu doručit a jakým způsobem. V obrázku zelená část je přeposlána do koncového zařízení. Pro GCM server tyto data nemají žádnou sémantiku. Do koncového zařízení tedy přichází již jen dvojice parametr a hodnota, které jsou ve zprávě přenášeny v poli *data*.

Je potřeba definovat povinné parametry a sémantiku zprávy v poli *data*, které plynou z požadavků systému. Nejedná se však o povinné parametry GCM zprávy, které jsou definované v tabulce 3.1.2. Tyto parametry budou zasílány v GCM zprávě v JSON poli *data* a zpracovává je až koncové zařízení. Všechny notifikace mají společné tyto parametry:

- **Seznam registration ID** – Tato společná vlastnost a povinnost plyne z tabulky 3.1.2. Konkrétně se jedná o parametr `registration_ids`.
- `Data` – Společné údaje v této sekci zprávy.
  - **Name** – Jméno jednoznačně určuje typ zprávy, povinné parametry a sémantiku zprávy. Každá zpráva má definovanou i reakci na přijetí a na kliknutí.

- **User ID** – Díky kontrole ID adresovaného uživatele a aktuálně přihlášeného uživatele se zaručí, že nedostane notifikaci nikdo jiný, než pro koho je notifikace určena i při nevalidním odhlášení. Za nevalidní odhlášení je například považováno vymazání dat aplikace v nastavení systému Android. Jak již bylo popsáno v sekci 3.1, *registration ID* adresuje konkrétní aplikaci na konkrétním zařízení. Nerozlišuje uživatele. Uživatel se musí přihlásit do aplikace. Jedno zařízení může využívat více uživatelů. Pokud existují dva uživatelé a první z nich se nevalidně odhlásil, zůstalo by *registration ID* uložené na serveru u prvního uživatele. Poté se přihlásí druhý uživatel. V tento moment jsou zaregistrováni (z pohledu GCM) dva různí lidé se stejným *registration ID*. Pokud aplikační server chce prvnímu uživateli poslat zprávu, obdrží ji druhý uživatel. Po detekci nesprávného doručení notifikace je o tom informován server, který si od uživatele vymaže neplatné *registration ID* a tak se uvede zpět do konzistentního stavu.
- **Časové razítko** – Server připojí aktuální časové razítko při vytváření notifikace do zprávy. Koncové mobilní zařízení totiž neví, zda obdrželo notifikaci okamžitě nebo zda byla uložena na GCM serveru několik dní. Aby bylo možné uživateli zobrazit, kdy byla zpráva reálně vytvořená, musí se čas odeslání připojit k notifikaci. Je možné také uživateli zobrazit čas přijetí notifikace, ale je to nerelevantní informace, protože zpráva mohla být na GCM serveru uložená až čtyři týdny.
- **ID notifikace** – Každá notifikace má své unikátní číslo, pomocí kterého jsme ji schopni identifikovat. ID notifikace použijeme zejména tehdy, kdy uživatel vlastní několik zařízení. Notifikace se odesílá standardně na všechna uživatelova zařízení. Pomocí ID notifikace jsme schopni poslat číslo přečtené zprávy na server a ten pomocí GCM pošle zprávy o zrušení zprávy v ostatních zařízeních.

Zprávy jsou rozděleny do následujících čtyř typů. Každá skupina má specifické vlastnosti.

- **Informační notifikace** – Jedná se o notifikace, které mají informativní ráz. Tedy nejedná se o nic důležitého. Například notifikace, kdy si uživatel nastaví, aby se mu posílala informace o překonání určité teploty.
- **Upozornění** – Jedná se o podobné notifikace jako u informačních. Tyto notifikace mají však větší váhu, mají přednost před jinými notifikacemi. Jsou to upozornění například *Hoří*, *V kuchyni se nachází voda* apod.
- **Reklamní notifikace** – Tyto notifikace nemají žádné spojení s napojenou domácností, ale budou informovat uživatele o novinkách, šířit reklamu, pobízet k užívání aplikace apod.
- **Kontrolní notifikace** – Neukládají se do historie notifikací. Jedná se o notifikace, které se budou vykonávat bez vědomí uživatele. Jsou potřeba pro správné fungování aplikace, proto je nelze zakázat. Například notifikace *Sesynchronizuj nové data* nebo *Vymaž notifikaci z notifikační lišty.*

5.1.2 Postup

Po prvním spuštění Android zařízení nemá *registration ID*. Při prvním přihlášení je zjištěno, že aplikace ještě nezná své *registration ID*. Proto požádá GCM server o přidělení nového ID. Tato operace se děje asynchronně, aby nezdržovala samotné přihlášení uživatele.

Pokud má zařízení připojení k Internetu, dostane nové *registration ID*. Toto ID se uloží do vnitřní paměti telefonu, aby se příště aplikace vyhnula komunikaci s GCM serverem. Ihned po uložení je odesíláno nové *registration ID* na aplikační server, který si uloží identifikátor do databáze. Až dojde k vyhodnocení podmínky pro zaslání notifikace určitému uživateli, vytvoří se na aplikačním serveru dotaz na databázi pro získání potřebných *registration ID*. Jeden uživatel může mít více zařízení a na všechny se posílá notifikace najednou.

Na serveru se vytvoří instance třídy dané notifikace a naplní ji požadovanými daty. Pokud se nejedná o kontrolní notifikaci, z instance si vyžádá XML řetězec a uloží do uživatelovy databáze (historie notifikací). Pomocí notifikátoru ji odešle na GCM server. GCM server se pokusí ihned doručit zprávu. Pokud je koncové zařízení aktuálně nedostupné, uloží si ji pro pozdější doručení.

Zařízení Android přijme zprávu od GCM serveru a vyvolá *broadcast intent*, který přijme pouze cílová aplikace. Zkontroluje se, zda má notifikace všechny požadované parametry, jinak se prohlásí za neplatnou a zahodí se. Pokud je notifikace adresována jinému uživateli než aktuálně přihlášenému, odesílá se oznámení na server, aby si u uživatele smazal toto *registration ID*.

Zpracuje se zbytek zprávy a vytvoří se příslušná instance třídy. Pokud je aplikace zapnutá, předá se instance notifikace do aplikace. Pokud nedojde ke zpracování, vytvoříme notifikaci v notifikační liště. Po kliknutí na notifikaci se pokračuje podle požadované reakce a odesílá se na server zpráva o přečtení zprávy.

Server rozesílá stejným způsobem zprávu na všechna uživatelova zařízení o zrušení notifikace z notifikační lišty. Zařízení, které tuto zprávu přijmou, se pokusí vymazat notifikaci z notifikační lišty.

Při dalším přihlášení do aplikace se již posílá jen zpráva s *registration ID* a s identifikací uživatele. Server si sice identifikátor nechává uložený, ale tímto se zajistí větší spolehlivost při nekorektním chování systému.

V sekci 3.1.3 je popsáno, v jakých případech může dojít ke změně *registration ID*. Je to při aktualizaci celého Android systému nebo aplikace. Oba tyto případy je nezbytné co nejdříve detekovat a zažádat GCM server o přidělení nového *registration ID*. Pokud GCM server přidělí jiný identifikátor, než je aktuálně uložený, musíme starý přepsat v paměti a poslat nový na server. Může se stát, že bude zařízení zrovna mimo připojení na Internet. V tomto případě se naplánuje tato akce po připojení k Internetu. V krajním případě se vše vykoná při dalším přihlášení.

Při odhlášení uživatele z aplikace je potřeba smazat identifikátor na serveru. Pokud by se identifikátor nesmazal, server by stále posílal notifikace na zařízení.

5.2 Geolokační oblasti

Geolokační oblast představuje uživatelem definovanou zónu, kterou může následně využít pro definici podmínky. Konkrétně v sytému BeeeOn slouží jako jeden z možných vstupu podmínky hlídače, který na základě splněné podmínky vykoná požadovanou akci.

Pro uživatelsky přívětivé definování oblastí je vhodné využít mapových podkladů. Je tedy vytvořena obrazovka s mapou, kde uživatel může interaktivně přidávat oblasti. Jedna oblast je definována středem a poloměrem kruhu. Střed je definován GPS souřadnicemi. Každá oblast má navíc svůj identifikátor a název, aby ji uživatel jednoduše rozpoznal.

Pro vytvoření oblasti se využívá *Location API*. Při registraci nové oblasti se nastaví naslouchání na vstup i výstup ze zóny. Obě události při překročení zóny jsou posílány na server.

V mapě uživatel vytvoří novou oblast dlouhým kliknutím na místo, kde je střed pomyslného kruhu oblasti. V dialogu dojde k pojmenování a definování poloměru kruhu oblasti (rádius). Samotný dialog kontroluje validnost vstupních hodnot a nedovolí uživateli vytvořit neplatnou oblast. Pro identifikátor musí být zajištěno, že je pro jednoho uživatele unikátní. To znamená, že i když uživatel vlastní několik zařízení a vytváří oblasti na všech, musí být zajištěno, že se v žádném případě nestane, že budou mít libovolné dvě oblasti stejný identifikátor.

Zároveň je potřeba perzistentně uchovat všechny další informace o definovaných oblastech, a proto interní databáze systému Android uchovává tyto data. V databázi jsou uchovány veškeré informace o dané oblasti. Aplikace může obsluhovat několik uživatelů, proto se musí odregistrovat oblasti při odhlášení a opět zaregistrovat při dalším přihlášení. Nesmí zůstat zaregistrované oblasti uživatele, který již není přihlášen v aplikaci. Zbytečně by se spotřebovávaly zdroje telefonu pro lokalizaci, která nikdy nebude využita.

Definované oblasti pak systém využívá pro definování pravidel hlídače, kde se na základě polohy mohou automatizovat činnosti. Pro tvorbu pravidla využívá právě identifikátor oblasti.

Jelikož ke správnému fungování musí být přístupné *Google Play Services*, je potřeba zkontrolovat jejich přítomnost. Aplikace vyvíjené pro platformu Android je možné spustit například na platformě Blackberry. Může se také jednat o alternativní Android platformu *CyanogenMod*[1]. V těchto případech nejsou *Goole Play Services* dostupné a je nutné zakázat přístup uživateli na obrazovku pro definování geolokačních oblastí.

Zvláštním případem chování je demo mód, který umožňuje uživateli vyzkoušet si aplikaci bez nutnosti přihlášení do systému a bez nutnosti vlastnit nějaký reálný adaptér. V demo módu jsou poskytnuta jen zkušební data, aby si uživatel vyzkoušel práci s aplikací. Pokud se uživatel nachází v demo režimu, vytvoření oblasti je pouze simulováno vytvořením záznamu v databázi, aby byla oblast viditelná v mapě. K samotné registraci sledování nedojde, tudíž vyvolání akce vstupu nebo výstupu z oblasti nikdy reálně nenastane. Uložená data jsou z databáze při odhlášení z demo režimu vymazána, aby při dalším spuštění byla demo data stejná.

[1] http://www.cyanogenmod.org/

Kapitola 6

Implementace

6.1 Google Cloud Messaging

Pro asynchronní komunikaci ze serveru na mobil je využita služba Google Cloud Messaging. Musí se provést prvotní konfigurace na Google Developers Console. Po založení projektu musí být aktivována služba a vytvořen veřejný přístup pro serverovou část aplikace. Pro vyšší míru zabezpečení je vhodné omezit přístup pouze na konkrétní IP adresu serveru, ze které budou požadavky na GCM server přicházet. Vygenerovaný API klíč a ID projektu se využívá při komunikaci směřující z aplikačního serveru.

Všechny části aplikačního serveru jsou vyvíjeny v jazyce C++. Proto i knihovna pro komunikaci s GCM serverem je vytvořena v tomto jazyce. Na obrázku 6.1 je znázorněn třídní diagram knihovny. Celá knihovna je navržena a implementována tak, aby umožnila jednoduché přidávání dalších zpráv pomocí dědičnosti a poskytla možnost rozšíření na další platformy bez úprav stávající knihovny.

Rozhraní `Notification` jednoznačně deklaruje všechny metody, které jsou z pohledu uživatele knihovny přístupné. Zejména se jedná o metodu `sendGcm()`, která jako parametr přijímá seznam *registration ID*, na které se zpráva doručí. Po vytvoření JSON požadavku se o samotné poslání zprávy stará statická metoda `Notificator::sendGcm()`, která požadavek zabalí do HTTP POST zprávy a odešle na GCM server. K vytvoření JSON formátu zprávy se využívá třída `JsonNotificationBuilder`, která reprezentuje návrhový vzor *stavitel* (anglicky *builder*). K samotnému dotazu na GCM server využívá metoda pomocnou knihovnu *cURL*[1]. Požadavky se posílají na `https://android.googleapis.com/gcm/send`, tudíž se jedná o zabezpečenou komunikaci. V příkladu zprávy `Watchdog` níže vidíme všechny náležitosti zprávy poslané na server.

```
POST /gcm/send HTTP/1.1
Host: android.googleapis.com
Content-Type: application/json
Authorization: key=AIzaSyDKkdcdTwSkDT2P-AFp21BN0ZsadEUkWX0
project_id: 72175564578
Content-Length: 384
{
  "registration_ids":["APA91bH_8E7ph0hY1cPoSGcznmYzF0pn_cahzm2OrUXPk42Cdr
    7KS_jxSNQcHLdhxfXEJeS2d_3J_KcHRxDJfcYONRs1jCNVpr4c2GOQP13PFflWIvK7iHA
    OH_-qD3L7mdz2vwPCyo3q_8_gF0m8KkeJpmHjmo9ETA"],
```

[1] http://curl.haxx.se/

```
  "data": {
    "name":"watchdog",
    "uid":"8",
    "time":"1431185259367",
    "type":"info",
    "mid":"123",
    "msg":"Vlhkost v˜obýváku překročila 60%",
    "aid":"245",
    "did":"caf6",
    "dtype":"100",
    "algid":"3",
    }
}
```

GCM server odpovídá na GCM požadavky některou zprávou definovanou v sekci 3.1.2. Po zpracování odpovědi navrací metoda seznam *registration ID*, které jsou určeny k vymazání z databáze nebo prázdný seznam.

Po odeslání zprávy je třeba notifikaci uložit do databáze, aby byla uchována historie pro koncového uživatele. Ne však každou zprávu chceme archivovat. Jedná se zejména o kontrolní zprávy, které jsou pro uživatele transparentní a nemá smysl je tedy ukládat pro pozdější zobrazení v historii notifikaci. Rozhraní `Notification` proto deklaruje metodu `saveToDb()`, která navrací `boolean` hodnotu, která určuje, zda se má zpráva uložit do databáze či nikoliv. Jelikož má každá zpráva jiný počet parametrů, jsou jako jednotlivé atributy tabulky uloženy pouze povinné parametry zpráv. Specifické hodnoty zpráv jsou do tabulky databáze vloženy jako XML řetězec. K vygenerování řetězce slouží metoda `getDbXml()`, která za pomocí statických metod ve třídě `XmlHelper` generuje pro každý nepovinný parametr zprávy XML řetězec ve tvaru:

```
<nazev_parametru>hodnota</nazev_parametru>
```

Každá zpráva je jednoznačně definována konstruktorem třídy a je nezávislá na cílové platformě (Android, iOS, Windows Phone). Jednotlivé cílové platformy se rozlišují až na základě použité metody pro odesílání požadavku. Jak je možné vidět v třídním diagramu 6.1, předkem všech zpráv je abstraktní třída `BaseNotification`, která si uchovává většinu povinných atributů. Od této třídy dědí abstraktní třídy `AlertNotification`, `InfoNotification`, `AdvertNotification` a `ControlNotification`, které jednoznačně zařadí koncovou zprávu do jednoho z typů. Každá zpráva zařazená do jedné ze skupin má tyto povinné parametry:

- `string name` - Jméno identifikuje, o jakou zprávu se jedná. Se jménem zprávy je definovaná i sémantika a dodatečné povinné parametry specifické pro konkrétní zprávu. Při zpracování na straně příjemce podle jména poznáme, jaké specifické parametry musí zpráva obsahovat.

- `string type` - Každá zpráva je zařazena do jednoho z typů (upozornění, informace, reklama, řídící). Každý typ má specifické vlastnosti a při zpracování zprávy na straně příjemce může být jinak obsluhována.

- `int userId` - Unikátní ID uživatele je zasláno kvůli kontrole přihlášené osoby na koncovém zařízení, aby se nestalo, že zpráva bude doručena jiné osobě než adresované.

- `int notificationId` - Jednoznačně identifikuje jednu konkrétní zaslanou zprávu. Identifikátor jedné zprávy zaslané na více platforem je pro všechny platformy stejný. Je možné tedy i po odeslání zprávy ze serveru aktualizovat již odeslanou zprávu nebo s ní jinak manipulovat.

- `long timestamp` - Časové razítko je čas vytvoření zprávy na serveru. Jelikož zpráva může být při doručení pozdržena i několik dní (nedostupný mobil apod.), musí uživatel vědět, kdy byla zpráva vytvořena. Razítko je v časové zóně UTC a reprezentován v UNIX formátu (počet milisekund od 1. ledna 1970).

V příloze A je seznam implementovaných zpráv, jejich specifické parametry a definované chování při přijetí zprávy nebo při interakci uživatele.

Zprávy, které patří do upozornění, informacím nebo reklamě, jsou hromadně označovány jako viditelné notifikace. Uživatel je vždy při přijetí této zprávy notifikován. Aby notifikace mohla být zobrazena, je potřeba mít k dispozici lokalizovaný text. Máme tedy dvě možnosti. První a preferovaná varianta je uložení textu a jeho lokalizace v cílovém zařízení, přičemž každá zpráva má definovaný řetězec k zobrazení podle jména zprávy. Výhody tohoto přístupu jsou zejména ve zmenšení objemu dat přenášených pomocí GCM a jednoduchá lokalizace cizích jazyků podporovaná nativně mobilními systémy. Můžeme také přizpůsobit text až v době zobrazování notifikace. Tento princip je využit u většiny notifikací.

V některých případech však definované řetězce v koncovém zařízení nestačí. Jedná se o situace, kdy se mění často text notifikace nebo si může text nadefinovat sám uživatel. Využijeme tedy druhý způsob. Text je zasílán spolu s ostatními parametry zprávy. Zde je potřeba myslet na to, aby text nepřekonal maximální limit zprávy. GCM server by pak takové zprávy odmítal a nebyly by doručeny. Výhodou tohoto přístupu je možnost změny řetězce bez potřeby aktualizace aplikace.

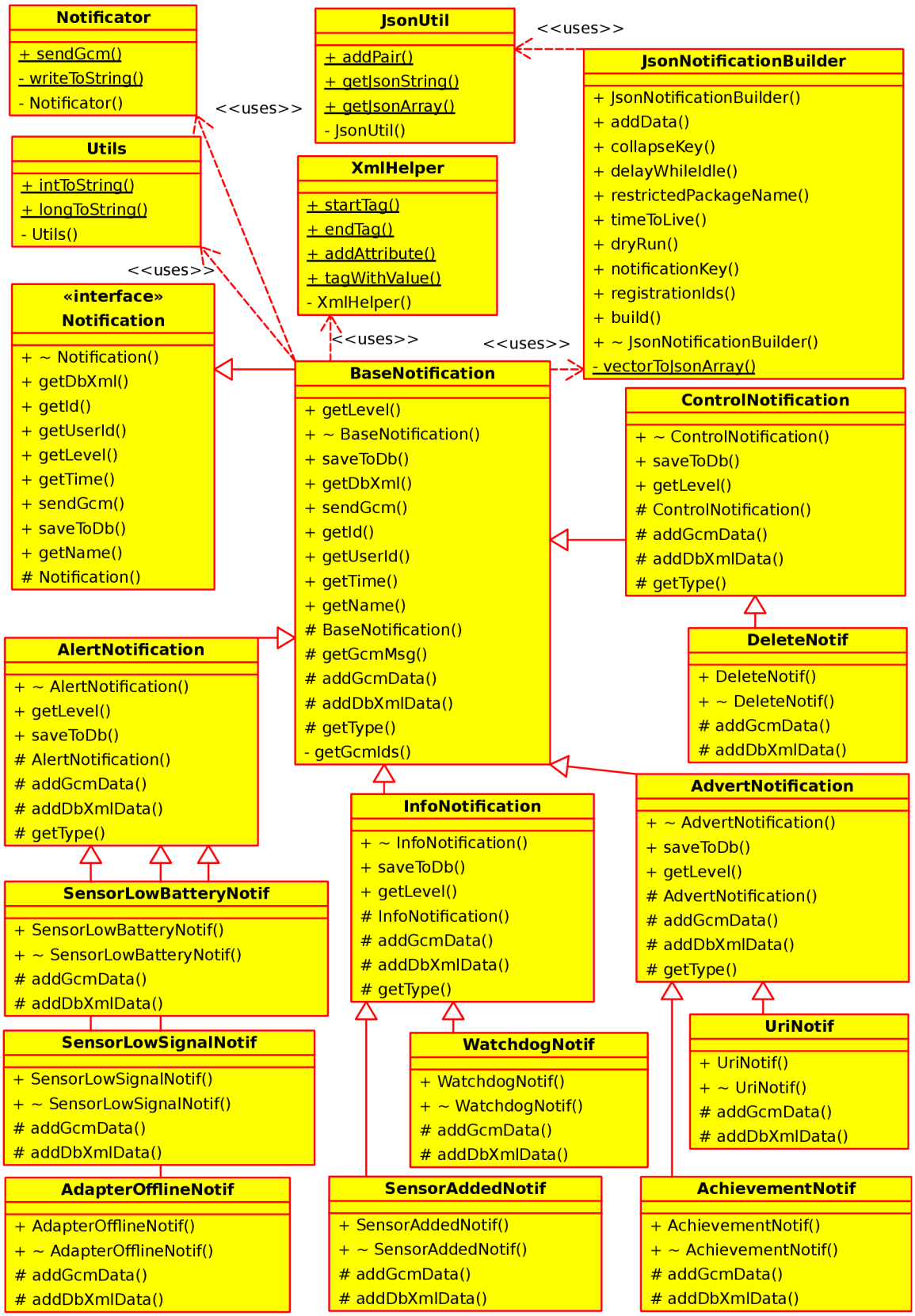
Po kompilaci knihovny se vytvoří dynamická i statická knihovna. Základní použití je prezentováno na příkladu v příloze B.

Serverová část tedy umí odeslat notifikaci. Aby Android aplikace byla schopna přijmout zprávu, musí mít několik povolení. Samozřejmě musí mít aplikace přístup k Internetu. Musí mít speciální povolení na přijmutí GCM zprávy. Abychom mohli zabránit procesoru přejití do režimu spánku, potřebujeme si vyžádat speciální povolení. Pro zabezpečení brodcastového vysílání od odposlechu si vytvoříme vlastní povolení, které musí začínat jménem balíčku a končit `.permission.C2D_MESSAGE`. Podle výše uvedeného popisu a ve stejném pořadí jsou potřebná povolení uvedená v `AndroidManifest.xml` k přijetí GCM zprávy názorně ukázané v příkladu:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.c2dm.permission
  .RECEIVE" />
<uses-permission android:name="android.permission.WAKE_LOCK />

<permission android:name="cz.vutbr.fit.beeon.permission.C2D_MESSAGE"
  android:protectionLevel="signature" />
<uses-permission android:name="cz.vutbr.fit.beeon.gcm.permission
  .C2D_MESSAGE" />
```

Všechna potřebná povolení jsou nastavená. Aby mohla být zpráva vůbec z aplikačního serveru odeslána, musí server znát cílové *registration ID*. Před žádostí GCM serveru o při-



Obrázek 6.1: Diagram tříd serverové části

dělení identifikátoru se provede kontrola, zda jsou v zařízení *Google Play Services*. Pokud jsou zastaralé nebo vůbec nejsou, upozorní se uživatel. Pokud je vše v pořádku, provede se žádost o registraci do služby. Tuto akci zajišťuje třída `GcmRegisterRunnable`, která může být spuštěna v hlavním nebo i vlastním vlákně. Prvotní registrace do služby se děje při prvním přihlášení uživatele do aplikace. Tento proces se odehrává na pozadí ve svém vlákně, aby neovlivnil rychlost přihlášení. Po získání *registration ID* se uloží do perzistentního úložiště aplikace spolu s aktuální verzí aplikace. Zároveň se odešle zpráva o přidělení nového *registration ID* přihlášenému uživateli.

Aby byla zajištěna spolehlivost doručování, musí na aplikačním serveru vždy být aktuální *registration ID*. To se může za určitých okolností měnit. Poměrně častou situací je aktualizace aplikace. V tomto případě nesmí být využita uložená hodnota z aplikace, ale musí se provést nový dotaz na GCM server. Není zaručené, že aplikace dostane přidělený nový identifikátor. Porovnáme ji tedy s již uloženou hodnotou a pokud je stejná, neodesíláme ji na server, ale pouze aktualizujeme verzi aplikace. Tím uděláme lokálně uložený identifikátor opět platným. Android umožňuje registrovat *reciever*, který je vyvolán při aktualizaci aplikace. Tuto broadcastovou zprávu zpracovává třída `UpdateBroadcastReceiver`. Docílí se tím okamžitého ověření identifikátoru a případné aktualizace hodnoty na serveru ihned po aktualizaci aplikace (i když nebyla ještě od aktualizace spuštěna).

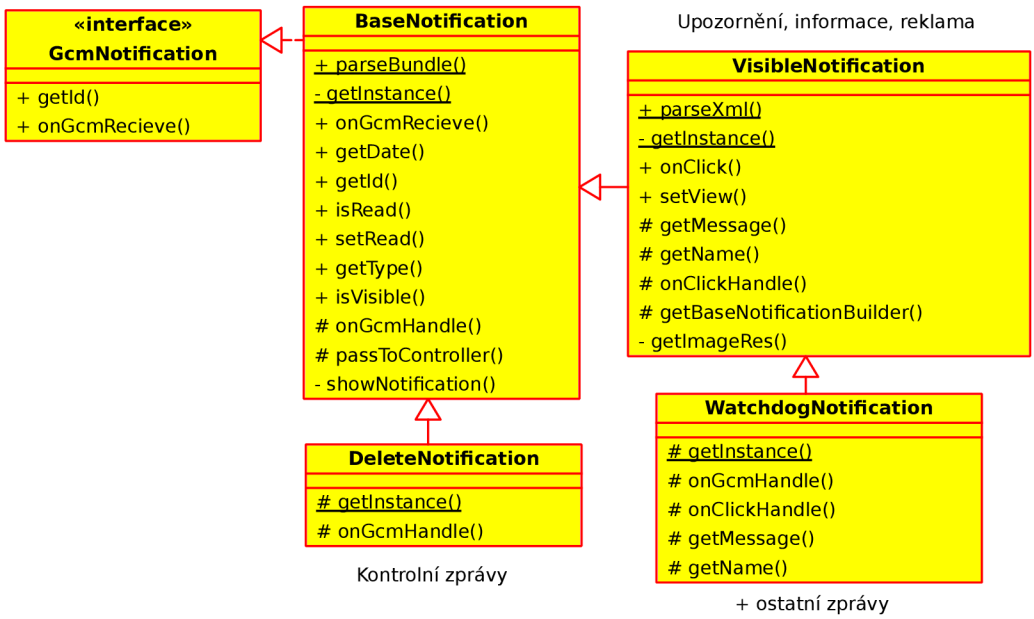
Méně obvyklým případem je aktualizace celého systému. Android v této situaci nevysílá žádnou zprávu o této skutečnosti. Jediná možnost, jak zajistit spolehlivost i v tomto případě, je ověření identifikátoru po každém restartování zařízení, jelikož každá aktualizace systému je spojená s restartem. Postupuje se stejně jako u aktualizace aplikace.

V obou výše uvedených případech může nastat, že při pokusu o opětovnou registraci není dostupný Internet. V takovém případě se provede ověření nejpozději při návratu do aplikace.

Aplikační server již dokáže adresovat aplikaci, může se tedy přejít na proces přijímání GCM zprávy. Systém Android oznamuje přijetí zprávy pomocí broadcastových zpráv. K tomu slouží třída `GcmBroadcastReciever`. Po přijetí GCM zprávy je probuzen tento *reciever*, který předá data třídě `GcmMessageHandler`. Tato třída využívá dědičnosti od třídy `IntentService`, takže každá zpráva se zpracovává asynchronně ve svém vlákně. Dvojice proměnná – hodnota jsou předány statickým metodám pro zpracování a ověření platnosti.

Každá zpráva, kterou aplikace umí zpracovat, je reprezentovaná samotnou třídou. Na diagramu tříd 6.2 se nacházejí dvě abstraktní třídy. Koncové třídy jsou kvůli přehlednosti v diagramu vynechány a naznačeny pouze dvěma příklady (`DeleteNotification`, `WatchdogNotification`). Nejobecnější abstraktní třída je `BaseNotification`, která je rodičem všech tříd. Uchovává všechny povinné atributy zprávy. Třídní statickou metodou `parseBundle()` se zpracuje přijatá zpráva, ověří se validnost a případně se vytvoří instance již konkrétní zprávy. Pokud by došlo k chybě při ověřování ID uživatele, zpráva je zahozena a o této skutečnosti je informován i aplikační server. Ten si musí odstranit u adresovaného uživatele aktuální *registration ID*, aby nedocházelo k opakovaným doručováním jinému uživateli. Pokud je zpráva ověřena a instance vytvořena, následuje volání metody `onGcmRecieve()`. Tato metoda nejdříve předá zprávu kontroléru aplikace, kde si mohou obrazovky registrovat *listener* pro příchozí notifikace – návrhový vzor *pozorovatel (anglicky observer)*. Pokud je notifikace kýmkoliv zpracována, považuje se za vyřízenou. Jinak se volá vnitřní funkce koncové třídy pro obsluhu notifikace.

Specifičtější abstraktní třída je `VisibleNotification`, která reprezentuje všechny viditelné notifikace. Obsahuje tedy navíc metody pro získání textu ke zobrazení. Zároveň na volání metody `onGcmRecieve()` vždy reaguje zobrazením notifikace v notifikační liště. Jeli-



Obrázek 6.2: Diagram tříd notifikací na platformě Android

kož aplikační server udržuje informaci o přečtení konkrétní zprávy, musíme zajistit zaslání příznaku přečtení na server. Systém Android neumožňuje definovat *callback* pro kliknutí na notifikaci v liště. Proto kliknutí na libovolnou zprávu znamená přesměrování na obrazovku historie notifikací. V *intentu* předáme stejný `Bundle`, který jsme získali v GCM zprávě a tím je zajištěno, že se zpracuje totožná zprávu. Vytvoří se také předchůdce obrazovky notifikací, jinak by uživatel při kliknutí *zpět* opustil aplikaci. Odkáže se tedy na hlavní obrazovku `MainActivity`. V `NotificationActivity` v metodě `onCreate()` se zkontroluje, zda byl nějaký `Bundle` předán. Pokud ano, zpracuje se instance zprávy, odešle se příznak o přečtení a využije se metody `onClick()`, která obslouží kliknutí.

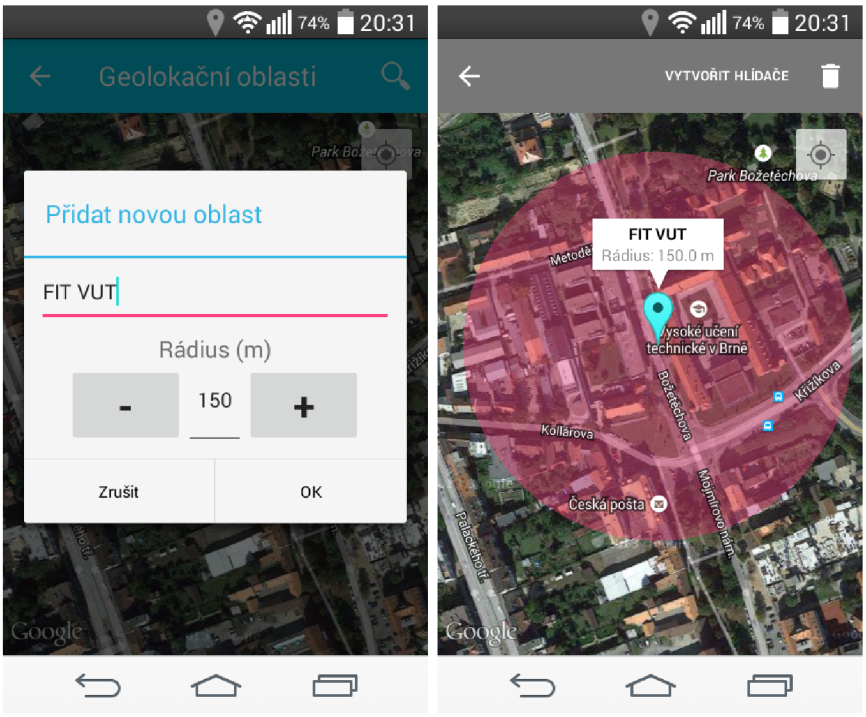
V třídě `VisibleNotification` se nachází i metoda `parseXml()`, která zpracovává seznam zaslaných notifikací z aplikačního serveru. V tomto případě se nesmí volat metoda `onGcmRecieve()`, ale zpráva reaguje pouze na kliknutí.

6.2 Geolokační oblasti

Vstupem pro vytváření nových oblastí je obrazovka `MapGeofenceActivity` (viz obrázek 6.3), která obsahuje Google mapu. Aby bylo možné Google mapy používat, musí být pro aplikaci správně nastavena služba v *Google Developers Console*. Projekt je již vytvořený a doplní se pouze služba *Google Maps Android API v2*. Poté je vytvořen veřejný API klíč pro Android aplikaci a zadány otisky SHA-1[2] certifikátu aplikace. Zde je potřeba si uvědomit, že i vývojová verze aplikace musí být podepsána klíčem. Používá se vývojový klíč společný pro všechny projekty ve vývojovém prostředí. Je třeba uvést vývojový i produkční otisk certifikátu. Výsledný API klíč je pak vložen do souboru `AndroidManifest.xml`.

Mapové podklady se stahují z Internetu, proto musí mít aplikace povolení a přístup na Internet. Jelikož je požadováno uživatele lokalizovat, musíme přidat i povolení k přístupu

[2] *Secure Hash Algorithm*



(a) Přidání nové oblasti (b) Zobrazení v mapě

Obrázek 6.3: Obrazovka pro geolokační oblasti

o aktuální poloze. Google mapy zároveň vyžadují přístup k externímu úložišti z důvodu cachování mapových podkladů. Aplikaci se tedy přidělí následující povolení:

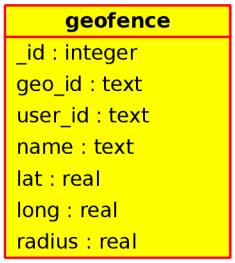
```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Aby uživatel mohl vyhledat určitou adresu, v hlavní liště se po stisknutí ikony vyhledávání otevře tzv. *collapse action view*, kde uživatel může vepsat hledanou adresu. Po vložení a potvrzení se vytváří nové vlákno, ve kterém se provádí dotaz na Google. Pokud rozpozná vstup od uživatele, navrátí GPS souřadnice a přesune mapu na výsledné místo. Pokud vstup od uživatele nerozpozná, je o této skutečnosti informován.

Pro lokalizaci uživatele a definování oblastí je využito služby *Location Services*. Při zobrazení geofence obrazovky se aplikace připojí ke *Google API Client* a zůstane připojená po celou dobu existence obrazovky. Pro vytvoření oblasti se využije statické metody `addGeofences()` třídy `LocationServices.GeofencingApi`. Tato metoda asynchronně vyřídí požadavek a pomocí *callback* rozhraní informuje o výsledku. Pokud bylo přidání úspěšné, je vykreslena oblast do mapy a uložena do databáze. Podobně probíhá i smazání oblasti.

Při vytváření oblasti je vygenerován unikátní identifikátor. Ten je tvořen konkatenací identifikátoru telefonu a aktuálního času v milisekundách. Tímto je zaručeno, že identifikátor je v rámci jednoho uživatele vždy unikátní.

O zpracování přechodů hranic oblasti se stará *service* `GeofenceIntentService`. Zkontroluje, zda aktuální oblast patří k aktuálně přihlášenému uživateli, určí, jestli došlo ke



Obrázek 6.4: Tabulka pro uložení oblasti

vstupu nebo výstupu ze zóny a odešle zprávu na aplikační server. Ten pak vyhodnotí, jaká pravidla jsou s touto oblastí a směrem (vstup nebo výstup) spojena a vykoná je.

Po vstupu na obrazovku pro zobrazení a zadávání oblastí se asynchronně načte mapa a poté mohou nastat tři scénáře podle počtu zaregistrovaných oblastí:

1. **Žádná oblast** – animace na poslední známou pozici uživatele
2. **Jedna oblast** – přiblížení konkrétní oblasti
3. **Více než jedna oblast** – vypočítá se oddálení mapy, aby byly viditelné všechny oblasti a přesune se na ně

Pro perzistentní uložení oblastí je využita *SQLite*[3] databáze. Každá oblast je uložena do tabulky 6.4. Práci s databází zapouzdřuje třída `GeofenceModel`, která mimo jiné převádí databázový záznam do třídy `SimpleGeofence`.

Při odhlášení uživatele z aplikace musí být všechny zaregistrované oblasti odstraněny z aktivního naslouchání. Pokud by nebyly odregistrovány, docházelo by stále k vyhodnocování oblastí, a tudíž ke zbytečnému využívání prostředků systému. Definované oblasti však zůstávají uložené v databázi a při dalším přihlášení uživatele musí být znovu zaregistrovány.

[3] https://www.sqlite.org/

Kapitola 7

Výsledky

Implementované řešení bylo pravidelně publikováno na *Google Play*. Bylo potřeba vytvořit projekt tak, aby nebyl veřejně dostupný. Testování tedy bylo rozděleno na *alpha* a *beta*. Přístup k těmto verzím se řídí Google+ komunitami, kdy uživatel musí být členem alespoň jedné z nich. Podle členství v komunitě je při stahování přidělena příslušná verze. Platí, že pokud je uživatel *alpha* tester, může dostat i *beta* verzi, pokud je novější než aktuální *alpha* verze. Každý měsíc byly organizovány demonstrační dny, kdy se v rámci systému předváděli nové funkce. Pro tuto událost byly vydávány stabilní *beta* verze, které se udržovaly po celý měsíc (pokud nebylo potřeba upravit kritické chyby). Mezitím se vydávali pravidelně *alpha* verze, které zejména sloužily vývojářům ostatních týmů pro vývoj. Celkově bylo zveřejněno na *Google Play* 49 verzí aplikace.

Výhodou šíření přes *Google Play* je možnost jednoduchého reportování pádu aplikace. Tyto reporty obsahují i podrobné trasování zásobníku, tudíž vývojář ví, co pád způsobilo. Také má k dispozici přesnou posloupnost volání metod vedoucích k pádu. Vývojář pak vidí i časové razítko, počet výskytů, verzi aplikace a systému a v neposlední řadě připojený komentář k pádu. Přístup k testování měli členové týmu IoT a počet aktivních instalací je přibližně dvacet.

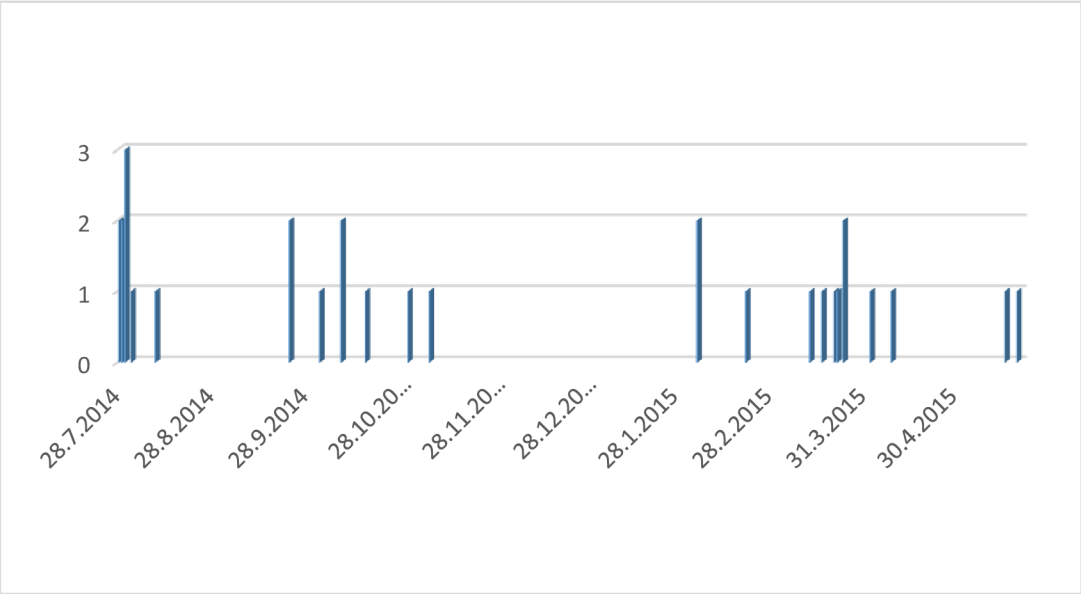
V reálném nasazení aplikace se předpokládá, že ne všichni uživatelé pády reportují. Proto zde byl zaveden ještě jeden systém. Aplikace je napojená na službu *Google Analytics*[1]. Mimo pádů je možné pomocí této služby sledovat i chování uživatele a základní údaje o něm. Lze vytvořit i svoje proměnné a události, které chceme sledovat. Oproti *Google Play* zde pády neobsahují trasovací zásobník, proto je většinou těžké přijít na chybu, která pád způsobila. Spíše se jedná o statistiku pádů a jejich názvů.

Ne vždy nesprávné chování aplikace končí pádem. Uživatel může například postrádat v mapě vloženou oblast. V takovém případě je možné využít reportovací systém *Redmine*[2], kde může uživatel vložit podmět k řešení problému a blíže ho popsat. V tomto systému bylo zaznamenáno celkově třicet nahlášených chyb nebo návrhů na vylepšení. Časový průběh hlášení je vyobrazen v grafu 7.1.

7.1 Experimenty

Pro ověření funkčnosti implementovaného řešení bylo provedeno několik experimentů. Všechny experimenty jsou prováděné na mobilním telefonu LG F60 (D930n) s verzí Androidu 4.4.4

[1] http://www.google.com/analytics/
[2] http://www.redmine.org/



Obrázek 7.1: Reportované úkoly v Redmine v závislosti na čase

	1. pokus	**2. pokus**	**3. pokus**	**4. pokus**	**5. pokus**
1. zpráva [ms]	3134	1963	684	661	1592
10. zpráva [ms]	3199	2265	958	876	1677

Tabulka 7.1: Zpoždění doručení při bufferování zpráv na GCM serveru

(API level 19), 1024 MB RAM a podporou LTE připojení.

Při testování notifikací je hodnoceno zpoždění mezi odesláním notifikace ze serveru a přijetím na mobilním telefonu. Jako testovací notifikace byla využita zpráva `Watchdog` o velikosti 384 B. Pro připojení k Internetu bylo zvoleno WiFi. Do zpoždění je započítán i čas pro zpracování zprávy.

Experiment se zaměřuje na rychlost doručení zprávy, pokud je zpráva již uložená na serveru GCM. Tento případ simuluje situaci, kdy je mobilní telefon mimo dosah sítě Internetu, a proto nemůže být zaslaná zpráva ihned doručena. WiFi i mobilní síť je vypnutá. Zašle se deset zpráv na GCM server. Zapne se WiFi a změří se, jak dlouho trvalo, než došlo k doručení zpráv.

Android nabízí možnost registrovat *receiver*, který informuje o odpojení nebo připojení k Internetu. Aplikace se zaregistruje, rozpozná připojení k Internetu a reaguje na něho uložením aktuálního časového razítka. Při přijetí zprávy se pak vypočítá rozdíl aktuálního času a uloženého časového razítka. Výsledné hodnoty jsou v tabulce 7.1. První řádek udává čas přijetí první zprávy od doby, kdy aplikace obdržela informaci o připojení k Internetu. Druhý řádek pak reprezentuje čas přijetí poslední (desáté) zprávy v pořadí od připojení do sítě. Jednotlivé sloupce

Je nutné poznamenat, že *receiver* reaguje až v okamžiku, kdy je ustaveno spojení a připojení k Internetu je ověřeno. Proto doba od kliknutí na aktivující tlačítko WiFi a obdržení *receiveru* se pohybuje od deseti do patnácti sekund.

Zátěžovou zkouškou notifikací bylo zacyklení posílání notifikací na server. Vytvořila se smyčka, ve které se vytvářel objekt zprávy s vždy o jedno větším identifikátorem notifikace než zpráva předchozí. Na straně Androidu jsme sledovali, zda byly doručeny všechny zprávy. Ze serveru bylo odesláno tisíc notifikací za sebou a Android zařízení i tisíc notifikací přijal. Jednotlivé zprávy byly přijímány s rozestupem cca desetinu sekundy.

Podobný pokus se zopakoval ještě jednou, ale nejprve jsme nechali uložit zprávy na GCM server. Připojení k Internetu bylo vypnuté, odeslalo se tisíc hodnot a po odeslání poslední notifikace aktivovali připojení do sítě Internetu na mobilním zařízení. Žádná zpráva nebyla doručena. Zmenšoval se tedy počet zaslaných zpráv po stovkách. Až desetina původního počtu zpráv byla zpracována. Při bufferování na GCM serveru bylo doručeno maximálně sto notifikací.

Pro otestování geolokačních oblastí byly zvoleny menší oblasti, kdy cílem bylo označit například svůj dům nebo práci. Samotný Google ve své dokumentaci uvádí doporučenou velikost poloměru minimálně sto metrů. Při tomto poloměru Android dokáže využít lépe lokalizaci pomocí mobilní sítě a WiFi a spotřebovává tak méně baterie. Bylo sledováno rozdílné chování při různých nastaveních telefonu.

Pro účely testování byla aplikace přizpůsobena tak, aby při překročení oblasti ihned vytvořila notifikaci do notifikační lišty a zejména vibrací a zvukem upozornila. Standardně se tato událost vykonává pouze na pozadí bez notifikace uživatele. Tuto úpravu bylo také nezbytné udělat z důvodu vyloučení latence síťové komunikace a serveru.

První část testování probíhala s povolenou vysokou přesností určování polohy. Testován byl vstup do oblastí v různých lokalitách. I když se mobil nacházel v kapse a obrazovka byla zamknutá, všechny reakční doby po vkročení do oblasti byly do patnácti sekund.

V druhé části bylo zakázáno v nastavení systému používat pro určování polohy GPS. Telefon pak polohu určuje jen na základě mobilní sítě a WiFi signálu. Po zakázání GPS se prudce zvýšila nepřesnost. Místy to bylo i přes sto metrů. V takových případech vytvořená oblast nezareagovala na vstup do oblasti, jelikož nepřesnost byla tak velká, že nebylo možné rozhodnout, jestli se nachází zařízení uvnitř oblasti či nikoliv. Velký vliv na přesnost při vypnuté GPS bylo místo, kde se zařízení nacházelo. Pokud se zařízení nacházelo uprostřed parku, byla nepřesnost velká. Naopak pokud se nacházelo ve městě s hustou zástavbou, přesnost byla do padesáti metrů. Velký vliv v tomto případě mají WiFi sítě. Ve městech je velká pravděpodobnost, že jednotlivé vysílače jsou již lokalizované. Společnost Google anonymně shromažďuje data o WiFi sítích. Pokud se v blízkosti sítě objeví zařízení s povoleným zasíláním informací a lze zjistit poloha pomocí GPS, společnost Google si uloží tyto data. Na základě těchto dat je poté schopen lokalizovat jiné zařízení pouze na základě WiFi sítě s dostatečnou přesností. Pokud tedy uživatel využívá geolokační oblasti s poloměrem větší než sto metrů, je tento způsob lokalizace dostatečný za předpokladu, že se nachází v oblasti, kde jsou lokalizované WiFi sítě.

Kapitola 8

Závěr

Cílem diplomové práce bylo nastudovat možnosti asynchronní komunikace telefonu a serveru. Na základě studie navrhnout způsob pro jejich vzájemnou notifikaci při událostech generovaných na serveru či telefonu s využitím v chytré domácnosti. Požadavkem byla zabezpečená komunikace a nízká spotřeba telefonu. Dalším úkolem bylo navržené řešení implementovat a otestovat. Všechny výše uvedené cíle práce byly splněny.

První část práce obsahuje popis chytré domácnosti. Zmíněna je i historie a předpokládaný vývoj v oblasti chytrého bydlení. Následuje analýza pěti existujících řešení, které již využívají k ovládání mobilní telefony. Dále se zde nachází popis klíčového projektu BeeeOn, který je vyvíjen v rámci Fakulty informačních technologií na VUT v Brně. Podrobná analýza a pochopení systému BeeeOn byla důležitou částí pro následný návrh systému. Na základě toho byly definovány požadavky.

Dalším krokem bylo seznámit se s možnostmi asynchronní komunikace mezi serverem a mobilním telefonem. V této části je podrobněji popsán způsob push notifikací a jejich nativní podpora pro tři nejrozšířenější platformy (Android, iOS, Windows Phone). Bylo nezbytné analyzovat alespoň tyto tři platformy, aby byl návrh serverové části dostatečně obecný.

Následně práce byla zaměřena na platformu Android. Pro účely notifikace serveru byla zvolena geolokace uživatele. Byly podrobněji rozebrány možnosti geolokace uživatele pomocí chytrého zařízení se systémem Android. Uživatel si může definovat oblasti v mapě a na základě vstupu nebo výstupu z oblasti informovat server o této skutečnosti. Poté se tato informace může využít pro automatizaci určitých úkonů v domácnosti.

Na základě předchozí analýzy bylo navrženo řešení a implementováno. Serverová část je jednoduše rozšířitelná pro zasílání notifikací na další platformy. Dále je rozebrán princip implementace příjímání notifikací a geolokace na zařízení s platformou Android.

Poslední část je věnována výsledkům testování. V průběhu vývoje byly pravidelně vydávány *alpha* a *beta* verze aplikace na *Google Play*. Aplikace měla po dobu vývoje dvacet aktivních instalací. Několik z uživatelů mělo systém nasazeno v reálném prostředí a aktivně využívalo. Nasazení přes *Google Play* a využití služby *Google Analytics* zjednodušilo hlášení chyb aplikace a pomohlo tak odhalit chyby, které se vyskytly při reálném nasazení.

Do budoucna se předpokládá rozšíření zpráv podle nových požadavků systému chytré domácnosti, na což je navržené řešení připravené. Z pohledu lokalizace by bylo zajímavé přesněji lokalizovat uživatele uvnitř budov.

Literatura

[1] Apple Inc.: Apple Push Notification Service [online]. 2014-10-31 [cit. 2015-01-05].
URL https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

[2] Apple Inc.: Provider Communication with Apple Push Notification Service [online]. 2015-03-09 [cit. 2015-05-03].
URL https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/CommunicatingWIthAPS.html#//apple_ref/doc/uid/TP40008194-CH101-SW6

[3] Bounegru, L.: Smart Houses: From Managing the House at a Distance to the Management of Life Itself. 2009-08-23 [cit. 2014-12-22].

[4] Ehsani, F.; Witt-Ehsani, S.; Rolandi, W.: Smart Home Automation Systems and Methods [online]. 2014-04-17 [cit. 2014-12-22], uS Patent App. 14/049,158.
URL http://www.google.com/patents/US20140108019

[5] Gann, D.; Barlow, J.; Venables, T.: *Digital futures: Making homes smarter*. Chartered Institute of Housing, 1999, ISBN 1-900396-14-9.

[6] Google Inc.: Google Cloud Messaging for Android [online]. [cit. 2015-01-03].
URL http://developer.android.com/google/gcm/gcm.html

[7] Google Inc.: Location APIs [online]. [cit. 2015-01-10].
URL http://developer.android.com/google/play-services/location.html

[8] Google Inc.: Receiving Location Updates [online]. [cit. 2015-01-10].
URL http://developer.android.com/training/location/receive-location-updates.html

[9] Google Inc.: Creating and Monitoring Geofences [online]. [cit. 2015-05-03].
URL https://developer.android.com/training/location/geofencing.html

[10] Harper, R.: *Inside the Smart Home*. Springer London, 2003, ISBN 978-1-85233-854-1.

[11] Microsoft: Push Notifications (Windows Phone) [online]. [cit. 2015-01-05].
URL http://msdn.microsoft.com/en-us/library/hh221549.aspx

[12] Microsoft: How to send and receive Tile notifications for Windows Phone 8 [online]. [cit. 2015-05-10].
URL https://msdn.microsoft.com/en-us/library/windows/apps/hh202970(v=vs.105).aspx

[13] Spicer, D.: If You Can't Stand the Coding, Stay Out of the Kitchen: Three Chapters in the History of Home Automation [online]. 2000-08-12 [cit. 2014-12-22].
URL http://www.drdobbs.com/architecture-and-design/if-you-cant-stand-the-coding-stay-out-of/184404040

[14] Valeš, M.: *Inteligentní dům*. ERA, 2006, ISBN 80-7366-062-8.

[15] Wikipedia: Apple Push Notification Service — Wikipedia, The Free Encyclopedia [online]. 2014 [cit. 2015-01-05].
URL http://en.wikipedia.org/wiki/Apple_Push_Notification_Service

Příloha A

Typy zpráv a jednotlivé zprávy

Upozornění, informace a reklamu řadíme mezi viditelné notifikace. U těchto zpráv je při přijetí vytvořena notifikace v notifikační liště. Tuto skutečnost již dále nebude uváděna u konkrétních zpráv.

A.1 Upozornění (alert)

SensorLowBatttery

- Upozornění na vybitou baterii.
- dodatečné parametry:
  - `int adapterId` - ID adaptéru
  - `string deviceId` - ID senzoru
  - `int battery` - stav baterie
- reakce na kliknutí:
  - zobrazit detail senzoru

SensorLowSignal

- Upozornění na nízkou úroveň signálu
- dodatečné parametry:
  - `int adapterId` - ID adaptéru
  - `string deviceId` - ID senzoru
  - `int signal` - úroveň signálu
- reakce na kliknutí:
  - zobrazit detail senzoru

AdapterOffline

- Upozornění na ztrátu konektivity mezi adaptérem a serverem
- dodatečné parametry:
  - `int adapterId` - ID adaptéru
- reakce na kliknutí:
  - otevření hlavní obrazovky s vybraným aktivním adaptérem

A.2 Informace (info)

SensorAdded

- Informační zpráva zaslaná v případě, že jiný uživatel domácnosti přidal na společný adaptér nový senzor.
- dodatečné parametry:
  - `int adapterId` - ID adaptéru
  - `string sensorId` - ID senzoru
- reakce na přijetí notifikace:
  - aktualizace seznamu senzorů
- reakce na kliknutí:
  - zobrazení detailu senzoru

AdapterAdded

- Informační zpráva zaslaná uživateli po přiřazení práv na nový adaptér.
- dodatečné parametry:
  - `int adapterId` - Id adaptéru
- reakce na přijetí notifikace:
  - aktualizace seznamu adaptérů uživatele
- reakce na kliknutí:
  - zobrazení hlavní obrazovky s aktivním novým adaptérem

Watchdog

- Po splnění podmínky hlídače je odeslána tato notifikace, která nese informaci o tom, které zařízení vyvolalo podmínku.
- dodatečné parametry:
  - `string message` - zpráva nadefinovaná uživatelem při vytváření pravidla
  - `int adapterId` - ID adaptéru
  - `string sensorId` - ID senzoru
  - `int type` - typ senzoru + offset
  - `int ruleId` - ID pravidla
- reakce na kliknutí:
  - otevření detailu senzoru
  - při možnosti akčních tlačítek možnost otevřít i pravidlo, které událost vyvolalo

A.3 Reklama (advert)

Uri

- Univerzální notifikace pro otevření odkazu
- Může se využít i *schema* cílové platformy (např. v Androidu URI ve formátu `market://details?id=<balik>` otevře aplikaci Google Play s detailem aplikace)
- dodatečné parametry:
  - `string uri` - odkaz pro otevření
  - `string message` - lokalizovaná zpráva ze serveru
- reakce na kliknutí:
  - otevření odkazu nebo příslušného programu podle *schema*

Achievement

- Zasílá oznámení o dosažení nové úrovně
- dodatečné parametry:
  - `int achievementID` - ID achievementu, který identifikuje určitý druh
- reakce na kliknutí:
  - zobrazení uživatelova profilu

A.4 Řídící (control)

DeleteNotification

- Při přečtení notifikace server rozesílá všem zařízením uživatele oznámení o přečtení
- dodatečné parametry:
  - `notificationId` - ID přečtené notifikace
- reakce na přijetí notifikace:
  - vymazání notifikace (pokud je stále notifikace viditelná - např. v notifikační liště)

Příloha B

Příklad použití knihovny pro Google Cloud Messaging

Pro překlad knihovny je vytvořen `Makefile`. Knihovna je přeložena pomocí příkazu `make`. Při překladu se vytvoří statická (libnotif.a) i dynamická knihovna (libnotif.so). Pro využití dynamické knihovny je nutné do překladu přidat:

```
-L<cesta_ke_knihvne> -lnotif -lcurl
```

Pokud není knihovna vložena do implicitního umístění knihoven systému, musí se přeložený program spustit následovně:

```
LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH  ./<nazev_prelozeneho_programu>
```

Příklad zdrojového kódu

```
Notification *not = new WatchdogNotif(
  12345,          // User ID
  1,              // Notification ID
  6164846,        // Unix timestamp (ms), kdy byla zpráva vytvořena.
  "Watchdog msg" // Zpráva, která bude zobrazena uživateli.
  cafe,           // Adapter ID
  1897,           // Senzor ID
  270,            // Typ + offset
  3               // ID pravidla, které notifikaci vyvolalo
);

// naplníme seznam Registration ID z DB
vector<string> ids;
ids.push_back("GCM_ID1");
ids.push_back("GCM_ID2");

not->sendGcm(&ids);

if(not->saveToDb()) {
  // připravené proměnné pro vložení do DB
  string text = not->getDbXml();
```

```
  int level  = not->getLevel();
  int userId = notif->getUserId();
  long timestamp = notif->getTime();
  int message_id = notif->getId();
  string name = notif->getName();
  boolean isRead = false;

  // zde by proběhlo samotné uložení do DB
}
```

Příloha C

Obsah CD

src/server/ složka se zdrojovými soubory serverové části

src/android/ složka zdrojovými soubory pro platformu Android

apk/beeeon.apk zkompilovaná aplikace

zprava/zprava.pdf zpráva ve formátu PDF

zprava/manual.pdf manuál

zprava/tex/ složka se zdrojovými soubory diplomové práce ve formátu LaTeX