VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

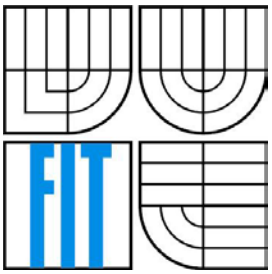
OPTIMALIZACE KOMUNIKAČNÍCH PROTOKOLŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Petr Buno
AUTHOR

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

OPTIMALIZACE KOMUNIKAČNÍCH PROTOKOLŮ

COMMUNICATION PROTOCOL OPTIMIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Petr Buno

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Jan Kořenek, Ph.D.

BRNO 2015

Abstrakt

Tato práce se zabývá problematikou inteligentní domácnosti, analyzuje existující řešení a zaměřuje se na systém vyvíjený ve spolupráci s FIT VUT v Brně. Důraz je kladen na popis klientské aplikace pro ovládání a monitoring systému chytré domácnosti. Dále jsou v práci rozebrány základní teoretické poznatky z oboru internetové komunikace, které úzce souvisí se zmíněnou problematikou. Jádrem práce je návrh komunikačního protokolu mezi serverem a mobilním zařízením. V neposlední řadě práce obsahuje také popis implementace síťové vrstvy na straně koncového zařízení používající tento protokol. Poslední část práce je věnována testování a vyhodnocení finálního řešení.

Abstract

This thesis deals with smart home issue, analyzes existing solutions and is focused on the system that is developed in collaboration with FIT VUT in Brno. The emphasis is on the description of client application for control and monitoring of smart home system. Furthermore, the work discussed base theoretical knowledge from internet communication field, which is closely related to these issues. The core of the work is to design a communication protocol between the server and a mobile device. Finally, the work also includes a description of a network layer at the device using this protocol. The last part is devoted to testing and evaluation of the final solution.

Klíčová slova

Domácí automatizace, inteligentní domácnost, komunikační protokol, zabezpečená komunikace

Keywords

Home automation, smart home, communication protocol, secure communication

Citace

Buno Petr: Optimalizace komunikačních protokolů, diplomová práce, Brno, FIT VUT v Brně, 2015

Optimalizace komunikačních protokolů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jana Kořenka, Ph.D. Další informace mi poskytli členové týmu zabývajícím se projektem inteligentní domácnosti. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Buno
27. 7. 2015

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu diplomové práce, Ing. Janu Kořenkovi, Ph.D. za odborné vedení a pomoc, ochotu a množství přínosných podnětů. Také bych chtěl poděkovat celému týmu inteligentní domácnosti za podporu a spolupráci. A v neposlední řadě své rodině za důvěru.

Obsah

1 Úvod

V dnešní době rychlého životního stylu, kdy jsou na každého z nás kladeny vysoké nároky, spoléháme víc než kdy dříve na technologie kolem nás. Lidé jsou tak stále více spojení s okolním světem pomocí chytrých zařízení, jako jsou mobilní telefony, tablety či hodinky schopné komunikace přes internet. Lze předpokládat, že tento trend bude dále pokračovat a uživatelů bude nadále přibývat jak v oblasti spotřební elektroniky, tak v oblasti služeb na internetu.

Odděleně od tohoto sektoru již delší dobu existuje a rozrůstá se obor, který jde naproti lidskému pohodlí a vyššímu standardu bydlení a života. Tento obor, obecně známý jako domácí automatizace, sahá do dob daleko před proniknutím internetového světa do našich všedních životů. Každý člověk ocení usnadnění nejen v práci, ale i v domácnosti. Bylo jen otázkou času, kdy se tyto dva segmenty protnou a společně zesílí, aby společně zaútočily na naše domácnosti. Společně s rozvojem pokročilých senzorů a dalších chytrých zařízení se tento sektor stává součástí pohledu tzv. „*Internet of Things*“, který spojuje všechna zařízení sítí, po které spolu mohou komunikovat.

Díky tomuto principu lze veškerá připojená zařízení kontrolovat (pasivní přístup) nebo přímo vzdáleně ovládat (aktivní přístup). Většina zařízení a systémů je však schopna se sama řídit a postupně se tak ve společnosti začíná uchycovat název „*inteligentní domácnost*“. Velká vlna nových zařízení a systémů se vzedmula v letošním roce na konferenci CES (Consumer Electronic Show) v Las Vegas, kde představilo hned několik velkých firem svá řešení pro ovládání domácnosti budoucnosti [9]. Tyto firmy se zaměřují na několik aspektů a požadavků uživatelů, mezi které patří hlavně komfort (jednoduchost, intuitivnost, snadné ovládání, rychlé použití, nízké nároky na údržbu). S komfortem dále souvisí variabilita a nároky na zprovoznění, kde kvalitnější a propracovanější jsou vestavěná řešení, zatímco skládačkové systémy jsou několikrát levnější. Samozřejmě důležitým kritériem je i spektrum podporovaných zařízení, pořizovací cena nebo rozšířená zákaznická podpora.

Nejen tato skutečnost je jasným znamením, že je tento směr tím, který v budoucích letech dobude naše domovy a veškerá chytrá zařízení kolem nás. To bylo jedním z důvodu, proč jsem se zaměřil na vývoj systému pro ovládání chytré domácnosti. S tím souvisí tato práce, která si klade za cíl blíže popsat zmíněnou problematiku, prostudovat možnosti komunikace a především se zaměřit na návrh a následnou implementaci komunikačního protokolu, který bude použit pro ovládání prvků inteligentní domácnosti z prostředí mobilního telefonu, či jiného chytrého zařízení.

Jak bylo řečeno, velkou roli zde hraje návrh komunikačního protokolu, což v sobě obnáší volbu technologií použitých k přenosu informací, strukturu samotných zasílaných zpráv a také volbu dat, která jsou potřeba pro komunikaci a jejich vhodné reprezentace. Tyto aspekty mají vliv nejen na složitost výsledného kódu, ale především na latenci systému, která je rozdělená na latenci sítě a dobu zpracování požadavku. Toto zpracování požadavků si vybírá svojí daň ve formě využití procesorového výkonu jak na straně klienta tak i serveru a s tím související spotřebu energie.

V této práci jsem se zaměřil především na vhodný návrh z pohledu bezpečnosti, spotřeby a latence. Mezi další požadavky, které jsem bral v úvahu, patří modularita, rozšiřitelnost a také jsem zahrnul požadavky na multiplatformnost a znovupoužitelnost.

Text práce je rozdělen do několika logických celků. Kapitola 2 blíže definuje pojem a principy inteligentní domácnosti. Přináší základní pohled na možnosti rozdělení problematiky do několika kategorií a představuje několik vybraných existujících řešení, která již vstoupila na český či světový trh. Dále je v této kapitole popsán systém vyvíjený v rámci spolupráce s FIT VUT v Brně, jehož

součástí je právě tato práce. Systém je rozdělen na jednotlivé části, u kterých je vysvětlena jejich funkcionalita, princip a základní vlastnosti a technologie, na kterých je postavena. Konec druhé kapitoly je zaměřen na mobilní aplikaci pro ovládání zmíněné inteligentní domácnosti, která úzce souvisí vyvíjeným protokolem. Následuje kapitola 3, která po teoretické stránce popisuje komunikační modely, a protokol pro zabezpečenou komunikaci. Plynule navazující kapitola 4 blíže specifikuje protokoly a textové formáty, které úzce souvisí se síťovou komunikací aplikace a s návrhem, vývojem a údržbou komunikačního protokolu. Kapitola 5 se zabývá specifikací požadavků systému inteligentní domácnosti, koncového ovládacího zařízení (chytrý telefon) a uživatelů. Kromě specifikace požadavků, je zde i nastíněn návrh konečného řešení pro protokol. Kapitola 6 se věnuje popisu implementace řešení navrženého v předchozí kapitole. Je zde popsán princip, jakým funguje submodul pro zpracování XML a jeho dvě hlavní součásti, XmlCreator a XmlParsers. V neposlední řadě tato kapitola vysvětluje postup, jakým funguje síťová vrstva realizovaná třídou Network. Výsledky praktické části práce, jsou diskutovány v kapitole 7, kde je mimo jiné podkapitola věnovaná testování a experimentům s finální aplikací se zaměřením na ověření splnění podmínek definovaných v kapitole návrhu. Především pak latenci síťové komunikace a velikost zasílaných a přijímaných zpráv. Následuje závěrečné shrnutí a zhodnocení práce a diskuze o dalších možnostech vývoje, kterým se může aplikace dále ubírat.

Pro velkou rozsáhlost je kompletní návrh komunikačních zpráv spolu s ukázkami a vysvětlením jednotlivých prvků umístěn v příloze této práce.

2 Inteligentní domácnost

Pojem inteligentní domácnost není nijak nový. O jistou úroveň automatizace domu, kanceláře nebo chaty se různá řešení pokouší už desítky let. Hlavním cílem všech těchto systémů je především usnadnit obyvatelům užívání těchto prostor, zvýšit jejich bezpečnost a snížit výdaje za energie při zachování stávajícího komfortu. Splnit tyto požadavky se snaží například pomocí elektrifikace mechanických částí, jako jsou okna, žaluzie, dveře, vrata, brány, zámky a mnoho dalších. Dále pak automaticky řízené vytápění, větrání a zavlažování bez dohledu člověka podle daných podmínek a scénářů. V neposlední řadě také vzdáleným ovládáním či monitorováním objektů skrze internet nebo jinou komunikační síť [1].

Tyto systémy se dají dělit podle několika kritérií a specifických vlastností. Například rozdělení podle typu instalace tzn., zda se jedná o vestavěné řešení již od počátku, nebo bylo dodáno až později. S tímto zpravidla souvisí především čistota řešení a úroveň s jakou je systém s budovou propojen. Komplexnosti těchto vestavěných řešení ale odpovídá i jejich cena. Na rozdíl od toho systémy stavěné či skládané (stavebnicové) samotnými uživateli jsou řádově levnější (jednotky tisíc vs. desítky až stovky tisíc), ale často tvořené zařízeními různých výrobců a nemusí tedy být vždy plně kompatibilní.

Za jiné kritérium dělení lze považovat samotnou úroveň automatizace a možnosti ovládání, což v sobě obnáší velké množství kombinací. Z laického pohledu je systém plně automatizovaný, polo-automatizovaný nebo plně v rukou uživatele. I když výrok „plně v rukou“ není v dnešní době přesný, jelikož spoustu systémů lze ovládat hlasem, zvukem nebo pohybem a dalšími metodami. Především pak použití přenositelných zařízení jako jsou notebooky, tablety a telefony je velmi rychle rostoucím segmentem na úkor centrálních řídících jednotek. Na druhou stranu plně automatizovaný systém není pro domácnost příliš vhodný z důvodu prozatím nedostatečně rozvinuté řídící jednotce (umělé inteligence) a jeho rozmach tak nelze v dohledné době očekávat. Naopak poslední ze zmiňovaných typů (hybridní) je pro domácí uživatele nejzajímavější. Nabízí jak možnost ovládat dům přímo, tak pomocí nastavených pravidel, která jsou uloženy v PC, centrální jednotce nebo například na serveru.

Jak bylo řečeno, tento sektor je již delší dobu cílem zájmu velkého počtu společností a díky novým technologickým trendům se požadavky stále vyvíjí a díky snižujícím se nákladům na stavebnicový typ se stává inteligentní domácnost pro masy dostupnější. Příkladem nových technologií, které přímo ovlivňují tento segment, jsou mobilní zařízení, pomocí kterých může být dům neustále monitorován jeho majitelem, ale i dům, může monitorovat díky GPS a dalším metodám pohyb majitele. Tyto informace pak může použít například k tomu, aby začal vytápět byt, ve chvíli kdy se majitel vrací z práce, nebo zamknout dveře, když se vzdálil na větší vzdálenost.

2.1 Současná řešení

V této podkapitole představím několik existujících řešení systémů inteligentní domácnosti, pokusím se poukázat na jejich výhody a nevýhody a nastínit jejich základní koncept. Mezi mnou vybraná řešení pro srovnání patří **Haidy**, **CannyNet**, **Belkin Wemo**, **Elektrobock**, **Insteon** a v neposlední řadě **Insight Home**, které jsem zvolil, aby tvořili široké spektrum různých přístupů. V popisu nebudu

zabíhat do zbytečných detailů, protože se jedná o obecné srovnání systémů zaměřených na různé cílové skupiny.

Haidy

Tento komerční systém je, vzhledem k náročnosti montáže, zaměřen spíše na novostavby a komplexní řešení inteligentní domácnosti než na skládačkový model. Jde o částečně autonomní systém, který se dokáže zotavit z výpadku centrální jednotky. Cena základního řešení začíná na 30 000 Kč, což ho řadí spíše mezi dražší varianty [10]. Mezi základní vybrané výhody a nevýhody z mého pohledu patří:

Výhody
- Drátové i bezdrátové senzory
- Detekce úniku vody
- Možnost rozšíření a aktualizací

Nevýhody
- Vyšší pořizovací cena
- Menší množství senzorů a aktorů
- Převážně pro novostavby

CannyNet

Také tento systém je schopen fungovat i po selhání řídící jednotky. Zajímavostí je, že je postaven na mikromodulech s podporou PCLBUS technologie (komunikační protokol využívající silové rozvody elektřiny 230V). Pro další komunikaci používá MODBUS protokol (otevřený protokol pro komunikaci různých zařízení na různých sítích a sběrnicích), TCP/IP a HTTPS [11].

Výhody
- Modulární struktura
- Použití otevřených standardů
- Zabezpečená komunikace pomoci HTTPS

Nevýhody
- Zaměření pouze na iPad a iPhone
- Spíše firemní řešení

Belkin WeMo

Toto řešení je zaměřeno spíše na menší stavebnicové prvky. Lze dokupovat jednotlivé koncové elementy, které se dají ovládat chytrým telefonem prostřednictvím Wifi. Podporuje základní aktory jako jsou zásuvky a světla, ale i méně rozšířené jako jsou hlídače mazlíčků a kuchyňské nádobí [12].

Zajímavostí jsou žárovky, které lze ovládat i pomocí definovaných skupin, či kamera, která je schopna na základě pohybu tyto žárovky rozsvěcet. Chytrý hrnec Crock-Pot nebo kávovar Mr.Coffee lze dálkově vypínat a zapínat nebo kontrolovat teplotu a délku vaření. Díky využití služby IFTTT[1] je možné definovat vlastní scénáře a kombinovat je s jinými službami.

Výhody
- Nižší cena ⇔ stavebnicový systém
- Atypické senzory (chytrý hrnec)
- Multiplatformní (Android, iPhone)
- Spolupráce s IFTTT

Nevýhody
- Není komplexní řešení
- Není příliš lokalizovaný
- Nízké hodnocení aplikací uživateli

[1] Dostupné z: https://ifttt.com

Elektrobock

Jde o českou společnost používající systém PocketHome. Jednotlivé prvky komunikující bezdrátově na frekvenci 433 MHz. Řešení je zaměřeno na ovládání vytápění, osvětlení a zabezpečení domů. Pro vzdálený monitoring je použita GSM síť [13].

Výhody

- Nižší cena
- Vlastní řešení komunikace
- Česká firma

Nevýhody

- Není využito ovládání přes internet
- Úzké zaměření (kotle, světla, apod.)

Insteon

Dalším systémem s velkou plejádou chytrých zařízení je Insteon, který se pokládá za kompletní řešení stavebnicového typu. Nabízí koncová zařízení nejrůznějších typů, od světel a zásuvek, přes kamery a spínače, až po termostaty [14]. Centrálním prvkem systému je Insteon Hub, ke kterému se připojují ostatní elementy a mohou skrze něj dále komunikovat přes internet.

Systém je částečně otevřený pro vývojáře třetích stran, kteří mohou pro komunikaci s domácností použít rozhraní REST (Representational State Transfer).

Výhody

- Nižší cena
- Využití všech mobilních platforem
- Široká nabídka senzorů a aktorů

Nevýhody

- Lokalizace
- Nízké hodnocení aplikací uživateli

Insight Home

Insight Home je dalším komplexním řešením na míru zahrnujícím vše od bezpečnosti, řízení osvětlení a vytápění, až po zábavu (mediální centrum). Jde o české řešení postavené na systému inHome AMX, který podporuje i pasivní technologie s ohledem na ekologii [15].

Výhody

- Komplexní řešení
- Velký důraz na zabezpečení osob i majetku
- Podpora komunikace mezi uživateli
- Robustní řešení

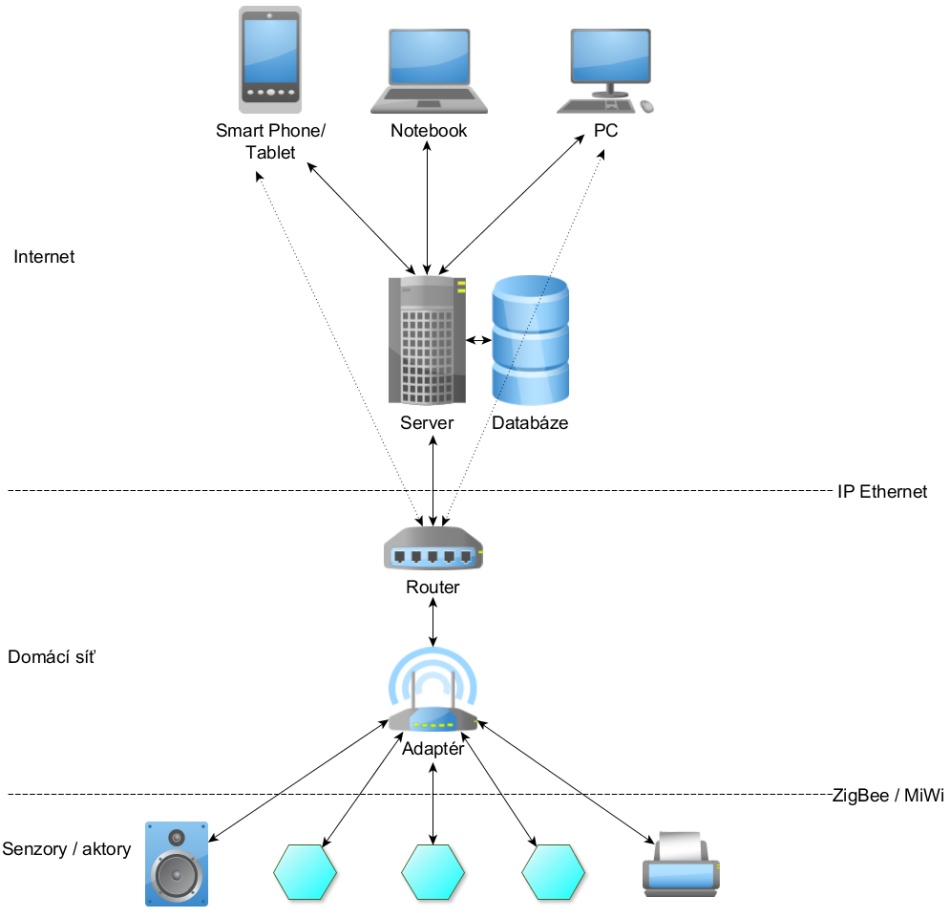
Nevýhody

- Vysoká pořizovací cena
- Nutnost zajištění infrastruktury

2.2 Systém BeeeOn

Základní koncept řešení inteligentní domácnosti vyvíjeného v rámci projektu IOT ve spolupráci s FIT VUT Brno, pracovně nazývaný BeeeOn, je postaven na stavebnicovém typu, kdy si uživatelé budou moci kupovat jednotlivá chytrá zařízení postupně. Jedná se o hybridní systém, tudíž ho lze ovládat jak například pomocí chytrých telefonů, tak pomocí nastavených pravidel spouštěných z centrálního serveru.

Pro snadnější pochopení je vhodné celý systém dekomponovat na menší funkční bloky jak je ukázáno na obrázku 2.1. Smysly domácnosti jsou **senzory**, které monitorují vlastnosti svého okolí jako například teplotu, vlhkost či tlak. Pro ovládání elektrických či mechanických spotřebičů jsou k dispozici **aktuátory** (aktory) popsané v kapitole 2.2.1. Další důležitou částí je **adaptér**, který komunikuje se **senzory** a **aktuátory** a distribuuje požadavky výše postavených prvků systému. Pokud bych použil přirovnání k lidskému tělu, pak by adaptér plnil funkci nervové soustavy. Bližší popis **adaptéru** je v kapitole 2.2.2. Mezi další části patří **serverová část**, kde jsou uložena všechna data od uživatelů až po hodnoty **senzorů**. Jedná se tedy o mozek celého systému. **Server** komunikuje s **adaptéry** a s **koncovými klienty** pro ovládání, jako jsou mobilní telefony, webové prohlížeče či jiné řídící prvky. **Serverem** se zabývá kapitola 2.2.4 a **koncovými prvky** pro ovládání pak kapitola 2.2.5 a blíže kapitola 2.3.



Obrázek 2.1: Funkční bloky BeeeOn systému

2.2.1 Senzory a aktuátory

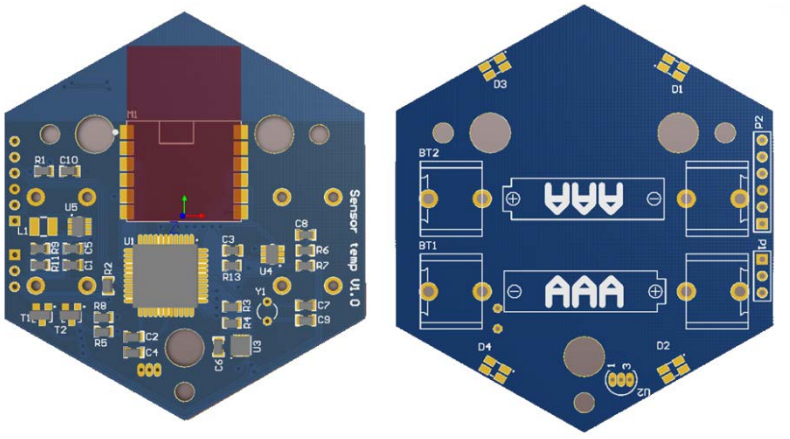
Jedná se o koncová chytrá zařízení, která dokážou komunikovat s **adaptérem** pomocí bezdrátové sítě MiWi. Jde o nejnižší vrstvu architektury, která snímá veličiny okolí (senzory teploty, osvětlení, vlhkosti, CO_2 a další) anebo své prostředí ovládá (aktuátory). Systém je navržen tak, aby bylo později možné využívat i koncová zařízení třetích stran. I přesto, že se jedná o chytrá zařízení, jsou zkonstruována, aby byla co nejjednodušší, s úzkým zaměřením a co nejlevnější a lehce nahraditelná v případě poškození nebo poruchy. Tudíž na sobě senzory nemají žádná tlačítka ani mechanické přepínače a s uživatelem komunikují pouze pomocí LED diod a pohybem, resp. zatřesením, díky vestavěnému akcelerometru, který v sobě senzory mají.

Před samotným zasíláním dat skrze **adaptér** je nutné zařízení s příslušným **adaptérem** spojit. Takzvané spárování lze provést jen tehdy, má-li uživatel již adaptér registrován, aby mohl tento **adaptér** přepnout do párovacího režimu, kdy adaptér skenuje danou síť, aby v ní odhalil nová zařízení. Samotný senzor je spuštěn zapojením baterií a do sítě začíná vysílat po zatřesení (aktivace akcelerometru). Každý senzor má přidělený unikátní identifikátor (IPv6/MAC), aby jej bylo možné jednoznačně odlišit od ostatních zařízení a také ověřit jeho validitu.

Po úspěšném spárování a odeslání prvních hodnot se senzor přepíná z párovacího módu do režimu spánku, ve kterém setrvá do vypršení nastaveného časového intervalu, kdy se zase probouzí a probíhá komunikace s **adaptérem** obsahujícím nová data. Následně se zase uspí a tento životní cyklus je stále opakován. Tato strategie umožňuje senzoru co nejvíce šetřit baterii a prodloužit tak jeho výdrž na několikanásobně delší dobu.

Senzor kromě naměřených dat odesílá na **adaptér** i hodnotu baterie, díky čemuž může být uživatel včas upozorněn na její nízký stav. Při této komunikaci může adaptér nově nastavit délku časového intervalu spánku senzoru. V případě aktuátorů je navíc zaslána informace, jestli má být obvod sepnut. Aktuátory nepřechází do režimu spánku, aby byly kritické prvky, které ovládají, vždy k dispozici, čehož lze dosáhnout pomocí připojení na zdroj elektrické energie, který řídí. Právě díky této skutečnosti odpadá i problém s baterií. Další výhodou je, že je zařízení stále připojeno a lze jej tak použít i jako retranslační stanici (posilující nebo znovu zasílající signál) a zvýšit tím dosah domácí sítě **adaptéru**.

Pro senzory byl zvolen netradiční šestiúhelníkový tvar mající připomínat buňku včelí plástve. Tento vzor je zobrazen na obrázku 2.2, na kterém je technický návrh plošného spoje základní desky senzoru [2].



Obrázek 2.2: Plošný spoj senzoru

2.2.2 Adaptér (brána)

Adaptér je další funkční blok spojující všechny **senzory** a **aktuátory** s ním spárované. Na druhé straně komunikuje skrze **router** se **serverem**, který zabezpečuje vyšší logiku. Na rozdíl od **senzorů** musí být adaptér neustále k dispozici, aby přijímal data od **senzorů** a pokyny od **serveru**. Tudíž musí být adaptér stále napájen a nelze efektivně využít zdroj, jako jsou baterie. Adaptér je sofistikovanější než koncová zařízení, díky čemuž může obsluhovat v budoucnu i elementy třetích stran. Navíc v případě výpadku **serveru** nebo ztráty konektivity musí být schopen lokálně řídit domácnost. Adaptér může být připojen několika způsoby. Může být připojen pomocí standardních rozhraní jako je USB, Ethernet nebo Wifi, ale může být i součástí samotného **routeru** (směrovač). Za přidanou hodnotu lze pokládat otevřenost adaptéru, což umožňuje majiteli do jeho systému přidat softwarové aplikace [2].

Stejně jako u koncových zařízení obsahuje adaptér, kromě potřebných periferií k připojení k routeru, pouze LED diody informující o jeho stavu. Adaptér se tak po připojení do elektrické a internetové sítě okamžitě hlásí na příslušný **server**. Spuštění párovacího režimu také vyžaduje interakci skrze **server**.

V případě výpadku konektivity se **serverem** a nutnosti ukládat data zaslaná **senzory**, má adaptér možnost ukládat informace na microSD kartu, která je již vložena do slotu na základní desce. Slot lze vidět na obrázku 2.3 vpravo. Dále je na levé straně odshora konektor na ethernet (až 100 Mb), dvakrát vysokorychlostní USB port a konektor pro zdroj napájení. Mezi další důležité prvky patří hlavní čip disponující A10 Cortex-A8 CPU (procesor) a Mali 400 GPU (grafická jednotka).

Obrázek 2.3: Základní deska adaptéru

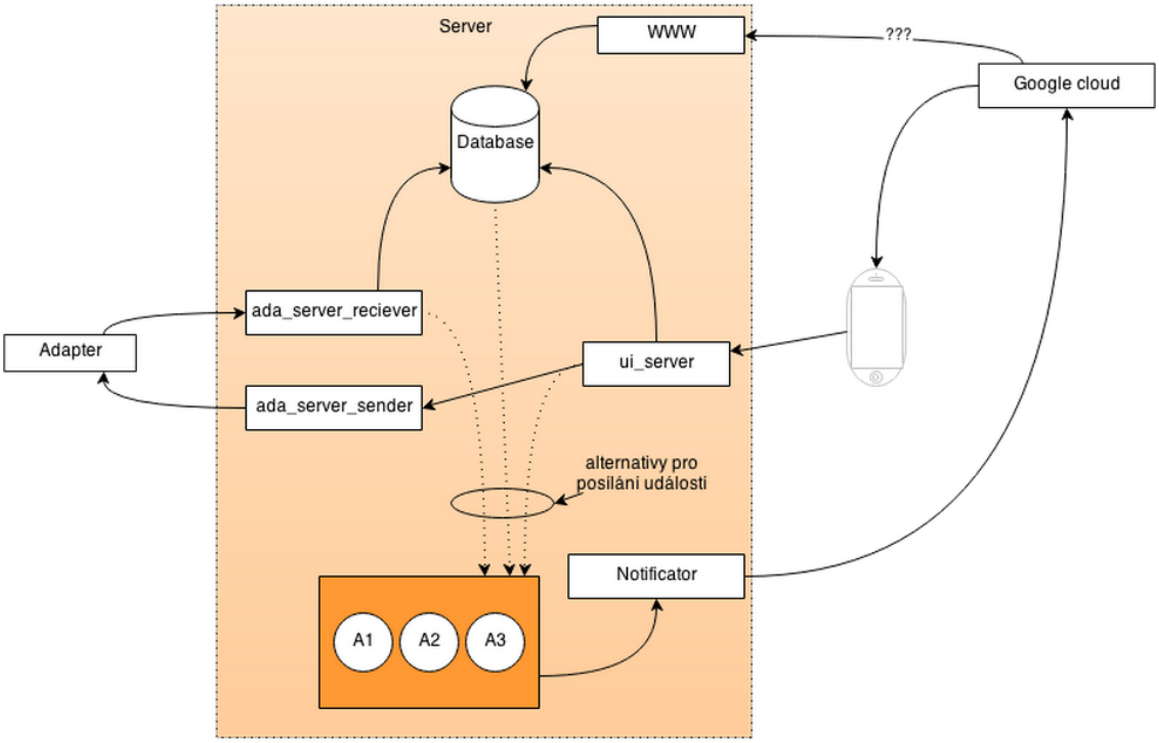
2.2.3 Směrovač (router)

Jde o část systému zabezpečující internetovou konektivitu pro **adaptér**. Každý koncový ovládací prvek se připojuje k **adaptéru** právě skrze router. V současnosti jsou ovládací prvky připojovány přes internet skrze **server**, ke kterému se **adaptér** po zapojení hlásí a sdělí mu tak svou IP adresu. Do budoucna je však počítáno i s lokálním ovládáním bez nutnosti vnější internetové konektivity, například s pomocí chytrého telefonu [2].

2.2.4 Proxy server

Pokud nelze zabezpečit dostupnou IP adresu pro **router**, je třeba zajistit komunikaci skrze proxy server, který zprostředkuje síťové spojení. Proxy server má přidělenou veřejnou IP adresu, ke které se mohou připojovat koncové ovládací prvky, jako jsou například mobilní nebo webové aplikace.

Kromě této datové cesty může serverová vrstva disponovat komplexnějšími funkcemi: zabezpečovat celou aplikační logiku inteligentní domácnosti, poskytovat predikční služby na základě naměřených hodnot a předpovědi počasí a v neposlední řadě zálohu dat. [2]



Obrázek 2.4: Diagram vnitřní struktury serveru

V případě BeeeOn systému je za server pokládán ne jeden, ale několik kooperujících programů běžících na fyzickém serveru. Jak zobrazuje diagram na obrázku 2.4, s **adaptérem** komunikují programy *ada_server_sender* (pro zasílání požadavků a nastavení) a *ada_server_reciever* (příjem dat a řídících zpráv), který tato data ukládá do *databáze*. Do *databáze* ukládá i *ui_server*, který tvoří rozhraní pro komunikaci s koncovými ovládacími prvky, jako jsou například mobilní telefony. Kromě ukládání předává *ui_server* získané požadavky na vstup *ada_server_senderu*. Mezi další nekritické součásti patří *webové rozhraní* a *notifikátor*, který, pokud jsou splněny nastavené podmínky, zasílá na mobilní telefony notifikace.

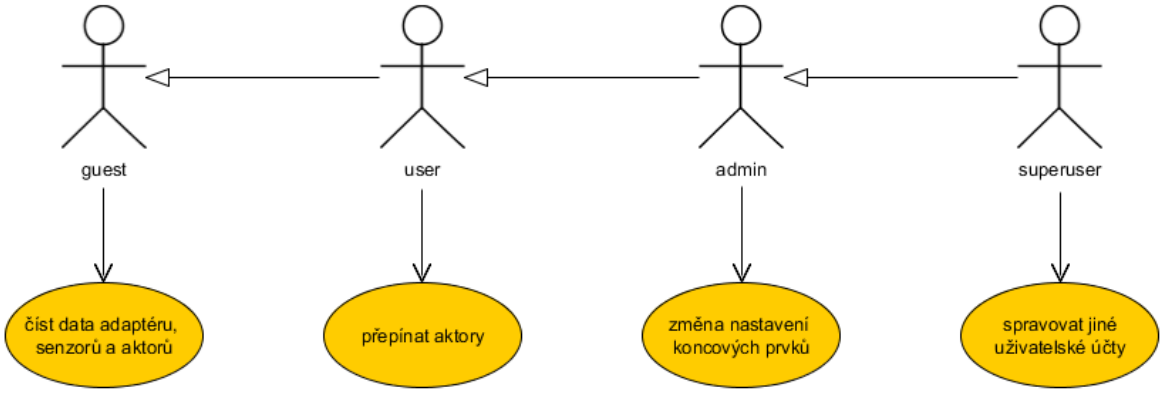
Důležitou funkcionalitou serverové části je i kontrola práv uživatelských účtů. Každému uživateli je při vstupu do systému přidělena role pro **adaptéry**, které má k dispozici. Tato role je mu přidělena buď defaultně, pokud je majitelem daného **adaptéru**, nebo jiným uživatelem majícím dostatečná práva na správu uživatelských účtů. Hierarchie rolí je zobrazena na obrázku 2.5.

Nejníže je postavený uživatel s právy role *host* (*guest*). Tomuto uživateli povolí **server** pouze číst data **adaptérů**, jako jsou jméno a seznam jeho koncových prvků a hodnoty těchto elementů (hodnoty senzorů a stavy aktorů). Tato role by se dala přirovnat k někomu, kdo přijde na několikadenní návštěvu, nebo k dětem.

O stupínek výše je role *běžný uživatel* (*user*), která již smí navíc oproti *hostovi* přepínat aktuátory. Tato role bude defaultně přiřazena novým uživatelům, kteří nejsou majiteli onoho **adaptéru**. Předpokládá se, že většina uživatelů bude mít standardně nastavenou právě tuto roli.

Dalším stupněm je uživatelská role *správce* (*admin*). Tito uživatelé mají téměř veškerá práva k manipulaci s inteligentní domácností. Kromě řízení aktorů mají práva ke změně nastavení veškerých **senzorů** a **aktorů**, jako jsou intervaly spánku, umístění v místnostech, přejmenování koncových prvků i místností. Dále pak nastavování podmínek a akcí na **serveru**, tedy definice scénářů, co se má za určitých podmínek stát (akce - vypnout/zapnout, notifikace, atp.). Tuto roli by měl mít jeden až dva uživatelé v domácnosti.

Poslední a nejvyšší rolí je *majitel* (*superuser*), který je vlastně první rolí na daném **adaptéru**. To znamená, že první uživatel **adaptéru** se automaticky po spárování účtu s **adaptérem** stává *majitelem*. Oproti ostatním rolím má navíc možnost spravovat uživatelské účty, což znamená, že je jediný, kdo může přidávat nové uživatele. Ale také může ty aktuální odstranit či jim změnit (zvýšit, snížit) jejich práva. Typicky je *superuser* právě jeden, ale může jich být i více, nikoli však méně než jeden. V případě, že chce uživatel s touto rolí degradovat svoje práva a za předpokladu, že je jediným *majitelem*, musí nejprve zvolit někoho, kdo jej nahradí. Pokud se uživatel dokonce odstraní, bude automaticky zvolen nový majitel na základě posloupnosti rolí a času získání této role. Krajní možnost je, že žádný další uživatel nezbyl, pak se **adaptér** stává opět nezaregistrovaným.



Obrázek 2.5: Role v BeeeOn systému

2.2.5 Koncové ovládací prvky

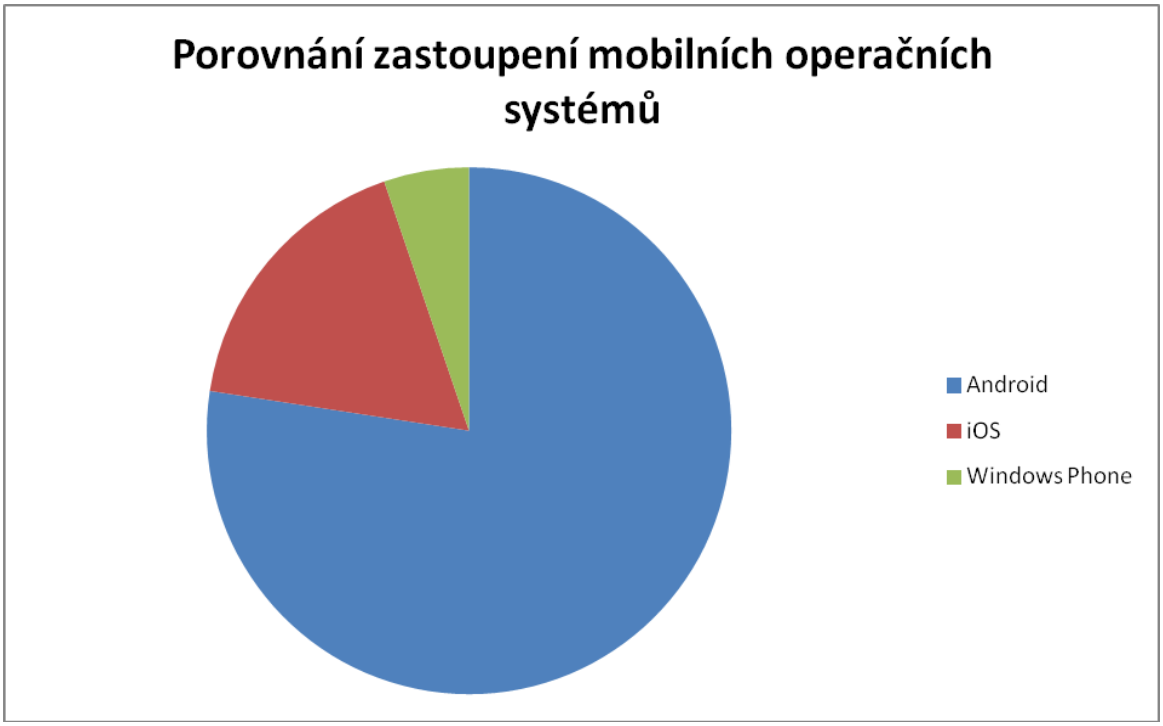
Jak zobrazuje obrázek 2.1, je zde několik možností jak ovládat BeeeOn systém koncovými prvky. Hlavním cílem tohoto řešení bylo zaměřit se na chytré telefony a tablety, protože jsou dnes masově rozšířené a uživatelé jim dávají přednost před centrálními jednotkami či dálkovými ovladači. S vědomím, že největší podíl na trhu má v současnosti operační systém Android, byl právě on zvolen jako hlavní platforma a vlajková loď ovládacích prvků. Z porovnání tří hlavních mobilních platforem, které je ukázáno na grafu 2.1, je evidentní majoritní postavení zařízení s Androidem na českém trhu s téměř tříčtvrtinovým podílem. Na druhém místě figuruje iOS od společnosti Apple s přibližně 16% a na třetím místě je Windows Phone s 5% zastoupení (měřeno na základě přístupů na web pomocí služby StatCounter[2] v období od července 2014 do července 2015).

A právě Windows Phone patří mezi další možnosti jak ovládat inteligentní domácnost, které jsou momentálně k dispozici. I přes menší zastoupení, než které má iOS byl zvolen kvůli větší

[2] Dostupné z: http://gs.statcounter.com/#mobile_os-CZ-monthly-201406-201506-bar

otevřenosti při vývoji a především, protože díky universálnímu vývoji lze bez větších úprav použít jádro aplikace nejen pro běh nad Windows Phone 8.1, ale také na tabletech a desktopech s operačním systémem Windows 8.1.

Posledním aktuálně dostupným řešením jak domácnost ovládat je webové rozhraní pro klasický internetový prohlížeč. Aplikace z celé palety kromě webového prohlížeče používají stejný komunikační protokol.



Graf 2.1: Porovnání zastoupení mobilních operačních systémů

2.3 Aplikace pro mobilní zařízení - BeeeOn klient

Jak bylo zmíněno v kapitole 2.2.5, hlavním cílem pro ovládací prvky je chytrý telefon. V této kapitole bude dále popsáno řešení klientské aplikace pro operační systém Android (pro Windows Phone a Windows je princip naprosto totožný).

Z požadavků na ovládání systému vyplývá, že klientská aplikace musí komunikovat skrz internetovou či jinou bezdrátovou síť se **serverem** nebo přes **router** přímo s **adaptérem**. Proto musí aplikace obsahovat komunikační modul, který bude získávat data o inteligentní domácnosti a zasílat pokyny uživatele zpět. Dnešním trendem je během krátké doby výrazně měnit styl grafické podoby systémů a aplikací. Z tohoto důvodu je dobrým řešením oddělit logickou část aplikace od jejího grafického rozhraní. Z výše řečeného vyplývá, že by aplikace měla obsahovat minimálně 3 kritické prvky pro snadný chod a pozdější úpravy. Na nejnižší vrstvě by měl stát **modul pro síťovou komunikaci**, dále **kontroler** pro zajištění logiky a v neposlední řadě **grafické uživatelské rozhraní**, které bude majiteli zprostředkovávat intuitivní cestou funkce ke správě jeho domácnosti.

Stejně, jako lze rozdělit celý BeeeOn systém, lze na funkční bloky dekomponovat i klientská aplikace. Na obrázku 2.6 je zobrazen model s jednotlivými vrstvami, které spolu komunikují přes definované rozhraní, a není tak problém měnit pouze jeden blok bez nutnosti změny ostatních. Nejníže postavená je **síťová vrstva**, která komunikuje přímo s **BeeeOn serverem**, případně s jinými

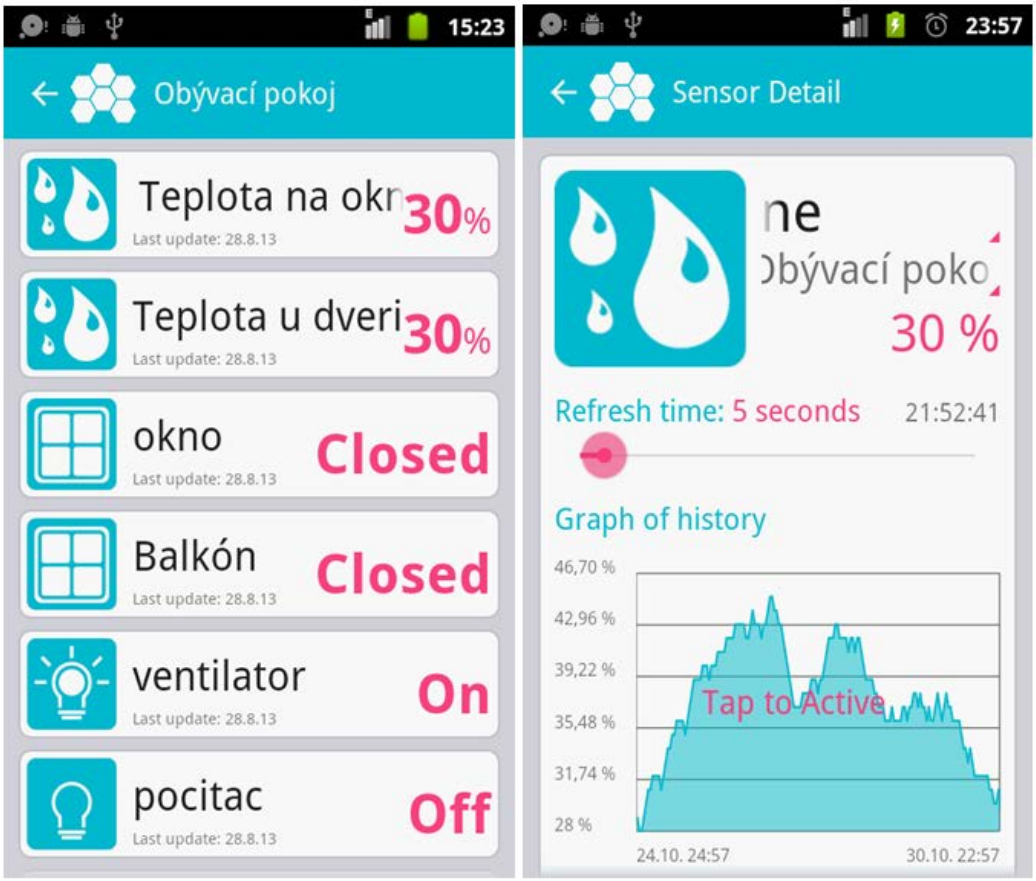
cloudovými službami jako je například Google či Facebook a to přes obecná rozhraní. Tato vrstva bude mít do budoucna za úkol komunikovat také přímo s **adaptérem** skrze **router**. Protože pro komunikaci je použit protokol ve formátu XML, je zde komponenta pro jeho vytváření a zpracování. Je však dostatečně oddělena na to, aby ji bylo možno kdykoliv nahradit za jiný typ (např. JSON).

Obrázek 2.6: Schéma funkčních bloků Android aplikace

Komunikační vrstvu používá přímo **kontroler**, který skrze tento modul získává data o inteligentní domácnosti a ukládá nastavení na **server**. **Kontroler** také řídí vnitřní stav aplikace a ukládá je společně s ostatními důležitými daty do lokálního úložiště díky komponentě pro perzistentní uložení. Opět se jedná o vlastní blok pro snadnou náhradu za jiný typ (např. lokální databáze). Na druhé straně komunikuje kontroler s **grafickým rozhraním**, které částečně zaštiťuje i komponenty pro widgety a notifikace. Díky tomuto rozdělení akcí je například možné spouštět více požadavků na **server** ve vlastních vláknech, protože v případě Androidu není možné spustit vlákno přímo z vlákna řídícího uživatelské rozhraní.

Obrázek 2.7: Logo Aplikace

Pro **GUI** (grafické uživatelské rozhraní) byl v případě Android aplikace zvolen standardní vzhled Material designu, jak lze vidět na Obrázek 2.8. Na tomto obrázku je vlevo snímek obrazovky obsahující výpis **senzorů** a **aktorů** ve vybrané místnosti (Obývací pokoj). Každý z prvků seznamu tohoto výpisu obsahuje schematickou ikonu typu zařízení, jeho uživatelem definovaný název, čas poslední aktualizace hodnoty **senzoru** či **aktoru** a v neposlední řadě onu poslední naměřenou hodnotu. V případě **aktoru** se jedná o výpis jeho aktuálního stavu. Dále je na Obrázek 2.8 vpravo snímek detailu **senzoru**, kde jsou další informace o zařízení, jako například interval doby spánku senzoru, který lze měnit pomocí posuvníku na jednu z předdefinovaných hodnot ({1, 5, 10, 20, 30} sekund, {5, 15, 30} minut, {1, 2, 3, 4, 8, 12} hodin nebo jeden den), nebo graf předchozích naměřených hodnot. Hodnoty v grafu je možné posouvat, přibližovat a oddalovat.



Obrázek 2.8: Snímky klienta Android aplikace

Pro představu o odlišnosti grafického provedení zobrazuje obrázek 2.9 snímky klienta z prostředí Windows Phone 8.1. Nalevo je opět snímek výpisu **senzorů** v místnosti, který tentokrát neobsahuje čas aktualizace, ale pouze poslední hodnotu, název a ikonu. Každá s dlaždic představuje jedno z čidel. V případě, že **senzor** obsahuje čidel více (jedná se o multi senzor), je to uživateli naznačeno rozložením na menších dlaždicích.

Obrázek napravo ukazuje detail **senzoru**, tentokrát však překrytý dialogem pro přidávání nového **senzoru** - párovací režim (30 sekund pro zatřesení se senzorem). Samotná obrazovka obsahuje opět prvky pro akce, jako je nastavení doby aktualizace nebo přehled hodnot v grafu.

Obrázek 2.9: Snímky klienta Windows Phone aplikace

V obou případech jsou aplikace laděny do azurového odstínu, stejně jako web či logo systému, které je pro navrženo tak, aby tvarem připomínalo včelí plástev.

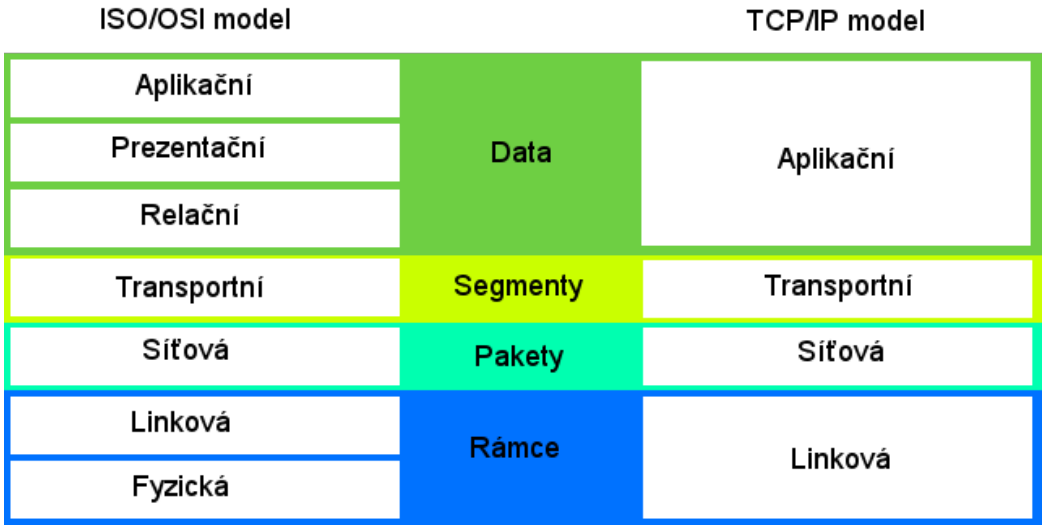
3 Analýza komunikačních modelů

V této kapitole popíši základy komunikace, která je nutná pro vzdálené ovládání inteligentní domácnosti a načrtnu principy potřebné k pochopení síťové vrstvy používané jak na straně klienta, tak i serveru. Nejprve bude představen TCP/IP model, který je standardem v dnešním internetu. Dále přiblížím klient-server model a peer-to-peer model, protože adaptér za určitých okolností lze považovat za klienta a při přímé komunikaci s ovládacím prvkem nepotřebuje nutně server. Pro komunikaci s inteligentní domácností je velmi důležitá bezpečnost a proto v neposlední řadě představím zabezpečený protokol SSL/TLS.

3.1 Model TCP/IP

V současnosti se na internetu a jiných počítačových komunikačních sítích nejvíce používají modely TCP/IP a ISO/OSI. Pro jejich pochopení následuje krátký avšak dostatečný popis pro účely této práce. TCP/IP model je složen z protokolu TCP (Transmission Control Protocol) a protokolu IP (Internet Protocol). Tento model, který je dnes standardem internetu vznikl jako experiment vojenské agentury DARPA (Defense Advanced Research Project Agency) v šedesátých letech dvacátého století za účelem vytvořit spolehlivou, robustní, decentralizovanou komunikační síť nezávislou na přenosovém médiu. Protože je složitost problému vysoká, je síťová komunikace rozdělena do několika vrstev. Výměna dat mezi jednotlivými vrstvami je přesně definovaná a každá z vrstev využívá služby té nižší [3].

Druhý používaný referenční model byl navržen standardizační organizací ISO (International Organization for Standardization) pod názvem OSI (Open System Interconnection). Tento model je považován spíše jako definice primární architektury počítačové sítě, která definuje funkce pro navázání, definování, používání a zrušení síťové komunikace mezi počítači. Stejně jako model TCP/IP definuje vrstvy, které logicky sdružují specifickou funkcionalitu.



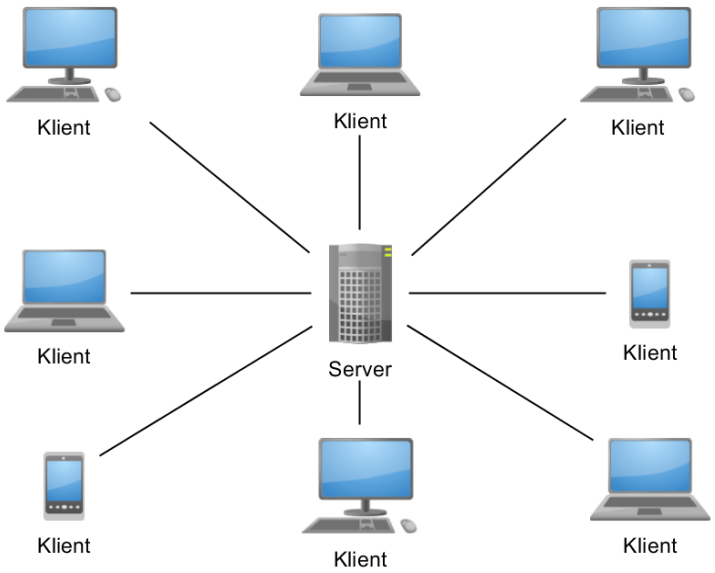
Obrázek 3.1: ISO/OSI - TCP/IP modely a porovnání

Oba zmiňované modely jsou si velmi podobné, každá vrstva modelu popisuje pravidla pro přenos informací na dané vrstvě. Na obrázku 3.1 je znatelné, že je TCP/IP model jednodušší, protože slučuje relační, prezentační a aplikační vrstvu ISO/OSI modelu do jedné aplikační vrstvy, stejně jako linkovou a fyzickou do jediné linkové vrstvy. Při průchodu dat shora dolů, každá vrstva přidává hlavičku k získaným datům a na nejnižší vrstvě se získaná data posílají po fyzickém médiu a u příjemce dochází k zpětnému rozbalování dat.

3.2 Model klient-server

Tato síťová architektura, jak název napovídá, definuje dva oddělené počítače komunikující spolu prostřednictvím počítačové sítě. Klient (často obsahující GUI) je prvek, nebo program, který žádá o služby jiného programu, serveru. Tento model je hojně na internetu využíván například pro přístup na E-mail, Web, atp. Každý klient může žádat o data jeden či více serverů a ty mohou přístup akceptovat a poskytnout zpět požadované informace. Této architektury využívají internetové protokoly, jako jsou HTTP (Hypertext Transfer Protocol), DNS (Domain Name System) a další [4].

Charakteristickou vlastností klienta je, že je aktivní, posílá žádosti a čeká na odpovědi od serveru. Obvykle je připojen k jednomu, či malému množství serverů. Na druhou stranu server hraje pasivní roli a naslouchá na síti a reaguje na žádosti připojených klientů. Na obrázku 3.2 je zobrazena architektura s jedním serverem a několika klienty.

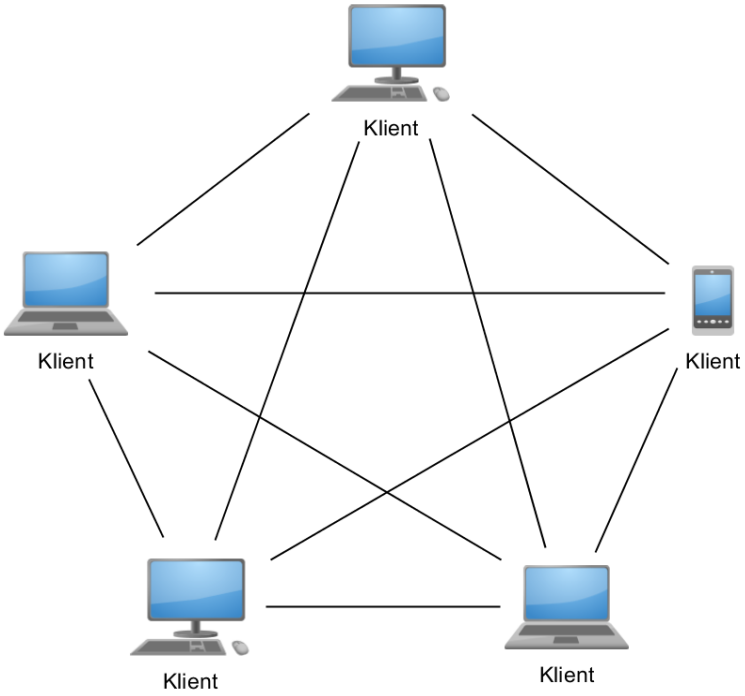


Obrázek 3.2: Architektura klient-server

Výhodou tohoto modelu jsou rozdělené jednotlivé úkony a zodpovědnost mezi klienta a server a tím snadnější údržba. Také zabezpečení serverů je mnohem vyšší a data na něm uložena jsou tak ve větším bezpečí. Navíc jsou data ukládána centrálně a jsou tak dostupnější a snadněji je lze aktualizovat, než například v peer-to-peer sítích. Mezi nevýhody lze zařadit problém s přetěžováním sítě v případě velkého množství souběžných požadavků od klientů. Oproti peer-to-peer sítím není tato architektura tak robustní proti výpadku. Pokud je server nedostupný, nejsou žádné požadavky klientů splněny.

3.3 Model Peer-to-Peer

Tento komunikační model označovaný někdy jako klient-klient (P2P) je typ sítě kdy jednotliví klienti komunikují mezi sebou bez potřeby centrálního serveru. Jak je ukázáno na obrázku 3.3, všechny uzly této architektury jsou si rovny. V praxi ovšem existují specializované servery, které pomáhají s navázáním komunikace [5].



Obrázek 3.3: Peer-to-peer model

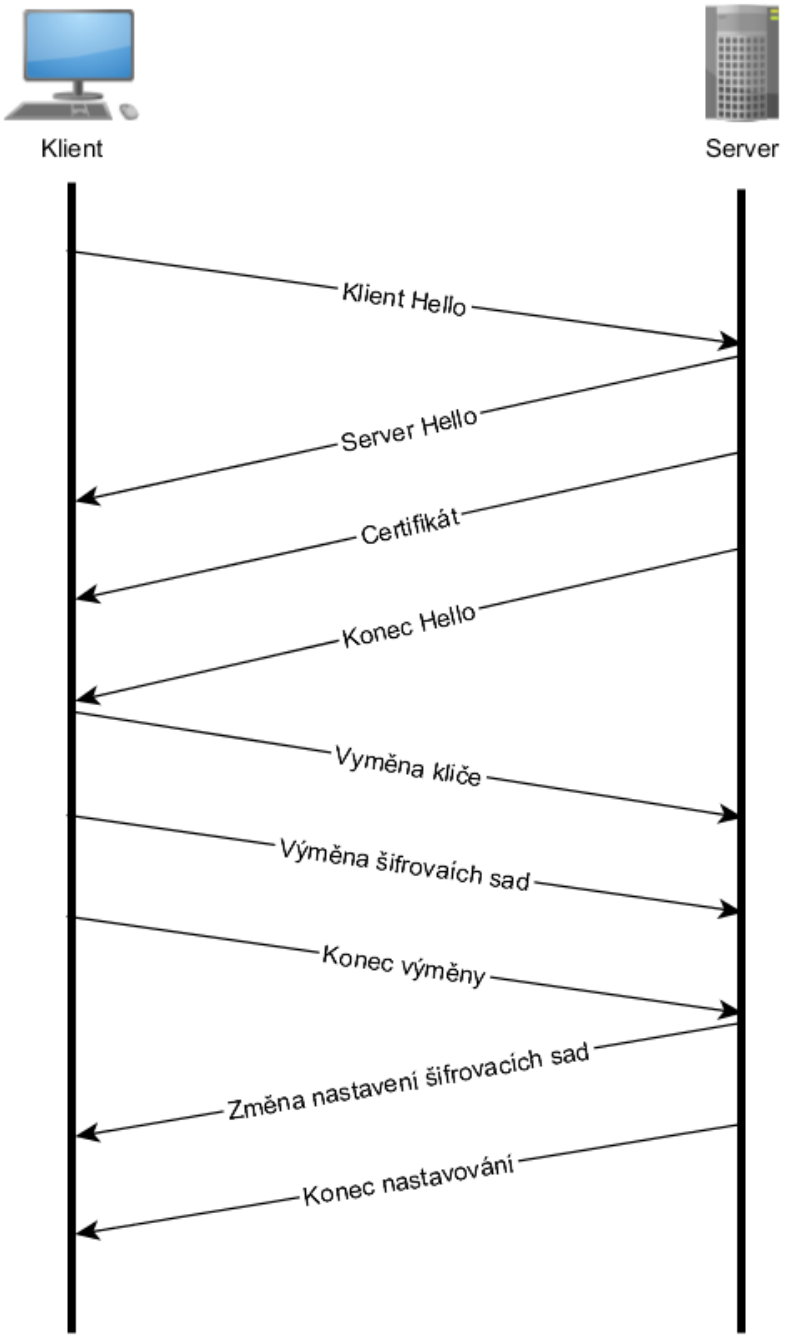
Výhodou těchto sítí je, že na rozdíl od modelu klient-server s rostoucím množstvím uživatelů dostupná kapacita roste. Mezi klasické zástupce využívající P2P patří například distribuční protokol BitTorrent, ale také open-source platební síť a kryptoměna Bitcoin, nebo velmi rozšířený program pro internetovou telefonii a instant messaging, Skype.

3.4 SSL/TLS

SSL (Secure Sockets Layer) a jeho následník TLS (Transport Layer Security) jsou kryptografické protokoly aplikační vrstvy TCP/IP modelu pro zajištění bezpečnosti na internetových sítích. V ISO/OSI architektuře je spojení zahájeno na relační vrstvě a prováděno na vrstvě prezentační. Pro ověření druhé strany se používá asymetrická kryptografie spolu s certifikáty X.509 a výměnou symetrických klíčů. Důležitou vlastností tohoto systému je, že ze zasílaného relačního klíče není možné odvodit privátní asymetrický klíč.

SSL 1.0 bylo vyvinuto firmou Netscape, ale kvůli bezpečnostním vadám nebylo vydáno. Na světlo světa se podívala v roce 1995 teprve verze 2.0, bohužel také obsahovala velké množství chyb a tak roku 1996 vyšla finální verze 3.0, která se později dočkala několika aktualizací. I po těchto aktualizacích je tento protokol zranitelný například proti POODLE útokům (Man in the middle).

Nejen proto v roce 1999 vyšel protokol TLS 1.0 jakožto pokračovatel SSL 3.0, který je dostatečně odlišný na to, aby byla komunikace s SSL 3.0 nemožná. Ovšem součástí TLS 1.0 je schopnost snížit, nebo spíše změnit protokol na úroveň SSL, což však přináší bezpečnostní rizika. Verze TLS 1.1 obsahovala několik úprav a oprav jako byla ochrana proti CBC (Cipher-block chaining) útokům nebo podpora pro IANA (Internet Assigned Number Authority) registrace. Poslední nasazená verze je 1.2, která například odstraňuje zmíněnou zpětnou kompatibilitu s SSL 2.0.



Obrázek 3.4: Základní handshake procedura protokolu TLS

Princip ověření a dohodnutí se na parametrech zabezpečené relace pomoci TLS, který je nazýván *handshake procedure* spočívá v několika krocích, jak je ukázáno na obrázku 3.4.

1. Procedura začíná, když se klient připojí na server podporující TLS a požádá o zabezpečené spojení. Také musí doložit seznam jím podporovaných šifrovacích sad (Cipher suite), což jsou sady obsahující algoritmus výměny klíčů, šifrovací algoritmus, kód pro ověření pravosti zprávy (obvykle MAC) a funkci pro generování pseudonáhodných čísel.
2. Server si ze zaslaného seznamu vybere sadu, kterou podporuje a oznámí to klientovi.
3. Dále server zasílá svou identifikaci ve formě digitálního certifikátu, který obsahuje jméno serveru, ověřenou certifikační autoritu (CA) a serverový veřejný klíč.
4. Klient ověří identitu certifikátu zaslaného serverem u nadřazené certifikační autority.
5. Následně klient vygeneruje relační klíč, což je pseudonáhodné číslo zašifrované veřejným klíčem poskytnutým serverem. Pouze server se svým privátním klíčem může toto číslo rozšifrovat.
6. Obě strany z náhodného čísla vytvoří nový klíč pro šifrování a dešifrování dat.

Obě strany teď mohou komunikovat s použitím šifrování a dešifrování s vytvořeným klíčem. Pokud jakýkoliv z kroků selže, je spojení ukončeno [6][7].

Bezpečnost protokolu kromě použití veřejných klíčů ověřovaných vůči certifikační autoritě pomocí digitálního podpisu dále zvyšuje, že klient kontroluje, zda je certifikační autorita (CA) na seznamu důvěryhodných autorit a také životnost serverového certifikátu. Proti útokům typu Man-in-the-Middle je porovnáváno DNS jméno serveru se jménem na certifikátu. TLS navíc obsahuje funkci, která pseudonáhodně rozděluje vstupní data na poloviny, které jsou každá zpracovávány jiným hashovacím algoritmem. Výsledek této akce je XORován, což poskytuje vyšší ochranu, pokud by byla nalezena slabina v jednom z algoritmů.

Většina knihoven podporujících SSL a TLS jsou open-source. Mezi pár vybraných patří například OpenSSL, JSSE (Java Secure Socket Extension), GNUTLS a další. K aplikacím a protokolům, které využívají toto zabezpečené připojení, patří protokoly aplikační vrstvy, jako jsou HTTP, FTP, SMTP a další. Také lze využít pro tunelování připojení, například pomocí síťových protokolů k vytvoření VPN (Virtual private network), jako je tomu u OpenVPN. Mezi aplikace používající zabezpečené připojení patří především webové prohlížeče. Poslední verzi TLS 1.2 podporují Google Chrome od verze 30, Mozilla Firefox od verze 24, kde však je tato možnost defaultně vypnuta (automaticky zapnuta ve verzi 27+). Stejný stav byl u Microsoft Internet Exploreru obsahující podporu již ve verzi 8, nativně však zapnutou teprve až od 11 verze. V případě Opery od verze 10, resp. 17 je součástí prohlížeče. Posledním z pětice hlavních prohlížečů je Apple Safari, který zavedl podporu od 7 verze.

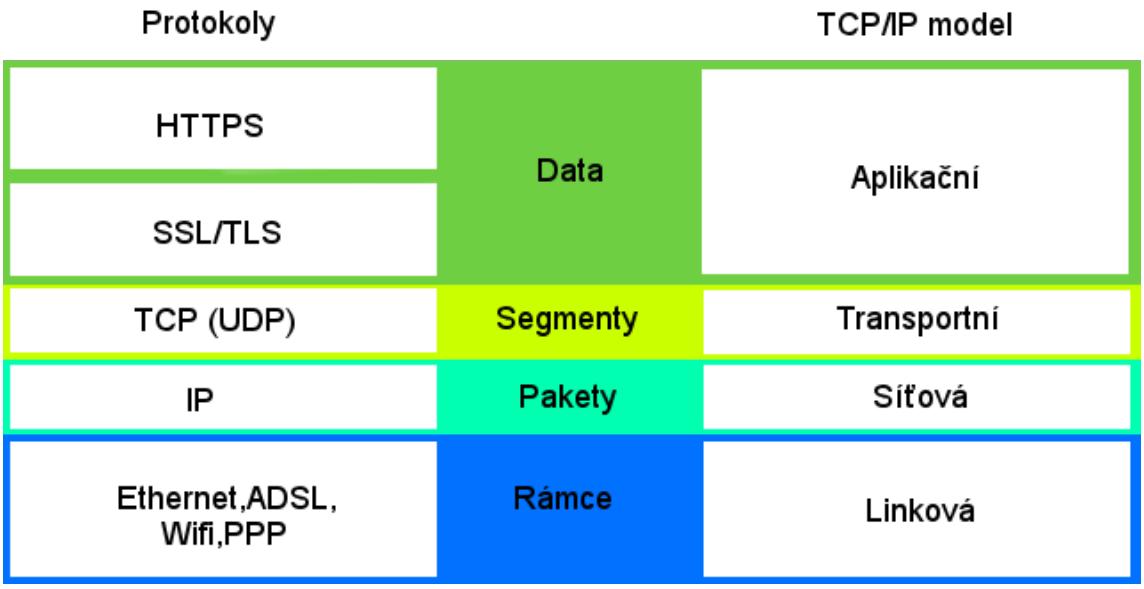
4 Protokoly aplikační vrstvy

Tato kapitola představuje tři protokoly, které lze pro účely komunikace obecně použít. Prvním je HTTPS, který na nižších vrstvách používá zmíněný SSL/TLS protokol. Nesmí chybět ani definice struktury posílaných zpráv a proto popíši i strukturu dvou nejpoužívanějších textových formátů vhodných pro zasílání dat, XML a JSON.

4.1 HTTPS

Jde o síťový protokol, který je zabezpečenou nadstavbou HTTP a chrání komunikaci mezi klientem a serverem (typicky webovým serverem) před odposloucháváním a podvržením dat či identity. Obrázek 4.1 zobrazuje postavení HTTPS (Hypertext Transfer Protocol Secure) protokolu v rámci výše zmíněného protokolu TCP/IP. HTTPS šifruje přenášená data pomocí SSL nebo novějšího TLS, který používá spolehlivého TCP. Nejčastěji používaný standardní port na straně serveru je 443.

Použití tohoto protokolu s sebou nese i některá omezení, mezi které patří například zpomalení. Nejedná se o výrazné zpomalení, je však znatelné (v řádu několika desítek milisekund). Dalším omezením, či spíše špatným použitím je šifrování statického webu. V dnešní době je velké množství stránek indexováno vyhledávacími roboty, díky čemuž se útočník může dostat k URI (Uniform Resource Identifier, např. adresa stránky) ze znalosti velikosti šifrované žádosti a odpovědi. To poskytuje informaci o šifrovaném i nešifrovaném obsahu stránky, což oslabuje použitý šifrovací algoritmus.



Obrázek 4.1: Postavení HTTPS a dalších protokolů v modelu TCP/IP

4.2 XML

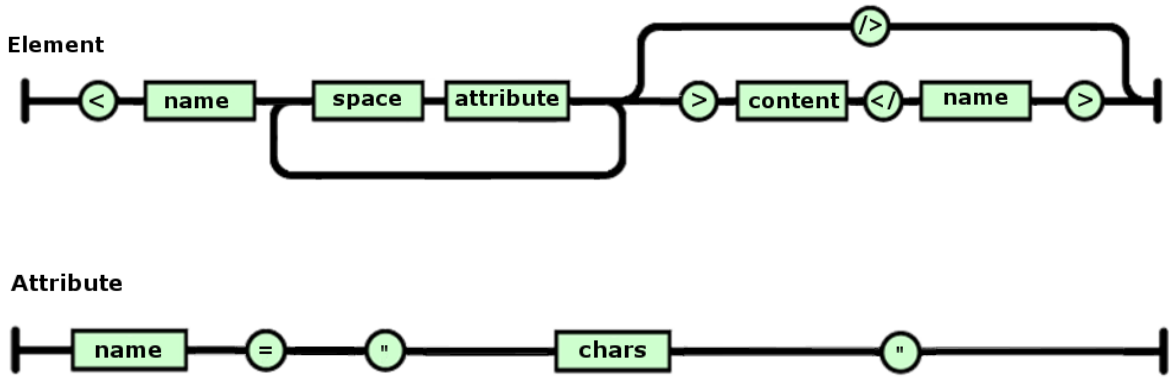
Jde o obecný značkovací jazyk, který zjednodušuje staršího předchůdce SGML (Standard Generalized Markup Language). XML (Extensible Markup Language) byl vyvinut a standardizován konsorciem W3C (World Wide Web Consortium) jako nástroj snadného vytváření značkovacích jazyků pro různé účely, jako je například serializace dat (převod objektu do sériové, tedy jednorozměrné podoby).

Samotné zpracování XML je podporováno ve většině programovacích jazyků. Ale existuje i spousta specializovaných knihoven (např. XmlPullParser pro Android). Kromě serializace je jazyk určen především pro publikování dokumentů, kde popisuje strukturu, ale nezabývá se výsledným vzhledem. Dále slouží pro přenos dat mezi aplikacemi, nebo je transformován do jiného typu dokumentu.

Syntaxe XML je vcelku jednoduchá. Dokument je vždy textový a vždy Unicode (nejčastěji UTF-8) a na začátku obsahuje hlavičku s verzí a kódováním. Dále dokument obsahuje sekci elementů, které mohou obsahovat atributy, případně v nich mohou být vnořené další elementy, nebo text. Validní dokument (tzv. well-formed) musí splnit podmínky, mezi které mimo jiné patří:

- Obsahovat právě jeden kořenový element (root)
- Neprázdné elementy musí obsahovat startovací i ukončovací značku
- Všechny atributy musí být uzavřeny do uvozovek
- Elementy mohou být vnořeny, ale nesmí se překrývat

Tuto syntaxi lze popsat pomocí diagramu, který je ukázán na obrázku 4.2. Každý element vždy začíná a končí lomenou závorkou. Identifikátorem elementu je jeho jméno, které nemusí být unikátní. Volitelně může element nést jeden či více atributů oddělené mezerou. Volitelný je i obsah elementu. V případě druhého diagramu (dole) popisujícího syntaxi atributu je struktura ještě jednodušší. Každý atribut je složen z unikátního jména v rámci zpracovávaného elementu a jeho hodnoty, která je uzavřena do závorek (lze použít i jednoduché uvozovky) a oddělena od jména rovnítkem. Jméno elementu ani atributu nesmí obsahovat, stejně jako hodnota elementu, znak menšítka, ampersandu a uvozovky použité pro uzavření hodnoty atributu.



Obrázek 4.2: Diagram syntaxe formátu XML

Nejčastější způsoby jak zpracovat XML dokument jsou pomocí DOM nebo SAX parseru (syntaktický analyzátor). DOM parser (Document Object Model) vytváří z dokumentu strom, nad kterým program pracuje. SAX parser (Simple API for XML) prochází postupně XML dokumentem a vyvolává události, které musí program sám ošetřit. V současné době je XML ve verzi 1.1 a od 1.0 se v podstatě liší pouze v tom, že 1.0 podporuje znaky pouze do Unicode 2.0, na rozdíl od 1.1, která podporuje i novější [8].

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Komentář, který není zpracováván. -->
<root atribut="hodnota" >
     <element>Text elementu</element>
     <element atribut_elementu="hodnota_elementu" />
     <element atribut="hodnota">Text</element>
</root>
```

Ukázka kódu 4.1: Syntaxe XML

Do základních vlastností tohoto formátu lze počítat vysoký informační obsah, což znamená, že obsahuje obecně více informačních znaků, než těch řídících. Další vlastností je mezinárodnost protokolu díky implicitnímu použití Unicode znakové sady. Užitečným rysem je jednoduchá konverze do jiných formátů, nebo snadné použití stylů zapsaný v jiném jazyce, například CSS (Cascading Style Sheets). Díky tomu, že tento jazyk nemá předdefinované žádné značky, lze jej tak doplnit o DTD (Document Type Definition) a na základě tohoto předpisu automaticky kontrolovat syntaktickou správnost nového dokumentu.

4.3 JSON

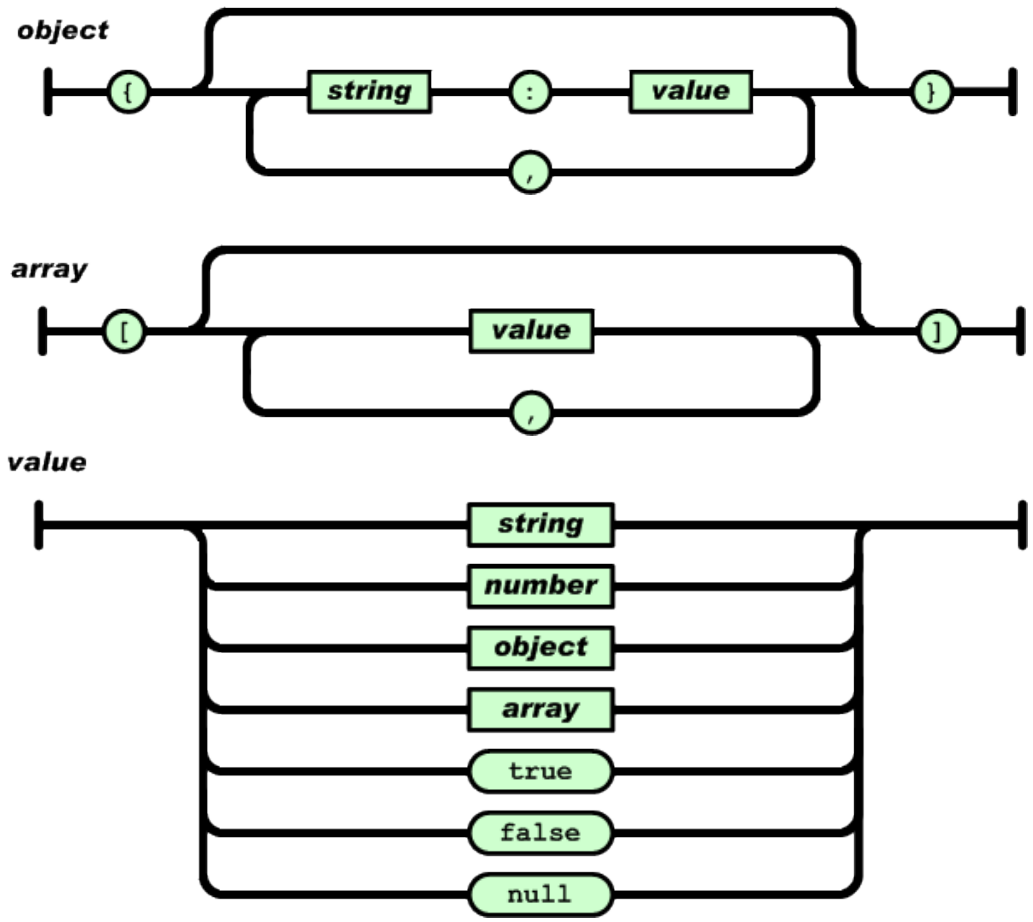
Další možností jak ukládat, přenášet či serializovat data je JSON (JavaScript Object Notation), který je stejně jako XML platformě nezávislý. JSON je optimalizovaný pro data, která lze seřadit jako pole, nebo objekty. Stejně jako XML nejčastěji používá UTF-8 kódování a má kromě JavaScriptu podporu i ve většině ostatních programovacích jazyků. Oproti XML je jednoduší a úspornější, ale v klasických implementacích nemá přímou podporu pro definování znakové sady a přenos binárních dat.

```
{
  "0"         : 2,
  "1"         : -1,
  "2"         : 5.333,
  "3"         : 6.0e+18,
  "4"         : "adc",
  "5"         : "\u00e2\n",
  "6"         : null,
  "7"         : [
                      2.5,
                      2.6,
                      [
                              "2.6.1"
                      ]
                ],
  "8"         : false,
  "9"         : true,
  "10"        : "",
  "key"       : "value",
  "abe\"def" : []
}
```

Ukázka kódu 4.2: Syntaxe JSON (mezery přidány pro zvýšení přehlednosti)

Struktura dokumentu v JSON formátu je vždy dána dvojicemi index a hodnota. Hodnota může reprezentovat řetězec, číslo a speciální typy jako jsou například booleovské hodnoty (true, false) nebo null. Jako hodnotu lze považovat i celé nové pole či objekt.

Tuto strukturu nejlépe popisují diagramy na obrázku 4.3. První z diagramů (nahoře) reprezentuje typ objekt, který je uzavřen do složených závorek a obsahuje zmíněné dvojice oddělené čárkou. Dvojtečka odděluje index od hodnoty. V druhém diagramu (uprostřed) je realizován popis pole, kde jsou hodnoty od sebe odděleny čárkou, a celé pole je ohraničeno hranatými závorkami. Poslední z diagramů reprezentuje hodnotu, která jak bylo řečeno výše, může obsahovat různé datové typy.



Obrázek 4.3: Diagramy syntaxe formátu JSON [17]

Rozšíření tohoto protokolu se událo především díky rozmachu JavaScriptu, resp. AJAXu (Asynchronous JavaScript and XML), který jej kromě XML používá pro serializaci dat při přenosu mezi klientem, typicky webovým prohlížečem, a serverem. Důvod k použití je především kvůli jeho minimální prostorové náročnosti. Dalším důvodem je také to, že je snadno čitelný člověkem, podobně jako je tomu u formátu XML. Mezi jeho nedostatky patří kromě nemožnosti definování znakové sady například absence komentářů [17].

5 Návrh komunikačního protokolu

Z teoretické části a informací z předchozích kapitol, je v této kapitole uveden návrh na základě principiálních požadavků řešení komunikačního protokolu mezi **serverem** a **koncovým ovládacím zařízením** (chytrý telefon). Výsledný návrh bere v úvahu nejen požadavky na komunikaci z hlediska klienta i serverové části, ale bere v úvahu i komfort uživatelů ve světle současných trendů.

5.1 Požadavky na komunikaci

Výsledné řešení by mělo splňovat následující požadavky na vlastnosti protokolu a implementace samotné komunikace:

- Nízká latence komunikace
- Bezpečnost komunikace
- Nízký objem přijímaných a zasílaných dat
- Spotřeba klienta
- Zátěž serveru
- Snadnost rozšíření a úprav - modularita
- Nezávislost na platformě klienta

Z pohledu uživatele pak mají nejvyšší prioritu první čtyři požadavky. Nízkou latencí je myšlena odezva klienta, resp. systému na požadavky zadané v GUI aplikace. V případě senzorů či získání dat z databáze, se jedná o součet časů zpracování v aplikaci, času komunikace a času zpracování požadavku na **serveru**. V případě přepnutí **aktoru** se pak jedná o odezvu celého systému, protože požadavek je ze **serveru** směrován přes **adaptér** na konkrétní **aktor**, který jej zpětně uvědomí o svém stavu - zda se akce provedla, či nikoliv.

Protože se jedná o komunikaci, na které mohou záviset kritické prvky domácnosti, je velice důležitá bezpečnost. Spojení mezi **klientem** a **serverem**, případně **adaptérem**, musí být šifrované a spolehlivé.

Nízký objem dat má kromě nepřímého vlivu na spotřebu především význam u uživatelů, kteří mají tarifní limity na množství přenesených dat. Proto je důležité, aby se zasílala jen potřebná data a aby maximum znaků v těchto zprávách bylo informačního rázu a ne řídících.

Jak bylo řečeno v předchozím odstavci, na spotřebu má vliv objem dat, dále pak doba, po kterou probíhá komunikace, a složitost zpracování přijatých informací. Se složitostí zpráv souvisí i zátěž **serveru**, který velkou měrou přispívá k latenci. V případě vysoké složitosti zpracování zpráv a při současně velkém vytížení mnoha klienty by byla odezva příliš dlouhá, proto je třeba hledat optimální řešení i z pohledu serverové části.

Z důvodu pozdější údržby a přidávání nových prvků do systému, je vhodné, aby byl protokol modulární, snadno pochopitelný a lehce čitelný. V neposlední řadě musí být protokol v 99% multiplatformní. To jedno zbývající procento odpovídá potenciálně různému typu ověřování klienta vůči serveru napříč různými platformami.

Protože neexistuje žádné uspokojivé open-source řešení, které by bylo dostatečně komplexní a robustní pro využití v komunikaci pro BeeeOn systém, je potřeba navrhnout vlastní, na míru postavené řešení.

Skoro celý dnešní Internet běží nad modelem TCP/IP (kapitola 3.1), proto na něm bude stavěno i toto řešení. Celý současný systém je navržen pro komunikaci skrze server, z toho důvodu bude brán v potaz komunikační model klient-server (kapitola 3.2) a v budoucnu v případě přímé komunikace klienta s adaptérem bude upraven na hybridní model s prvky P2P sítě (kapitola 3.3).

Pro zajištění bezpečnosti komunikace se nabízí HTTPS, ale protože je žádoucí vlastní řízení toku přenášených dat, bude použito SSL/TLS (kapitola 3.4) zabezpečení a formát zpráv bude definován vlastním protokolem. Pro protokol se nabízí několik variant. První rozhodnutí tkví ve volbě mezi textovým a binárním formátem zpráv. Použití binárního protokolu přináší výhodu ve formě menšího množství dat, na druhou stranu textový formát může být snadno čitelný člověkem, tedy správcem systému. V případě volby textové varianty je třeba se rozhodnout, zda použít nějaký standardizovaný formát, či vyvinout vlastní syntaxi.

Zde především kvůli údržbě a obecnosti vyvstávají dvě možnosti - XML nebo JSON. Mezi těmito dvěma jak bylo řečeno v podkapitole 4.3 má JSON výhodu v poměru množství řídících symbolů k informačním. Na druhou stranu i XML (kapitola 4.2) lze napsat stylem velmi podobným jako je JSON a rozdíly jsou pak vcelku zanedbatelné.

K rozhodnutí pro zvolení XML na úkor binárního či JSON formátu přispívá serverová část, protože používá typ databáze, která má přímou podporu pro XML a tudíž je jeho zpracování optimální. Spolu se snadnou modularitou, údržbou a čitelností je XML dobrá volba.

5.2 Návrh řešení

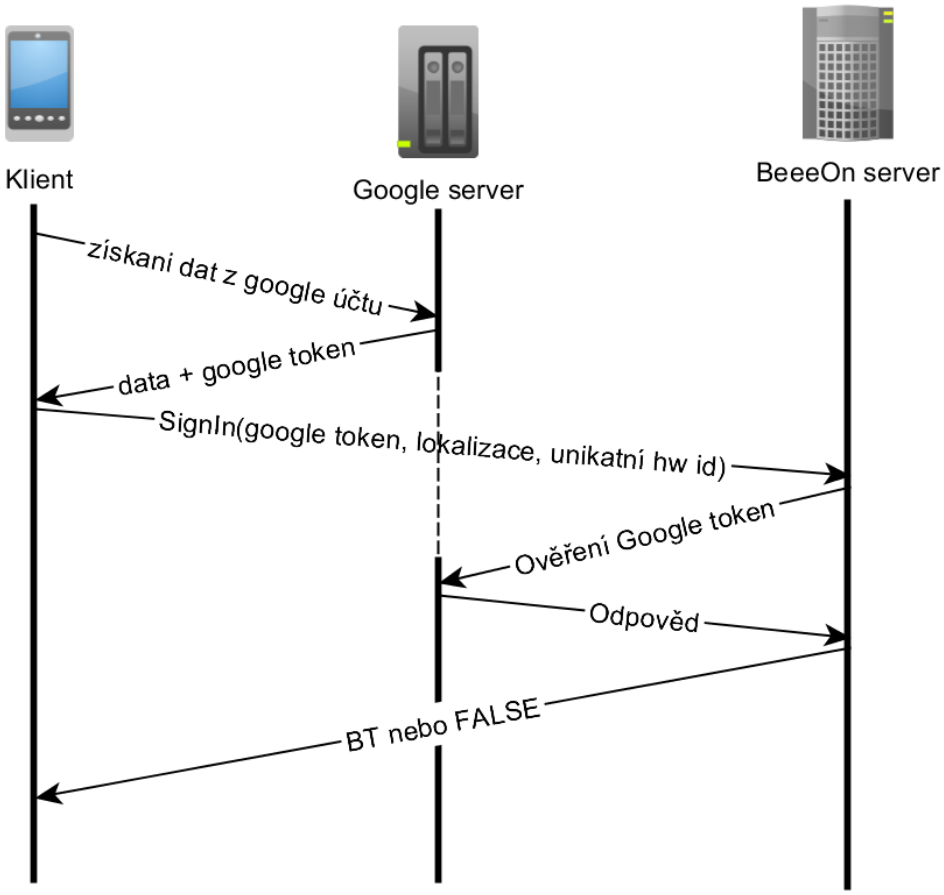
Kromě požadavků na použité technologie a formáty, jsou zde i nároky na obsah a sémantiku komunikačních zpráv. Tyto zprávy lze rozdělit podle různých kritérií na několik typů. Například rozdělení podle typu objektu, se kterým je manipulováno, na základních deset variant:

- **přihlašovací, registrační**
  registrace uživatele do systému, přihlášení na server
- **potvrzovací**
  reakce serveru na zaslaný požadavek (vše v pořádku / chyba)
- **operace s adaptérem**
  získání přístupu k datům adaptéru, řízení funkcí adaptéru
- **operace s koncovými zařízeními**
  získávání dat senzorů, ovládání aktorů
- **manipulace s místnostmi**
  vytváření, změna či mazání místností (více o místnostech v kapitole 5.2.5)
- **operace s pohledy (grafy vlhkosti)**
  vytváření, úprava nebo změna pohledu
- **správa uživatelských účtů**
  přidávání nových uživatelů, nastavování přístupových práv
- **manipulace s časem a lokalizací**
  nastavení časové zóny adaptéru a jazyka koncového zařízení
- **úpravy podmínek a akcí - algoritmy**
  pokročilá funkcionalita pro řízení uživatelem definovaných podmínek a akcí
- **notifikace a geofencing**
  oznámení o stavu domácnosti, splnění podmínky, vykonání definované akce

Jednoduše by se zprávy daly rozdělit i na řídící a manipulující s daty (get — set), nebo na zprávy od klienta a zprávy od serveru. Tato práce se však bude dále držet prvního ze zmiňovaných rozdělení a to z důvodu toho, že ne všechny z kategorií jsou existenčně důležité a jsou tedy spíše rozšířením základního konceptu potřebného k ovládání inteligentní domácnosti systému BeeeOn.

5.2.1 Přihlášení, registrace

V případě nového uživatele, je nejprve nutné zaregistrovat jej na serveru a získat přístup k účtu. Tato akce je prováděna zprávou **SignUp**. Pro základní autentifikaci a verifikaci uživatele, tedy přihlášení jsem navrhl zprávu **SignIn.** Jakmile je jednou uživatel registrován, může se přihlásit pomocí této zprávy a získat povolení pro další komunikaci se serverem. Účelem této zprávy je zaslat na server důležité informace o uživateli a jeho zařízení, na kterém je klient nainstalován. Pokud je vše v pořádku odpoví server zprávou obsahující BT (BeeeOn Token - unikátní identifikátor, který je serverem přiřazen pro každou relaci) jak je ukázáno na obrázku 5.1, kde je jako metoda přihlášení (poskytovatel přihlášení) na klientovi zvolen google účet.



Obrázek 5.1: Přihlášení do systému

V této sekci se nachází následující zprávy:

- **SignUp** - registrace uživatele do systému
- **SignIn** - přihlášení uživatele do systému
- **BT** - odpověď obsahující BeeeOn Token
- **GetUserInfo** - dotaz na informace o uživateli obsažené na serveru
- **UserInfo** - odpověď obsahující informace o uživateli
- **Logout** - odhlášení ze systému
- **JoinAccount** - přidání nového poskytovatele přihlašování
- **CutAccount** - odstranění některého z poskytovatelů přihlášení

Zpráva **SignUp** na server zasílá jen nezbytná data pro registraci, která se mohou různit v závislosti na vybraném poskytovateli přihlášení. Mezi poskytovatele přihlášení patří BeeeOn, tedy proprietární systém, který si žádá unikátní jméno a heslo. Dále pak Google, který pro autentizaci používá Google Token (klíč pro ověření identity vůči Google účtu). K dalším poskytovatelům se řadí Facebook a MojeID.

Přihlášení pomocí **SingIn** zprávy, je obsahově velmi podobné výše zmíněné registrační zprávě, navíc však obsahuje HW ID (jedinečný identifikátor zařízení - telefon, tablet) a lokalizace uživatele. Pokud **server** vyhoví požadavku, je zaslána zpráva **BT**, která nese požadovaný BT a ten bude klient dále používat jako relační identifikátor, dokud se uživatel neodhlásí, nebo nevyprší jeho platnost na serveru. Pokud nastane chyba, je zaslána zpráva **FALSE**.

V případě, že uživatel použije k registraci poskytovatele jiného než BeeeOn, stahuje server automaticky od tohoto poskytovatele veškeré relevantní dostupné informace. Tato data se zpětně dají získat pro potřeby klienta (celé jméno, pohlaví, profilový obrázek) díky zprávě **GetUserInfo** a odpovědi na tuto zprávu, **UserInfo**, která obsahuje navíc kromě standardních dat unikátní perzistentní identifikátor uživatele, nebo například seznam poskytovatelů přihlášení, ke kterým byly dodány přihlašovací údaje. Tento seznam lze upravovat pomocí zpráv **JoinAccount** (přidání dalšího poskytovatele) a **CutAccount** (odebrání stávajícího poskytovatele).

Nesmí chybět ani zpráva pro ruční odhlášení uživatele od systému, **Logout**. Bližší informace včetně ukázek zpráv a popisu jejich struktury jsou představeny v příloze v sekci **1: Ukázky komunikačních zpráv**.

5.2.2 Potvrzovací zprávy

Potvrzovací zprávy jsou pouze dvě a to **TRUE**, která je zasílána jako kladná odpověď a zpráva **FALSE**, která signalizuje nějakou chybu. Obě zprávy jsou zasílány ze strany serveru, aby klient věděl, zda požadavek uspěl či nikoliv.

Zpráva **TRUE** je velice jednoduchá a krátká, aby se snížil datový tok. Jedná se totiž o jednu z nejpoužívanějších zpráv při manipulaci s domácností. Její přínos je zejména v případech s pomalým připojením, kdy nemusí být jasné, zda došlo k chybě či zpráva s chybovým stavem zatím nedorazila.

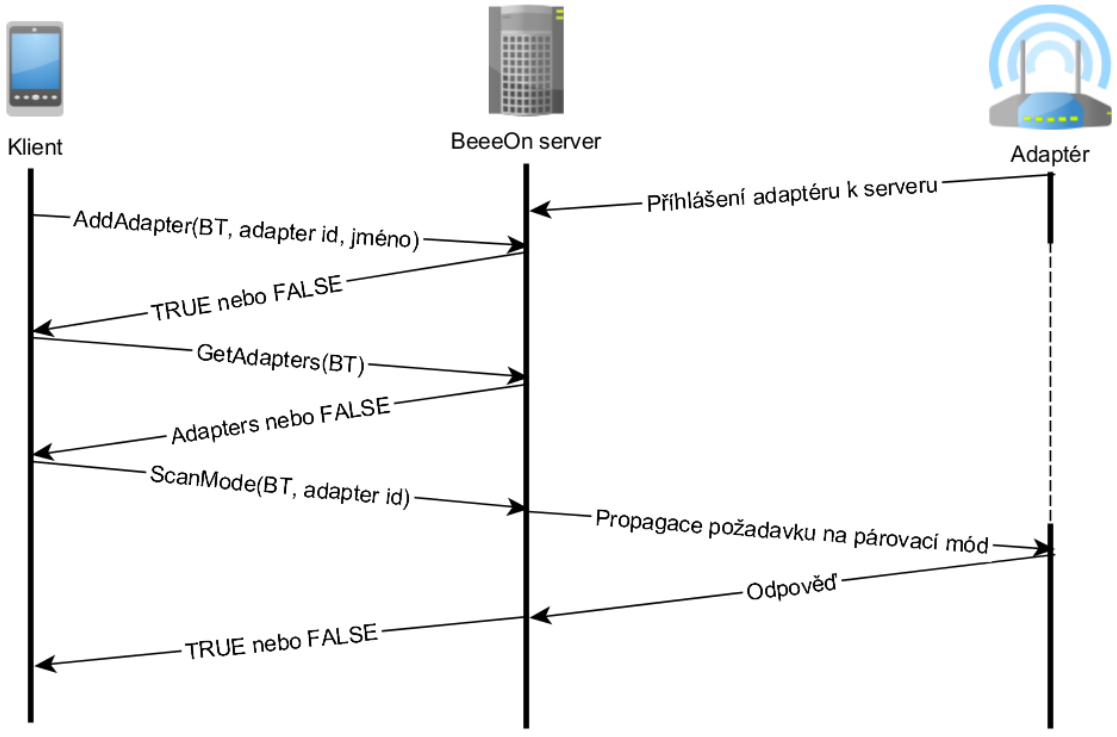
Naproti tomu zpráva **FALSE** je o něco složitější, obsahuje navíc atribut *errcode* (error code), který nese číselný kód chyby, která během komunikace nebo zpracování nastala. Pokud se jedná o vážnější, nebo složitější chybu, lze poslat i textový řetězec, který jí blíže popíše, nebo dá instrukce uživateli, co by měl podniknout za akci.

5.2.3 Operace s adaptérem

Po získání BT, který je pro další komunikaci se serverem nezbytný, musí klient nahrát data **adaptéru**, resp. zaregistrovat si **adaptér**, pokud tak zatím neudělal. Další důležitou zprávou je iniciování párovacího režimu, aby se mohla jednotlivá koncová zařízení (**senzory** a **aktory**) spojit s daným **adaptérem**. Do této kategorie se řadí zprávy:

- **AddAdapter** - přidání adaptéru
- **DelAdapter** - odstranění adaptéru
- **GetAdapters** - požadavek na seznam adaptérů
- **Adapters** - odpověď se seznamem adaptérů
- **ScanMode** - požadavek na přepnutí adaptéru do párovacího režimu
- **ReInitAdapter** - zpráva pro výměnu adaptéru
- **GetGateInfo** - získání informací o adaptéru
- **GateInfo** - požadované informace o adaptéru
- **SetGate** - nastavení informací o adaptéru

Obrázek 5.2 ukazuje komunikaci mezi klientem a **serverem** při manipulaci s **adaptérem**. V prvním kroku musí být **adaptér** zapojen do sítě a dát o sobě **serveru** vědět. Následně je možné z klienta poslat zprávu **AddAdapter**, která se pokusí přihlášenému uživateli zaregistrovat daný **adaptér**. Pokud **server** odpoví kladně, je zaslána zpráva **GetAdapters**, na kterou **server** odpoví seznamem všech **adaptérů** uživatele. Poté je již **adaptér** v aplikaci klienta aktivní a lze například přepnout **adaptér** do párovacího režimu pomocí zprávy **ScanMode**. **Server** tuto zprávu přeposílá na **adaptér**, který na ní reaguje a posílá zpět odpověď, kterou **server** zpracuje a výsledek přepošle na klientskou aplikaci.



Obrázek 5.2: Diagram komunikace při manipulaci s adaptérem

Jak bylo zmíněno výše, zpráva **AddAdapter** je zasílána z ovládacího zařízení směrem na **server** a registruje **adaptér** pro aktuálního uživatele. Tento uživatel se stává vlastníkem adaptéru a má nejvyšší roli *superuser* (více v kapitole 2.2.4).

V případě **ReInitAdapter** se nejedná o běžně používanou zprávu, ale pouze o řešení případné chyby. Pokud by uživatel z nějakého důvodu musel vyměnit nefunkční **adaptér**, ale přitom by nechtěl přijít o všechna data a **senzory**, použije se tato zpráva, která starý **adaptér** nahradí za nový.

Další zprávy jako je **GetGateInfo, GateInfo** a **SetGate**, pouze získávají, či nastavují informace o Adaptéru (brána), které nejsou pro běh aplikace z pohledu uživatele kritické, jako je například počet připojených zařízení, počet uživatelů či IP adresa brány.

5.2.4 Operace s koncovými zařízeními

Jednou z nejdůležitějších kategorií jsou samozřejmě zprávy nesoucí informace o samotných **senzorech** a **aktorech**. Do této kategorie se řadí:

- **GetAllDevs** - získání všech zařízení daného adaptéru
- **GetNewDevs** - získání všech neinicializovaných zařízení daného adaptéru
- **GetDevs** - požadavek na specifická zařízení
- **GetLog** - požadavek na historická naměřená data
- **AllDevs** - odpověď se všemi zařízeními
- **Devs** - odpověď s neinicializovanými nebo specificky volenými zařízeními
- **LogData** - požadovaná uložená data
- **SetDevs** - nastavení senzorů
- **Switch** - přepnutí aktoru
- **DelDev** - odstranění zařízení ze serveru

I přestože, název zprávy **GetDevs** je velice podobný dvěma předcházejícím, syntaxe i sémantika je značně odlišná. Zpráva si žádá data konkrétních koncových prvků, tentokrát navíc napříč různými **adaptéry**. Odpovědí je **Devs** stejně jako v případě požadavků **GetAllDevs** a **GetNewDevs**, je-li požadavku vyhověno, v opačném případě opět **FALSE**.

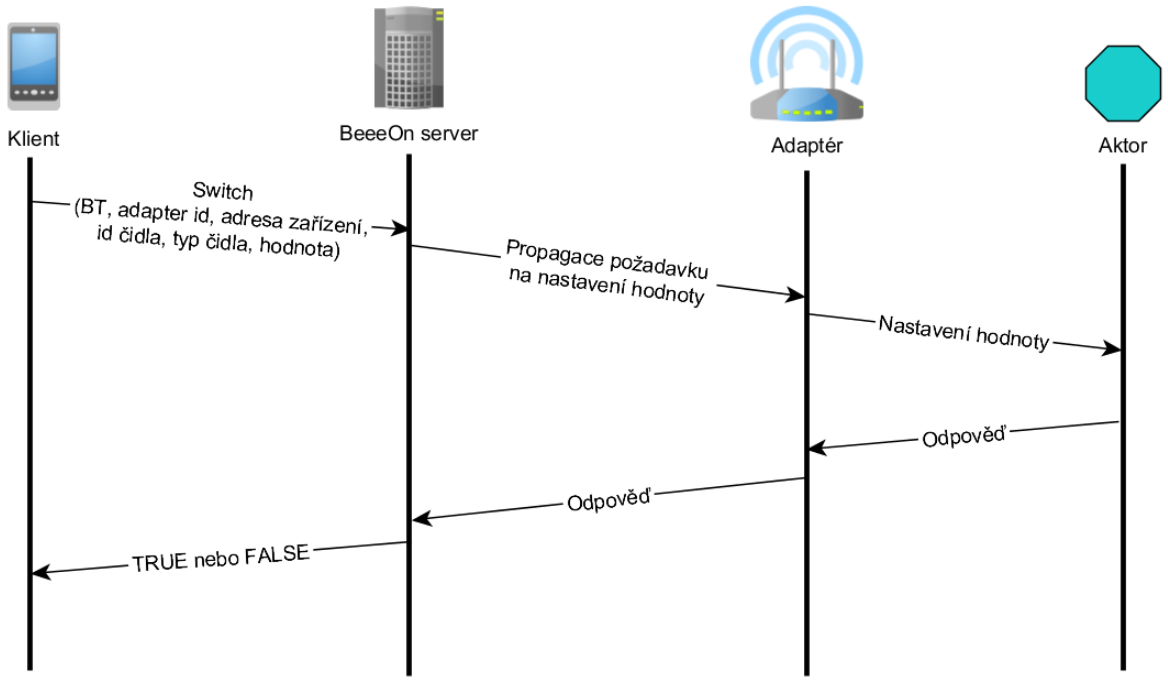
Další zprávou z klienta žádající si data je **GetLog**. Jak název napovídá, požadovaná data jsou logy, resp. uložené historické hodnoty. Zpráva obsahuje důležité informace, jako jsou:

- počáteční a koncový čas požadovaných dat
- typ agregační funkce (průměr, suma, atp.)
- časový interval, nad kterým je agregační funkce vykonávána
- jedinečnou identifikaci senzoru nebo aktoru

Očekávanou odpovědí je zpráva **LogData**. Jak je ukázáno ve zprávě 20 v příloze, zpráva obsahuje pouze seznam elementů *row* (řádek). Každý z těchto řádků obsahuje právě jednu časovou značku a jednu hodnotu. Protože je však velmi pravděpodobné, že se bezprostředně za sebou velmi často objevují stejné hodnoty, navrhnul jsem optimalizaci spočívající v seskupení stejných hodnot do jednoho řádku. Takto optimalizovaný řádek tak obsahuje počáteční časovou značku a hodnotu, která je pro všechny záznamy stejná. Navíc je zde atribut *repeat*, který definuje počet opakování této hodnoty a atribut *interval*, který nese informaci o tom, jak jsou hodnoty od sebe časově vzdálené.

Informaci o vzdálenosti hodnot sice klient má, protože si sám definoval, ve zprávě **GetLog**, jak velké rozpětí chce, ale toto udržení kontextu může být pro nižší vrstvy komunikace složité, proto je tato informace součástí zprávy. Vzhledem k tomu, že se jedná o zprávu zasílanou ze **serveru**, je to právě on, kdo má v rukou optimálnost této zprávy, resp. kdy se už vyplatí použít optimalizaci a kdy ne.

Poslední zprávou, která manipuluje přímo se záznamy **senzorů** nebo **aktorů** je **DelDev**, která odstraní koncový prvek z **adaptéru**. Tento prvek se tak opět stává neinicializovaným a nespárovaným, jako po prvním spuštění.



Obrázek 5.3: Diagram přepnutí stavu aktoru (nastavení hodnoty)

Pro případ potřeby přepnout či nastavit hodnotu **aktoru** jsem navrhl zprávu **Switch**, která ač je minimalistická (kromě identifikace aktoru nese jen požadovanou hodnotu) trvá její vyřízení z pohledu koncového klienta relativně dlouho oproti například smazání **aktoru**. Důvod je zcela prostý a je načrtnut na obrázku 5.3. Především z důvodu bezpečnosti je zapotřebí aby byla odpověď na požadavek na přepnutí **aktoru** vždy validní. Například při akci zavření okna musí být okno skutečně zavřené dříve, než je stav přepnut i v mobilním zařízení. To znamená, že nelze pouze předat požadavek na server, ale je nutno čekat až se požadavek a jeho odpověď protlačí skrze celý systém.

5.2.5 Manipulace s místnostmi

Aby bylo snazší identifikovat jednotlivá koncová zařízení, byl zvolen model dělící tyto **senzory** a **aktory** do virtuálních místností. Uživatel si může tyto místnosti vytvářet, upravovat a rušit podle sebe. Mezi tyto zprávy patří:

- **AddRoom** - přidání místnosti
- **SetRooms** - přejmenování místnosti
- **DelRoom** - odstranění místnosti
- **GetRooms** - požadavek na seznam místností

- **Rooms** - odpověď se seznamem místností
- **RoomID** - odpověď s ID nové místnosti

První zprávou je **AddRoom**, která vytváří novou místnost. Definuje tak její název atributem *name* a typ atributem *ltype* (location type), čehož se využívá především v aplikaci pro výběr odpovídajícího zobrazení (ikonky). Tabulka 5.1 ukazuje základní předdefinované místnosti, ke kterým jsou dané ikony. **Server** na tuto zprávu reaguje zprávou **RoomID**, pokud se podařilo přidat tuto místnost. Ta vrací klientovi nově na **serveru** vygenerované id. Jde o extrémně jednoduchou zprávu obsahující pouze *ver*, *state* a *lid* (location ID).

typ	název
1	Koupelna
2	Ložnice
3	Zahrada
4	Kuchyně
5	Obývací pokoj
6	WC
7	Kancelář
8	Knihovna

Tabulka 5.1: Předdefinované typy místností

5.2.6 Operace s pohledy

Za účelem větší uživatelské přívětivosti je součástí aplikace možnost přepnout na specifické pohledy na inteligentní domácnost. V současnosti je podporován speciální grafový pohled na vlhkost domácnosti. Další návrh zpráv, které by tuto funkcionalitu podporovaly, závisí především na požadavcích vyšších vrstev, proto bude tato problematika řešena později. V této fázi lze představit výběr z několika zpráv pro pohled na vlhkost (ostatní v příloze):

- **GetHumOverview** - požadavek na přehled vlhkosti
- **HumOverview** - přehled vlhkosti domácnosti
- **GetHumMsgDetail** - požadavek na detailní popis události
- **HumMsgDetail** - zpráva s detailem vzniklé události
- **GetHumLocDetail** - požadavek na přesná data vlhkosti v dané místnosti
- **HumLocDetail** - detail vlhkosti požadované místnosti
- **GetHumLocStats** - požadavek na získání statistik vlhkosti v místnosti
- **SetHumNotifs** - nastavení notifikace pro vlhkost
- **SetHumThreshold** - nastavení prahu pro sledování vlhkosti

5.2.7 Správa uživatelských účtů

Další důležitou funkcionalitou je řízení uživatelských účtů pomocí definovaných rolí. Tuto možnost mají pouze uživatelé s rolí *superuser*. Tito uživatelé mohou přidávat, měnit či mazat uživatelské účty pomocí těchto zpráv:

- **AddAccs** - přidání nového uživatele
- **SetAccs** - úprava práv uživatele
- **DelAccs** - odstranění uživatele z adaptéru
- **GetAccs** - požadavek na seznam všech uživatelů adaptéru
- **Accounts** - odpověď se seznamem všech uživatelů

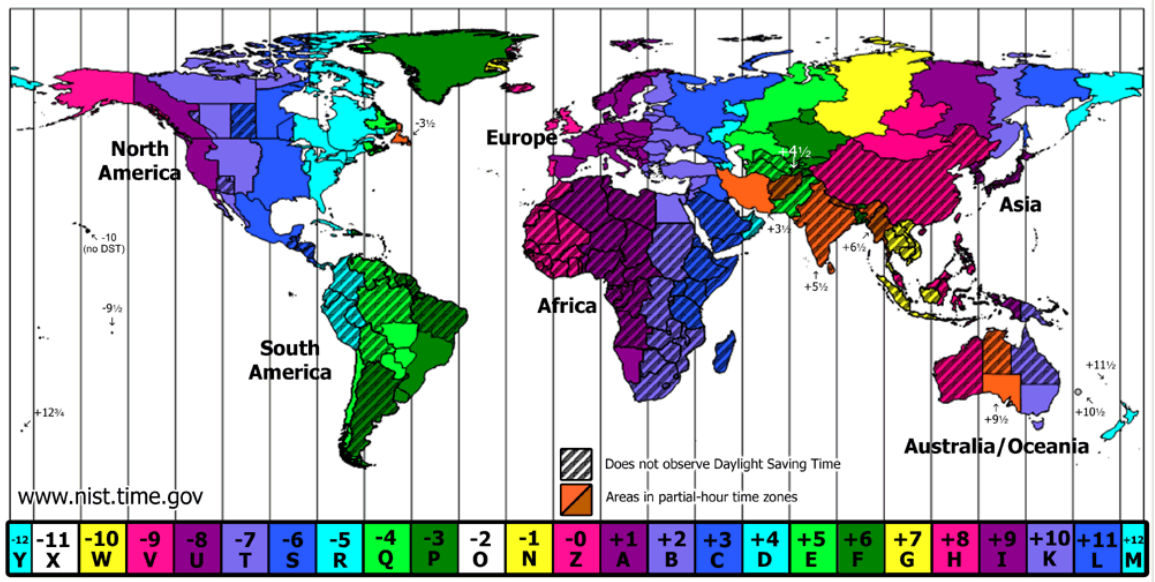
Důležitá zpráva je především **AddAccs** přidávající nové uživatele k danému **adaptéru**. Obsahem zprávy je seznam uživatelů, kteří jsou definování emailem (atribut *email*) a rolí (atribut *role*). Server na tuto akci reaguje zprávou **TRUE** nebo **FALSE**.

Druhou stěžejní zprávou je **Accounts.** Jde o zprávu zaslanou ze **serveru**, která obsahuje informace o uživatelích daného adaptéru. Obsahem je list elementů *user*, které nesou email a roli, ale také jméno (*name*), příjmení (*surname*) a pohlaví (*gender*), které **server** získá od třetích stran na základě emailu, pokud je to možné (např. z Google účtu).

5.2.8 Manipulace s časem a lokalizací

Následující zprávy nejsou příliš frekventované, ovšem mají smysl, především pokud uživatelé často cestují, nebo se zrovna plánují přestěhovat, případně mají chuť trénovat cizí řeči. Tyto zprávy byly navrženy tak aby bylo možné použít libovolné časové pásmo. Diagram se všemi časovými zónami je vyobrazen na obrázku 5.4. Následuje jejich výčet:

- **SetTimeZone** - nastavení časové zóny
- **GetTimeZone** - požadavek na získání časové zóny
- **TimeZone** - odpověď s časovým pásmem
- **SetLocale** - nastavení jazyka klienta



Obrázek 5.4: Časová pásma [16]

5.2.9 Úpravy podmínek a akcí - algoritmy

Stejně jako v případě pohledů je i použití podmínek a akcí závislé na dalších částech systému a proto zde uvedu jen předběžný seznam předpokládaných zpráv, které by měly pokrýt potřeby přidávání, manipulace a měnění uživatelských podmínek a akcí.

Podmínky, které jsou zde zmíněny, jsou nadmnožinou používaných zpráv, tzn. hlídače, který používá definované rozhraní pro algoritmy běžící na serveru, které se starají o správné vyhodnocení uživatelem definovaných událostí.

Tyto uživatelem definované podmínky mohou být velmi jednoduché, založené na výskytu jediné události. Současný stav implementovaných algoritmů obsahuje tři hlavní typy. *Watch_and_notify* je prvním z nich, který hlídá hodnotu senzoru, či aktoru pomocí funkcí. Funkce, které lze pro definování podmínek použít, jsou matematické (=, >, <, <=, >=). Pokud je podmínka splněna, je dle požadavků uživatele buď nastavena hodnota některého z aktorů nebo je aktivována notifikace, kterou si uživatel definoval. Dalším typem je *Quality_of_life*, který na základě hodnoty senzoru a typu místnosti, monitoruje a oznamuje aktuální stav kvality prostředí domácnosti. Posledním typem je *Geofencing*, který je použit pro monitorování polohy uživatele vůči definovaným regionům. Vstupem je definovaná oblast a zvolený typ interakce s touto oblastí (vstup nebo výstup). Pokud je tato událost zaznamenána, např. příchod člena rodiny domů, je přepnut aktor nebo je vystavena notifikace. Tato geolokační problematika je přidružena k notifikacím v další podkapitole.
Výběr několika typů zpráv (kompletní seznam je umístěn v příloze):

- **AddAlg** - přidání algoritmu
- **AlgCreated** - algoritmus úspěšně přidán
- **GetAllAlgs** - požadavek na všechny algoritmy
- **Algs** - seznam všech definovaných algoritmů (pravidel)
- **DelAlg** - smazání algoritmu

5.2.10 Notifikace a geofencing

A v neposlední řadě je zde sekce zabývající se notifikacemi. Základní zasílání notifikací je řešeno mimo tento komunikační protokol prostřednictvím specializované služby Googlu (případně jiných služeb, pokud se nejedná o koncové zařízení s operačním systémem Android). Zprávy zde definované jsou tak spíše řídící nastavení potřebného stavu či umožňující uživateli zobrazit i staré notifikace.

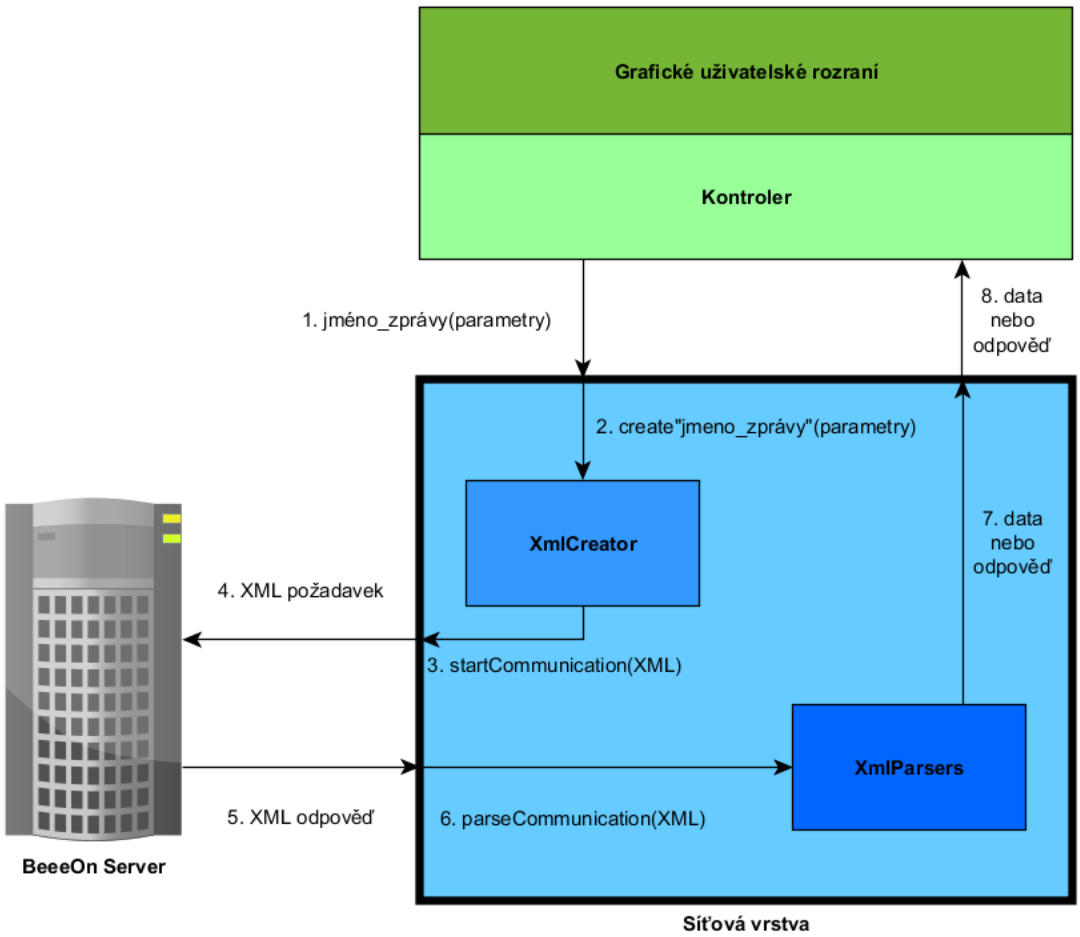
Ze stejného důvodu jako je tomu v případě pohledu, podmínek a akcí, představím v této podkapitole pouze výčet potřebných zpráv.

- **DelGCMID** - smazání GCMID (Google Cloud Messaging ID)
- **SetGCMID** - nastavení GCMID
- **GetNotifs** - dotaz na seznam notifikací
- **Notifs** - seznam notifikací
- **NotifRead** - označení notifikace za přečtenou
- **PassBorder** - geolokační zpráva oznamující serveru změnu polohy uživatele

6 Implementace

Tato kapitola předkládá postup vývoje a implementační detaily praktické části práce, která se zabývala návrhem textového komunikačního protokolu založeného na XML notaci, vývojem a optimalizací modulů pro síťovou komunikaci a zpracování navrženého protokolu na straně klientské aplikace.

Jak bylo zmíněno v kapitole 2.3, aplikace je rozdělena na několik modulů, které komunikují přes daná rozhraní. V rámci práce jsem implementoval síťovou vrstvu spolu s procesorem na zpracování XML pro platformu Android v programovacím jazyce Java (pro vývoj použito Android Studio, dříve Eclipse). Obě vývojová prostředí využívají Android SDK obsahující ovladače a další specifika pro potřeby programování pro tuto platformu (v případě Eclipse jde o plugin) a také disponují emulátorem, který je schopen emulovat celou paletu verzí systému a spoustu modelů mobilních zařízení (pro Eclipse opět ve formě pluginu). Pro vývoj síťové vrstvy bylo však téměř vždy použito reálné zařízení a to především kvůli obtížím s použitím Google účtu (autentizace) v prostředí emulátoru v raných fázích vývoje.



Obrázek 6.1: Propojení modulů a komunikace mezi nimi

Obrázek 6.1 zobrazuje, jak jsou jednotlivé moduly propojeny mezi sebou, kontrolérem a serverem. Požadavek vyšších vrstev na získání informací od serveru začíná voláním (1) kontroleru některou

z metod třídy **Network**, které jsou předány parametry (2) do modulu pro sestavení XML požadavku. Takto získaný požadavek je zaslán (3)(4) na serverovou část, kde je vyhodnocen a je zaslána (5) odpověď. Zpracování (6) odpovědi začíná v modulu pro analýzu XML, který v případě úspěchu vrací (7) získaná data či odpověď, kterou zaslal server. V poslední fázi jsou data předána (8) kontroleru, který si tuto akci vyžádal.

Kanály 1 a 8 jsou vnějším rozhraním síťové vrstvy používané kontrolérem. V případě volání 2, 3, 6 a 7 jde o vnitřní komunikaci **Networku** s modulem pro vytváření, resp. zpracování XML zpráv. V neposlední řadě jsou na schématu znázorněny zabezpečené kanály označené 4 a 5.

Stejný modul byl vyvinut spolu s kontrolérem pro platformu Windows Phone 8.1 a Windows 8 v jazyce C#. Pro vývoj bylo použito nejnovější Visual Studio 2013 Ultimate. I toto vývojové prostředí obsahuje vestavěný emulátor pro Windows Phone či tablet s operačním systémem Windows 8.1. V tomto případě byl pro vývoj využit převážně právě emulátor, pro jeho rychlost a jednoduchost v použití (emulátor může být použit teprve od Professional verze hostitelského operačního systému z důvodu využití Hyper-V technologie).

Pokud nebude řečeno jinak, bude další popis implementace uvažovat Android projekt.

6.1 XML procesor

V návrhu, diskutovaném v kapitole 5, jsem pro komunikační zprávy zvolil XML formát. S tím souvisí nutnost vytvořit na straně klienta submodul pro jeho vytváření a zpracování. Jeho vytváření je prováděno ve třídě **XmlCreator**, která je použita jako obalovací vrstva pro volání statických metod, které na základě předaných parametrů vytvářejí požadované XML zprávy. Na parsování přijatých zpráv od serveru je použita třída **XmlParsers**, která bezpečně získá z textového vstupu objekty obsahující data požadovaná klientem.

6.1.1 XmlCreator

Tato strukturou jednoduchá třída obsahuje statické metody, které jsou přímo volány ze síťové vrstvy. Každá z těchto konstrukčních veřejných metod začíná předponou *create* pokračujíc názvem zprávy, která má být zaslána na server. Například pro vytvoření zprávy obsahující příkaz pro přepnutí aktoru vypadá hlavička funkce takto:

public static String createSwitch(String bt, String aid, Device device);

Kde *bt* je unikátní relační identifikátor právě komunikujícího uživatele, *aid* je unikátní identifikátor brány (adaptéru), a *device* je objekt reprezentující senzory či aktory. V tomto případě se pro identifikaci požadovaného aktoru vybere adresa zařízení a jeho typ. Součástí je i hodnota, na kterou má být aktor přepnut.

Všechny hodnoty jsou převedeny na typ textového řetězce (String) a k následné serializaci je použita vestavěná knihovna **XmlSerializer**, které se nastaví kódování a jmenný prostor, v našem případě "*UTF-8*" a *null* (namespace není v tomto případě použito). Poté se pomocí volání metod **XmlSerializeru** *startTag* (přidání nového elementu, kde parametrem je jméno elementu), *atribute* (vložení atributu ke stávajícímu elementu, kde parametry jsou jméno přidávaného atributu a jeho hodnota) a *endTag* (ukončení aktuálního elementu, kde parametrem je jméno tohoto elementu), vytváří požadovaná XML struktura zpráv.

Z důvodu velkého počtu zpráv má třída relativně velký počet řádků, proto z důvodu zamezení dalšího růstu je velký počet metod, díky XML návrhu založeném především na atributech, realizován skrze volání speciální metody *CreateComAttribsVariant*. V ukázce 6.1 je ukázáno, jak metoda přijímá volitelný počet parametrů, které musí být sudého počtu (dvojice název a hodnota atributu), které jsou přidány do těla XML zprávy. Metody *beginXml* a *endXml* genericky vkládají začátek, resp. konec zprávy.

```
protected static String createComAttribsVariant(String... args) {
  if (0 != (args.length % 2)) { // odd
     throw new RuntimeException("Bad params count");
  }
  StringWriter writer = new StringWriter();
  try {
     XmlSerializer serializer = beginXml(writer);
     for (int i = 0; i < args.length; i += 2) { // take pair of args
            serializer.attribute(ns, args[i], args[i + 1]);
     }
     endXml(serializer);
     return writer.toString();
  } catch (Exception e) {
     throw AppException.wrap(e, NetworkError.CL_XML);
  }
}
```

Ukázka kódu 6.1: Metoda pro tvorbu zpráv obsahujících jen atributy

Pro udržování řetězcových konstant používaných ve zprávách, jako jsou názvy elementů a atributů je použita jednoduchá třída **XmlConstants**, což přispívá k přehlednosti, znuvupoužitelnosti a čitelnosti samotného kódu.

Stejně jako je tomu v případě Android aplikace je i pro Windows Phone pro tento účel realizována třída obsahující statické metody vytvářející XML zprávu. Postup je zde o něco jednoduší než s použitím Javy, například pro tvorbu nových elementů je vestavěná třída **XElement**, které je předán název elementu a instance dalších elementů či atributů, jako parametry konstruktoru. Obdobně je tomu u třídy **XAttribute,** která, jak vyplývá z názvu, vytváří objekty reprezentující atributy elementů. Pro názornost zde uvedu ukázku metody napsané v C#, která vykonává stejnou akci jako funkce z ukázky kódu 6.1.

```
private static string createComAttribsVariant(params string[] atribs)
{
    if (0 != (atribs.Length % 2)) // odd
            throw new Exception("Bad params count");

    XElement xml = new XElement(XmlConstants.COMMUNICATION,
                   new XAttribute(XmlConstants.VERSION,
                       XmlConstants.PROTOCOLVERSION));

    for (int i = 0; i < atribs.Length; i += 2)
            xml.SetAttributeValue(atribs[i], atribs[i + 1]);

    return xml.ToString();
}
```

Ukázka kódu 6.2: Ekvivalentní metoda tvorby zpráv v C#

6.1.2 XmlParsers

Tato třída instanciovaná ze síťové vrstvy obsahuje řadu privátních metod začínající prefixem “parse”, které dále pokračují názvem přijímané zprávy. Například pro zpracování zprávy obsahující informace o dostupných branách se použije metoda s touto hlavičkou:

private List<Adapter> parseAdaptersReady();

Hlavní metodou volanou z vnější je *ParseCommunication*, které se předá přijatá zpráva jako textový řetězec, a v případě úspěšného zpracování vrátí objekt *ParsedMessage*, který obsahuje stav (typ zprávy) a především získaná data. Pro parsování je využita vestavěná knihovna **XmlPullParser**. Metoda nejprve inicializuje instanci **XmlPullParseru** a následně získá ze zprávy číslo verze. V případě, že verze nesedí, řeší se problém podle verzovací politiky následovně:

1. Z čísla verze skládající se z hlavního a vedlejšího (majoritního a minoritního) čísla získá majoritní a porovná jej z verzí parseru.
2. Pokud čísla neodpovídají nelze pokračovat a zpracovávání je ukončeno vyvoláním výjimky.
3. Jinak jsou porovnána minoritní čísla.
4. Pokud je minoritní číslo zaslané ve zprávě nižší než verze parseru (server je zastaralý), je opět výpočet ukončen s chybou.

Pokud verze odpovídá, nebo je získaná verze vyšší (lišící se pouze ve vedlejším čísle), je získán stav, podle kterého se dále zpráva parsuje, to znamená, že jsou volány privátní metody specializované dle zpráv (stavů).

Tyto metody procházejí jednotlivé elementy pomocí volání metod **XmlPullParseru**, *nextTag* (přechod na další element), *getName* (jméno elementu) a *getAttributeValue* (hodnota atributu, kde parametrem je jeho název), volaná skrze vlastní funkci *getSecureAttrValue* vracející namísto *null* prázdný řetězec. Stejně jako v případě **XmlCreatoru** je i zde využita třída **XmlConstants**, pro udržení přehlednosti.

Řešení pro Windows Phone je také pomocí třídy **XmlParser**, v tomto případě však neobsluhuje všechny požadavky jediná veřejná metoda, která teprve na základě zpracovávání zprávy volá příslušnou proceduru, ale tato rutina je volána již ze síťové vrstvy. Tyto funkce dále k parsování používají vestavěné třídy **XmlReader**, jejíž instance se textem zprávy pohybuje pomocí metod jako je například *ReadToFollowing* nebo *MoveToAttribute*. Samotná hodnota je následně získána dle zvyklostí této platformy jako property objektu readeru (*reader.Value*).

Pro obě platformy platí, že pokud je zjištěna chyba na straně serveru a je detekována zprávou FALSE, je z ní získán chybový kód, který je přeložen ve vyšších vrstvách na textovou zprávu, kterou lze zobrazit uživateli. Následující tabulka 6.1 obsahuje základní chybové kódy a jejich popis.

kód	popis	poznámka
0	-	rezervováno
1	špatná verze komunikačního protokolu	aplikace nabídne aktualizaci
2	špatný google token, email, jméno	vypršela platnost, nebo nelze ověřit u poskytovatele
3	existující uživatel	tento uživatel již existuje, nelze použít toto jméno při registraci
4	nepodporovaná lokalizace	automaticky se zvolí angličtina
5	špatné číslo adaptéru	tento adaptér neexistuje
6	špatné číslo adaptéru	tento adaptér je již zaregistrovaný někým jiným
7	špatné číslo adaptéru	tento adaptér jste si již zaregistrovali
8	špatný typ agregační funkce	neexistující typ funkce u **GetLog** zprávy (*ftype* v zprávě 17)
9	špatný interval	interval není zarovnaný[3]
10	špatný typ a id	kombinace typu a id zařízení neodpovídá
11	špatný typ místnosti	neexistující typ místnosti (viz. tabulka 5.1)
12	poškozené XML	nastala chyba při vytváření zprávy
13	entita neexistuje	např. aktualizace aktoru po jeho smazání jiným uživatelem
14	špatná ikona	souvisí s chybou 11 při definování vlastní místnosti
15	špatná akce	neodpovídající akce
16	špatná práva	nedostatečná práva, např. na párovací režim
17	špatný email/role	v případě editace uživatele
18	špatné utc	špatná časová zóna
19	špatná hodnota aktoru	neodpovídající hodnota aktoru, např. mimo rozsah
20	špatný bt	relační token vypršel
21	nevhodné heslo k registraci	uživatel použil krátké, či jinak nevhodné heslo
22	nevhodné uživatelské jméno	použití nevhodných, či blokovaných slov (admin, atp.)
23	čekající uživatel	nelze přihlásit před aktivací účtu (email)
24	neexistující uživatel	špatné jméno a/nebo heslo
25	špatné heslo	špatné heslo, zaneprázdněnost poskytovatele
26	blokovaný uživatel	uživatel byl zablokován
27	neplatný poskytovatel	např. byl odstraněn z účtu
28	chybějící email	specifická chyba pro přihlášení přes poskytovatele Facebook
29	email neodpovídá	email neodpovídá danému poskytovateli
30	odhlášení se nezdařilo	chyba při odhlašování ze systému

Tabulka 6.1: Chybové kódy zasílané zprávou FALSE

6.2 Síťová vrstva

Komunikaci se serverem zajišťuje modul, který je realizovaný třídou **Network**, která je implementací rozhraní **INetwork**. Rozhraní je použito z důvodu jednotného přístupu k metodám z kontroleru například při použití demo režimu, což je režim aplikace nekomunikující se serverem, ale z pohledu uživatele plně funkční nad statickými ukázkovými daty uvnitř aplikace. Tento režim má za úkol kromě nápovědy, představit schopnosti aplikace a systému před koupí adaptéru a senzorů či aktorů. V takovém případě je použita implementace **DemoNetwork**, která pouze imituje síťovou komunikaci.

Network třída je instanciovaná kontrolerem, který jí přímo využívá skrze volání veřejných metod, které odpovídají názvům zpráv, které budou posílány. Každá z těchto metod interně volá

[3] interval: měsíc = 604800, týden = 604800, den = 86400, hodina = 3600, minuta = 60, raw = 0

XmlCreator nad předanými parametry. Následně zasílá XML zprávu na server a poté jí pomocí **XmlParseru** zpracuje a překontroluje její výsledek. V případě jakékoliv chyby je vyvolána odpovídající výjimka.

Třída používá TLS šifrování, a je schopná připojovat se alternativně na více serverů, dle vydání aplikace (zda se jedná o alpha verzi, beta verzi, nebo produkční verzi). Podle potřeby je možné komunikovat v jednotlivých relacích, nebo nechat relaci (socket) otevřenou a ušetřit tak čas, který konzumuje SSL handshake a ustavování spojení. Proces zasílání zpráv lze rozdělit do několika základních kroků:

1. Kontrola zda se jedná o jediný požadavek, či celou relaci.
    a. Pokud jde o relace, je použit dříve otevřený socket (volání *multiSessionBegin*, který je potřeba na konci relace zavřít voláním *multiSessionEnd*).
    b. Jde-li o jediný požadavek, vytvoří se socket, který se po přijetí odpovědi uzavře. Otevírání socketu obsahuje časově náročné operace jako je výpočet pravosti certifikátu a ustavení zabezpečeného spojení se serverem.
2. V dalším kroku je metodou *sendRequest* požadovaná zpráva zaslána na server.
3. Na odpověď čeká blokující funkce *receiveResponse*, která po úspěšném přijetí celé zprávy vrací textový řetězec obsahující XML.
4. Pokud se nejedná o relaci, je socket a další zdroje uzavřen a XML je předáno ke zpracování.

Kromě jiného obsahuje třída metodu *disconnect*, která ukončí aktuálně vykonávané požadavky a uzavře veškerá spojení se serverem. Využití této funkce je především ve chvíli, kdy server dlouho neodpovídá a uživatel si přeje ukončit aktuální akci. Druhou metodou používanou pro jiné než komunikační účely je *isAvailable*, což je jednoduchý test, zda je zařízení připojeno k internetu.

U síťové vrstvy doznala implementace pro Windows Phone oproti XML Procesoru větších změn. Na rozdíl od operačního systému Android, je politika bezpečnosti Windows nastavena tak, že je při každém spuštění aplikace potřeba dynamicky instalovat certifikát pro naši certifikační autoritu, oproti které je ověřováno zabezpečené TLS spojení se serverem. Oproti Javě je zde však lépe řešena problematika vláken, a to kouzelnými klíčovými slovy *async* a *await*, která se postarají o všechny detaily vícevláknového systému.

Podobně jako je tomu u druhého řešení, je i ve Windows Phone implementaci řada metod, které realizují požadavky kontroleru. Všechny tyto funkce uvnitř volají komunikační metodu *call2Server*, které je jako parametr předán řetězec s XML zprávou. Tato metoda se již dále postará o vytvoření zabezpečeného spojení, odeslání požadavku a přijetí odpovědi od serveru. V případě chyby je chybový kód i zpráva uložena do veřejné proměnné.

7 Výsledky

Výsledná aplikace byla v pravidelných intervalech dostupná skrze Google Play. Implementované řešení během vývoje bylo publikováno neveřejně, aby nebylo dostupné všem uživatelům. Testování probíhalo pouze u uživatelů, kteří dostali povolení k instalaci. Tito uživatelé byli rozdělení do dvou skupin podle stability a četnosti vydání aplikace na *alpha* a *beta*.

Během vývoje bylo vydáno a zveřejněno (testerům) na padesát verzí aplikace. Jak bylo zmíněno výše, tito testeři byli rozdělení na základě Google+ komunit, kdy pro získání aplikace musí být uživatel členem alespoň jedné. Posloupnost získávání nových verzí je nastavena tak, že *beta* tester může dostat pouze *beta* verzi aplikace (stabilnější), zatímco *alpha* tester může získat i *beta* verzi pokud je novějšího vydání než poslední *alpha* verze aplikace.

Alpha verze byly vydávány průběžně s menšími úpravami a opravami nekritických chyb, které byly používány především vývojáři z ostatních týmů pracujících na systému BeeeOn. Na druhou stranu *beta* verze byly publikovány spolu s plánovanými demonstračními dny přibližně jednou měsíčně, kde byly představeny nové funkce a pokroky ve vývoji.

Jednou z hlavních výhod publikace řešení skrze Google Play je možnost přímého reportování pádů vydané aplikace. Každý report, pokud je uživatelem odeslán, obsahuje trasování zásobníků s jednotlivým voláním metod. Díky této schopnosti lze rychle odhalit problematické části v kódu. Kromě posloupnosti volání obsahuje report i verzi aplikace a systému, počet výskytů této chyby a časové značky. Pokud jej tester vyplní, je k dispozici i komentář, který může například obsahovat posloupnost úkonů, které vedly ke kritické chybě. Počet testerů se vyšplhal k číslu 20, kde většina z nich byla součástí vývojového týmu BeeeOn (případně členové IoT).

Tito testeři měli kromě jiného přístup do systému Redmine[4], který kromě plánování úkolu a správě projektů, lze použít jako reportovací platformu pro chyby a nevalidní chování, která však nezpůsobilo pád a odeslání reportu přes Google play. V tomto hlášení je možno popsat problém a kroky vedoucí k jeho reprodukci, verzi aplikace a operačního systému, čas vzniku, a také přiřadit vývojáře k tomuto úkolu. Během používání tohoto systému bylo evidováno přibližně 30 podnětů k opravě chyb, či vylepšení některé funkce.

Vzhledem k tomu, že po plném zveřejnění aplikace, nebudou všichni uživatelé ochotní oznamovat pády a vyplňovat reporty, je pro reálné podmínky přidána komponenta s napojením na službu Google Analytics, která zasílá anonymní zprávy na pozadí. Na rozdíl od Google Play však nezasílá detailní sled volání, čímž je dohledání konkrétního problému mírně obtížnější. Obsahuje ale chování uživatele a základní údaje o něm. Také statistky pádů a lze nastavit vlastní proměnné a události, které budou automaticky sledovány.

V případě Windows Phone 8.1, resp. Windows 8.1 aplikace také zveřejněna v obchodě s aplikacemi, Windows Store. Obdobně viditelnost pro veřejnost byla omezena a aplikaci bylo umožněno nainstalovat pouze zvoleným testerům s mobilním zařízením s Windows Phone 8.1 operačním systémem (desktopová a tabletová verze byla distribuovaná jinou formou).

Samotný protokol byl inkrementální cestou vylepšován na základě potřeb serveru či vyšších vrstev aplikace a dočkal se již verze 2.5 (v ukázkách protokolu v příloze je pro názornost vždy použito 1.0).

[4] Dostupné z: http://www.redmine.org

7.1 Testování a experimenty

Pro ověření funkčnosti výsledného řešení byla provedena řada testů, která má za úkol prověřit, zda byly splněny definované podmínky na latenci a velikost zpráv, rychlost a úspornost. Testy pro platformu Android byly prováděny s mobilním telefonem ZTE Blade s operačním systémem Android 2.3.4. (API 12). Díky použití staršího zařízení byla ověřena zpětná kompatibilita. Další parametry zařízení jsou:

- Operační paměť - 512 MB RAM
- Konektivita - WIFI, 3G
- CPU - Qualcomm MSM7227 600 MHz (ARMv6).

Testy ověřující síťovou vrstvu byly prováděny na WIFI síti s rychlostí 16,58/2,26 Mb/s (download/upload) a pingem 18ms (testováno online SpeedTestem[5]), latence přímo na adresu serveru byla navíc také ověřena nástrojem ping v prostředí Windows - 18ms.

Měření na platformě Windows Phone probíhalo na zařízení Nokia Lumia 735 s operačním systémem Windows Phone 8.1 a také na emulátoru. Vlastnosti telefonu jsou následující:

- Operační paměť - 1 GB RAM
- Konektivita - WIFI, LTE
- CPU - Qualcomm MSM8926 1 200 MHz (ARMv7-A)

Tyto testy probíhaly jak na zmíněné WIFI sítí, tak také na celulární síti LTE s naměřenou rychlostí 44,99/5,14 Mb/s s pingem 110ms (opět použita služba SpeedTest.net, tentokrát prostřednictvím aplikace).

Vzhledem k odlišnosti uživatelského rozhraní je zde mimo jiné i jistá odchylka ve využití specifických komunikačních zpráv při obsluze stejného případu užití. Například na Windows Phone platformě není využita Google Cloud Messaging služba, tzn. notifikace. Zprávy, které zabezpečují registraci a další akce související s touto službou nejsou tedy použity ani implementovány.

7.1.1 Měření latence přihlášení

Prvním testem, je test latence u prvního zvoleného případu užití, který popisuje přihlášení do aplikace a automatické stažení informací o uživateli, zaregistrování zařízení pro získávání notifikací, stažení dat místností a všech zařízení. V tomto měření byl k účtu připojen jeden adaptér s dvěma senzory, každý ve vlastní místnosti (senzory byly bez dat grafu). Měřen byl čas od začátku komunikace, po zpracování XML zprávy (vytváření zprávy nebylo součástí měření). Tento případ užití je zobrazen na obrázku 7.1, kde jsou vypuštěny detaily, které byly ukázány na obrázku 5.1, kvůli irelevantnosti k výsledkům celkového měření (ovlivňují pouze první zprávu).

Z výsledků experimentu, který byl pětkrát opakován (pro obě varianty, jednou s použitím relace, jednou bez jejího použití), které jsou v tabulce 7.1, potvrzují předpoklady, že využití jedné relace pro více zpráv rapidně zkrátí dobu výpočtu. V konečném součtu průměrně až 2x zlepšení, v některých případech bylo však zlepšení až 5x. Všechny zprávy byly stejné délky, celkově 2080 (875 odeslané a 1205 přijaté) znaků pro jedno měření.

[5] Dostupné z: http://www.speedtest.net

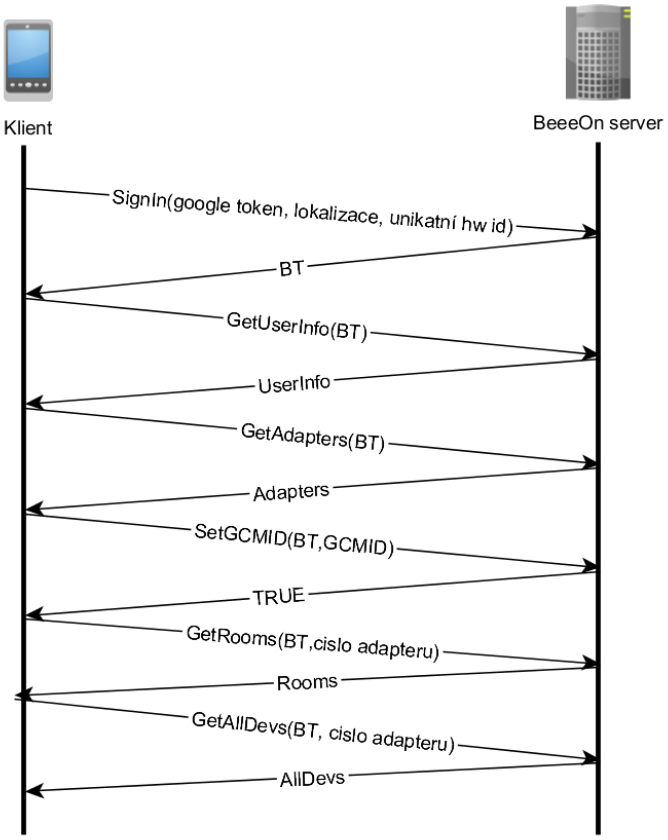
	SignIn + BT		GetUserInfo + UserInfo		GetAdapters + Adapters		Setgcmid + TRUE		GetRooms + Rooms		GetAllDevs + AllDevs		celkem	
délka (out/in) [znaky]	205	86	96	357	96	132	264	72	106	162	108	396	875	1205
android s použitím jedné relace [ms]														
1	642		91		27		100		27		37		924	
2	562		30		71		28		32		37		760	
3	561		25		1403		83		73		79		2224	
4	593		32		37		70		29		24		785	
5	566		31		25		27		48		81		778	
průměr	584,8		41,8		312,6		61,6		41,8		51,6		1094,2	
android bez použití relace [ms]														
6	991		306		237		249		306		473		2562	
7	803		421		222		186		184		448		2264	
8	776		249		278		240		249		560		2352	
9	782		206		225		206		185		358		1962	
10	979		292		301		287		340		515		2714	
průměr	866,2		294,8		252,6		233,6		252,8		470,8		2370,8	
windows phone bez použití relace [ms]														
emulator	2735		xxx		482		xxx		211		278		3706	
wifi	1839		xxx		199		xxx		189		188		2415	
wifi	1461		xxx		231		xxx		252		218		2162	
LTE	1948		xxx		265		xxx		227		224		2664	
LTE	1513		xxx		220		xxx		234		251		2218	
průměr	1899,2		xxx		279,4		xxx		222,6		231,8		2633	

Tabulka 7.1: Naměřené časy pro první testovaný případ užití

Vliv relace na zlepšení je ukryt v možnosti znovupoužít ustavené spojení, bez nutnosti znovu provádět SSL handshake, který zabere většinu času. Toto zpoždění lze pozorovat u zprávy **SingIn**, u které se, jakožto u první zprávy, ustavuje spojení i při využití relací. Vyšší naměřené hodnoty v obou případech jsou dány komunikací serveru s Google serverem (viz. obrázek 5.1). U třetího měření bylo zaznamenáno vyšší zpoždění, zřejmě kvůli latenci sítě, či směrování.

Třetí část tabulky obsahuje měření na platformě Windows Phone 8.1 a to jak s emulátorem, tak s reálným zařízením. Jak bylo řečeno dříve, některé zprávy nejsou použity a jsou označeny „xxx“. Kromě zmiňovaných notifikací jsou to informace o uživateli ve formě **GetUserInfo** a **UserInfo**, které nejsou v uživatelském rozhraní této aplikace na rozdíl od Androidu použity.

Z naměřených hodnot lze vyvodit podobné závěry jako v případě Androidí aplikace a to, že nejdéle trvá úvodní přihlášení a to v průměru až 2x déle, což zastoupí prodlevu chybějících zpráv. Ostatní hodnoty jsou již téměř stejné a tedy i celkový čas je průměrně velmi podobný. Kvůli problematice úvodního přihlášení byl vyvinut a implementován mechanismus automatického přihlašování, který spočívá v lokálním uložení *beeeOn tokenu* (*BT*) v paměti telefonu. Díky tomu není nutné se při každém zapnutí aplikace znovu přihlašovat. Tato akce je nutná pouze po vypršení platnosti tokenu, nebo po manuálním odhlášení.



Obrázek 7.1: První testovaný případ užití - přihlášení

Jak bylo řečeno výše, relace přinesly velké zrychlení. Je to patrné i na grafu 6.1, kde kromě zpráv **GetAdapters** a **Adapters**, je čas měřený s relací několikrát nižší. Při bližším měření jsem zjistil, že nejvíce procesorového času zabírá metoda **OpenSSLSocketImpl**.*getSession* a to až 66-85% jednoho měření. Pravým viníkem je pak funkce na výpočet pravosti certifikátu, který je použit pro komunikaci, **java/security/cert/CertPathValidate**.*validate*.

Graf 6.1: Srovnání komunikace s použitím a bez použití relací

7.1.2 Měření latence detailu senzoru a editace

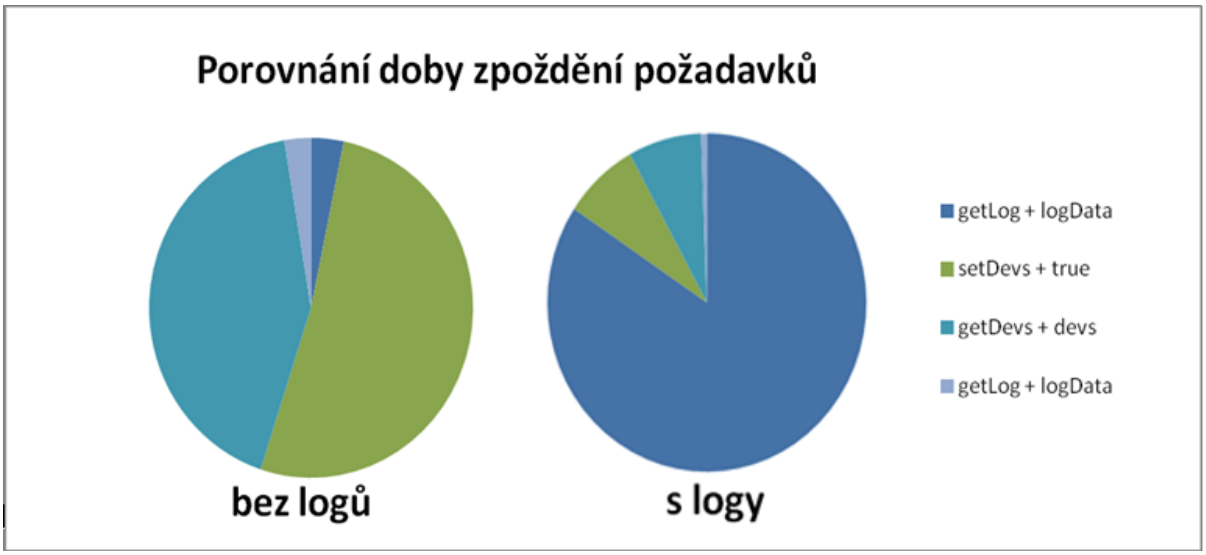
Jako druhý případ užití k testování jsem zvolil posloupnost úkonů vedoucí ke změně intervalu aktualizací hodnoty senzoru (přechod na detail senzoru, úprava hodnoty doby intervalu, návrat na seznam senzorů).

	getLog + logData		setDevs + true		getDevs + devs		getLog + logData		celkem	
délka [znaky]	189	76	164	73	165	255	189	76	707	480
s použitím jedné relace [ms]										
1	26		505		364		24			919
2	29		474		330		26			859
3	38		528		611		24			1201
4	28		475		374		24			901
5	32		458		314		27			831
průměr	30,6		488		398,6		25			942,2
s historickými daty										
délka [znaky]	189	4055	164	73	165	255	189	104	707	4487
6	4180		504		353		26			5063
7	3967		535		424		32			4958
8	4059		392		407		37			4895
9	7440		482		333		31			8286
10	4179		244		575		38			5036
průměr	4765		431,4		418,4		32,8			5647,6

Tabulka 7.2: Časy druhého testovaného případu užití

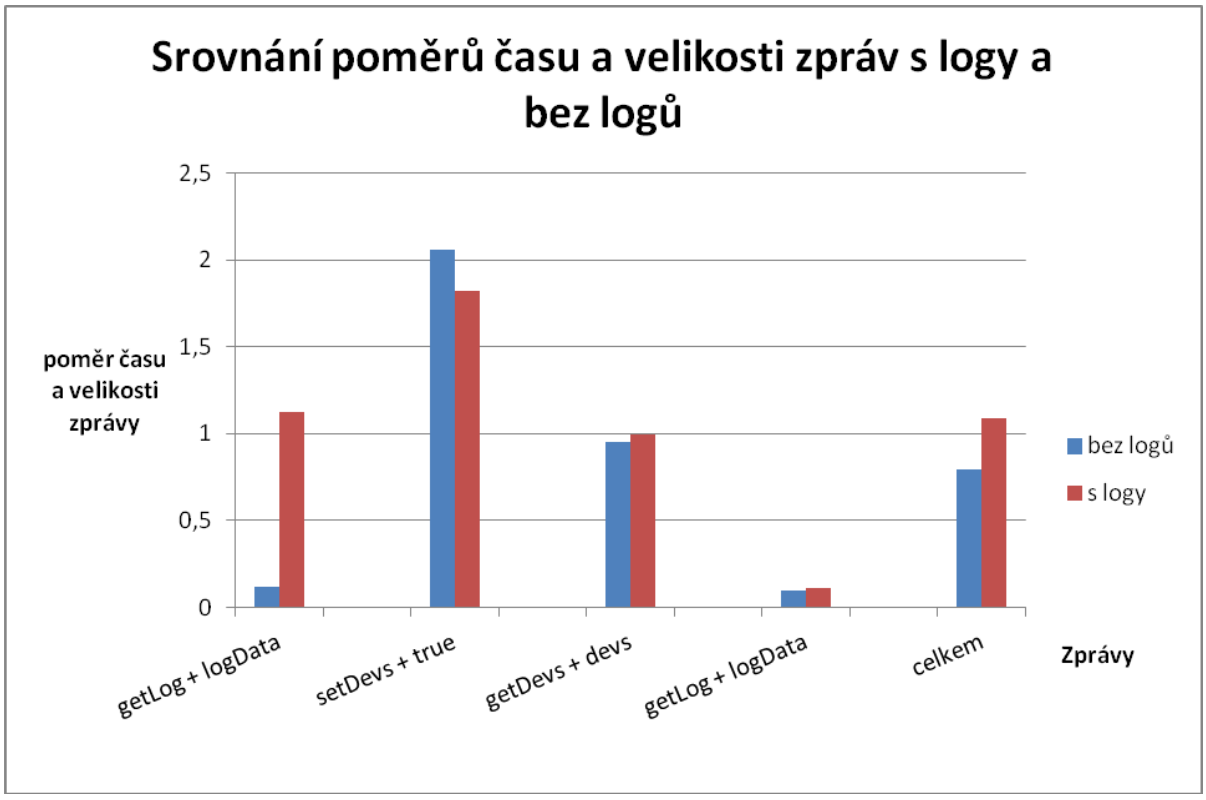
Data získaná měřením jsou v tabulce 7.2, kde lze pozorovat, že tím, že byly v první polovině použity senzory bez historických dat, je velikost odpovědi minimální, stejně tak i zpoždění těchto zpráv. Následující graf 6.2 pak shrnuje data z předchozí tabulky. Z grafu bez logů (vlevo) lze vyvodit, že zprávy manipulující s data senzorů na serveru trvají řádově déle, i přestože velikost zpráv je velmi podobná. Toto je zapříčiněno především ukládáním, resp. vybíráním dat senzoru z databáze na serveru. Zpracování XML na klientovi, nemůže mít takový vliv, protože v případě zprávy **SetDevs** se parsuje pouze odpověď **TRUE** a časy zpoždění jsou až o 20% nižší než v případě zpětného dotazu na uležená data, **GetDevs**, kde se zpracovává celá zpráva obsahující získaná data senzoru.

V druhé polovině tabulky jsou data naměřena s povolenými historickými hodnotami pro grafové zobrazení v aplikaci. Velikost zpráv obsahující logy (jediné lišící se) jsou zvýrazněny oranžovou barvou. Jak se dalo očekávat, velké množství dat, které obsahuje první odpověď s historickými hodnotami, se podepsala na celkovém zpoždění. V druhém případě volání **GetLog** je velikost řádově menší, protože je již přenášen pouze rozdíl oproti předchozímu stavu. Graf 6.2 tak srovnává poměrný nárůst doby trvání zpracování požadavku pro variantu s logy a bez nich.



Graf 6.2: Porovnání doby zpoždění jednolitých požadavků pro druhý testovací případ užití

Pro snadnější a přehlednější zobrazení rozdílů ve velikosti a zpoždění zpráv pracující s historickými hodnotami ukazuje graf 6.3 srovnání poměru času k velikosti jednotlivých zpráv. Nejmarkantnější rozdíl je samozřejmě u první zprávy obsahující objemné logy. Ostatní sloupce jsou velice podobné.



Graf 6.3: Srovnání poměru času k velikosti zprávy pro testy s logy a bez logů

Závěr

Cílem této práce bylo seznámit se s řešením pro inteligentní domácnost vyvíjeným v rámci projektu IOT na FIT VUT v Brně. Dále nastudovat požadavky pro komunikaci různých částí systému se serverovou částí. Studium následně vyústilo v návrh komunikačního protokolu pro komunikaci koncového ovládacího zařízení (chytrého telefonu, tabletu, aj.) se serverem (příp. adaptérem).

Úvodní část práce se věnuje definici problematiky domácí automatizace, resp. inteligentní domácnosti a s tím souvisejících požadavků a klíčových prvků. Další podkapitola je věnována systémům a řešením, které se již vyskytují na českém či světovém trhu a úspěšně se zde prosadily. Nedílnou součástí první kapitoly je samozřejmě rozbor systému, v jehož rámci je tento projekt vytvářen - BeeeOn. V této části popisuji jednotlivé prvky systému, které se podílejí na řízení, správě a monitoringu domácnosti. V neposlední řadě je zde část zaměřená na klientskou aplikaci pro mobilní zařízení k ovládání prvků domácnosti.

Dále jsem popsal teoretický základ nezbytný pro pochopení problematiky návrhu komunikačního protokolu a pozdější implementaci síťové vrstvy používající tento protokol pro zasílání zpráv mezi klientskou aplikací v mobilním zařízení a serverem distribuujícím požadavky dále do domácnosti. Také jsem zde popsal alternativní technologie, které by se daly využít, jako je JSON a HTTPS.

V následující kapitole jsem sestavil seznam požadavků jak na samotný komunikační protokol ze strany serveru, klientské aplikace, tak na samotného uživatele systému. Tyto požadavky pokrývají širokou škálu a někdy jdou proti sobě, proto jsem se dále zaměřil na ty nejdůležitější, jakými jsou latence komunikace, složitost a paměťová náročnost. V neposlední řadě jsem představil návrh jednotlivých zpráv, které jsou rozděleny do logických celků podle jejich funkcionality a pole působnosti. Všechny odchozí zprávy směrem k serveru (kromě zpráv registračních a přihlašovacích) obsahují unikátní identifikátor pro zvýšení bezpečnosti komunikace. Naopak všechny příchozí zprávy (kromě odpovědi na přihlášení) neobsahují tento identifikátor pro snížení paměťové náročnosti. Dalším významným krokem byl návrh generických odpovědí TRUE a FALSE na různé požadavky, což snížilo počet potřebných zpráv a složitost programu. Celý návrh s ukázkami struktury jednotlivých zpráv je poté součástí přílohy této práce.

Práce pokračuje praktickou částí, která začíná popisem implementace finálního řešení, které jsem vytvořil. Finálním řešením je v tomto kontextu myšlen síťový modul, který je použit v klientské aplikaci BeeeOn, jež ovládá prvky inteligentní domácnosti. Tato kapitola podrobněji rozebírá tři základní bloky realizované v rámci práce, jimiž jsou tvorba XML a jeho zpracování a samotná síťová komunikace skrze zabezpečenou relaci se serverem. Výběr formátu pro komunikační zprávy jsem založil na požadavcích, které byly na začátku stanoveny. Ohled byl brán především na požadavky, jakými jsou obecnost, rozšiřitelnost, multiplatformnost, nízká paměťová náročnost a čitelnost (snadno čitelné lidmi), čemuž vyhovoval formát XML. Komunikaci jsem po zhodnocení informací získaných studiem technologií popsaných v teoretické části práce implementoval pomocí protokolu SSL/TLS.

V poslední kapitole jsou diskutovány výsledky, kterých bylo dosaženo během vývoje aplikace. Dále jsou zde popsány testy a experimenty, které byly prováděny za účelem validace konečného řešení a získání parametrů, kterých aplikace dosahuje. Z požadavků na konečné řešení byl brán při testování ohled především na latenci komunikace, která byla měřena, a také na velikost zpráv. Požadavek na bezpečnost byl splněn použitím SSL/TLS šifrované komunikace. Z výsledků těchto

testů vyplívá, že použití jedné relace pro více požadavků byl krok správným směrem. Tento přístup výrazně snížil latenci komunikace a tím i odezvu celé aplikace a to především na pomalejších sítích a to v průměru až o 50%.

V rámci této diplomové práce byly splněny všechny body zadání, jako je návrh komunikačního protokolu, tedy celková specifikace všech zpráv potřebných k ovládání inteligentní domácnosti skrze server. Dále implementace tohoto protokolu a síťové vrstvy aplikace pro zmíněný systém BeeeOn a další body. Navíc nad rámec zadání byla implementována druhá aplikace pro platformu Windows Phone 8.1, která potvrdila multiplatformnost vytvořeného návrhu. Obě aplikace byly podrobeny testování a měření jejich parametrů a funkčnosti.

Výsledky této práce budou dále použity v rámci systému BeeeOn, který bude brzy veřejně publikován na českém trhu jako open-source. Do budoucna je počítáno s rozšířením protokolu o další typy zpráv podle potřeb uživatelů.

Literatura

[1] GERHART, James. *Home automation and wiring*. New York: McGraw-Hill, c1999, xiv, 322 p. Sweet's cost guides. ISBN 00-702-4674-2.

[2] BRYCHTA, Tomáš. *Bezdrátové senzory pro inteligentní domácnost*, bakalářská práce, Brno, FIT VUT v Brně, 2014

[3] STEVENS, Richard. *TCP/IP iIllustrated*. Vyd. 1. Boston: Addison-Wesley, 1994, 576 s. ISBN 02-016-3346-9.

[4] MOTYČKOVÁ, Lenka. *Distribuované systémy: Výpočty v sítích*. 1. vyd. Veletiny: Science, 1997, 178 s. ISBN 80-901-4758-5.

[5] BANDARA, H. M. N. Dilum a Anura P. JAYASUMANA. Collaborative applications over peer-to-peer systems–challenges and solutions. *Peer-to-Peer Networking and Applications*. 2013, vol. 6, issue 3, s. 257-276. DOI: 10.1007/s12083-012-0157-3. Dostupné z: http://link.springer.com/10.1007/s12083-012-0157-3

[6] FREIER, A., P. KARLTON a P. KOCHER. The Secure Sockets Layer (SSL) Protocol Version 3.0. In: *IETF Tools* [online]. 2011 [cit. 2015-01-07]. Dostupné z: http://tools.ietf.org/html/rfc6101

[7] DIERKS, T., E. RESCORLA. The Transport Layer Security (TLS) Protocol Version 1.2. In: *IETF Tools* [online]. 2008 [cit. 2015-01-07]. Dostupné z: http://tools.ietf.org/html/rfc5246

[8] Extensible Markup Language (XML) 1.0 (Fifth Edition). In: *W3C* [online]. 2008 [cit. 2015-01-07]. Dostupné z: http://www.w3.org/TR/REC-xml/#rfc2376

[9] *International CES* [online]. 20115 [cit. 2015-01-15]. Dostupné z: http://www.cesweb.org/

[10] *Haidy: Chytré a úsporné řešení* [online]. 2014 [cit. 2015-01-15]. Dostupné z: http://haidy.cz/cs

[11] *Canny Net: Komfort bez kompromisů* [online]. 2014 [cit. 2015-01-15]. Dostupné z: http://www.cannynet.com/

[12] BELKIN. *WeMo* [online]. 2014 [cit. 2015-01-15]. Dostupné z: http://www.belkin.com/us/Products/home-automation/c/wemo-home-automation/

[13] *Elektrobock CZ* [online]. 2014 [cit. 2015-01-15]. Dostupné z: http://www.elektrobock.cz/cs/inteligentni-dum/text.html?id=34

[14] *Insteon* [online]. 2014 [cit. 2015-01-15]. Dostupné z: http://www.insteon.com/

[15] *Insight Home: Řešení pro chytré bydlení* [online]. 2014 [cit. 2015-01-15]. Dostupné z: http://www.insighthome.eu/inHome.html

[16] *Time.gov* [online]. 2015 [cit. 2015-07-28]. Dostupné z: http://time.gov/images/worldzones.gif

[17] *Introducing JSON* [online]. 2015 [cit. 2015-07-28]. Dostupné z: http://json.org/

Seznam příloh

Příloha 1: Ukázky komunikačních zpráv

SignUp

Jak je vidět na ukázce zprávy 1, každá zpráva je definována jedním kořenovým elementem *com* (communication) a obsahuje řadu atributů. Prvním atributem je *ver* (version), který obsahuje verzi komunikačního protokolu. Dále je zde atribute *state*, který oznamuje stav, resp. typ zprávy.

Následuje další atribut, který již podávají serveru informace o uživateli. *Srv* je identifikací poskytovatele přihlášení, v tomto případě Google. Na základě zvoleného poskytovatele obsahuje element *par* příslušné atributy. Pro Google přihlášení je to *gt* (Google token), který je použit k verifikaci toho, zda se skutečně jedná o uživatele, za kterého se vydává. Tento token je časově omezený.

```
<com
    ver="1.0"
    state="signup"
    srv="google" >
    <par
        gt="ya29.YwCZ9wjs8MQnahwAAABNoED_qEEMqC-IQvnXG4X-auYGIRw"
    />
</com>
```

Zpráva 1: SignUp

SignIn

Zpráva pro přihlášení je velmi podobná registrační zprávě, navíc však obsahuje atribut *pid* (phone ID), což je unikátní hardwarové číslo identifikující dané zařízení. A také je zde atribut *loc* (locale), který nastavuje na serveru lokalizaci, kterou bude server dále používat při komunikaci s tímto klientem. Jako poskytovatel přihlášení je v ukázce zprávy použit BeeeOn, který požaduje jméno a heslo pro autentizaci.

```
<com
    ver="1.1"
    state="signin"
    pid="5455548989657"
    loc="cs"
    srv="beeeon" >
    <par
        name="frantisekVostrouzka"
        pswd="645655"
    />
</com>
```

Zpráva 2: SignIn

BT

```
<com
    ver="1.0"
    state="bt"
    bt="10c2682a647i9e9"
/>
```

Zpráva 3: BT

GetUserInfo

```
<com
    ver="1.0"
    state="getuserinfo"
    bt="1026877"
/>
```

Zpráva 4: GetUserInfo

UserInfo

V této zprávě reprezentující informace o uživateli jsou základní údaje jako je jméno, příjmení, pohlaví nebo email. Dále je zde element *accounts*, který obsahuje záznamy o všech účtech, ke kterým dal uživatel přístup. V této ukázce jsou v elementech *srv*, ukázáni poskytovatelé Google, Facebook a BeeeOn.

```
<com
    ver="1.0"
    state="userinfo"
    uid="10c2682a647i9e9"
    name="frantisek"
    surname="vorisek"
    gender="male"
    email="vorisek@gmail.com"
    imgurl="https://mujavatar.beeeon.com">
    <accounts>
        <srv name="google" id="1445511455"/>
        <srv name="beeeon" id="frantisek_omacka"/>
        <srv name="facebook" id="75asd1545"/>
    ...
    </accounts>
</com>
```

Zpráva 5: UserInfo

Logout

Zpráva pro odhlášení má naprosto stejnou strukturu jako zpráva **GetUserInfo**, s tím rozdílem, že stav obsahuje klíčové slovo *logout*.

JoinAccount

Stejně tak tato zpráva je téměř identická s již představenou zprávou, **SignUp**. Navíc je přidán zde pouze atribut *bt*, který je nutný k rozpoznání přihlášeného uživatele.

CutAccount

```
<com
    ver="1.0"
    state="cutaccount"
    bt="42"
    srv="google"
/>
```

Zpráva 6: CutAccount

TRUE

```
<com
    ver="1.0"
    state="true"
/>
```

Zpráva 7: TRUE

FALSE

```
<com
    ver="1.0"
    state="false"
    errcode="99">
        plaintext v dané lokalizaci popisující blíže chybu
</com>
```

Zpráva 8: FALSE

AddAdapter

Zpráva **AddAdapter**, stejně jako další zprávy zasílané směrem na server obsahuje atributy *bt*(unikátní identifikátor), *ver* (verze) a *state* (stav). Kromě těchto atributů obsahuje atribut *aid* (adapter ID), což je sériové číslo adaptéru a *aname* (adapter name), které nese uživatelem definované jméno adaptéru (např. Domov).

```
<com
    ver="1.0"
    bt="42"
    state="addadapter"
    aid="51966"
    aname="home"
/>
```

Zpráva 9: AddAdapter

GetAdapters

```
<com
    version="1.0"
    bt="42"
    state="getadapters"
/>
```

Zpráva 10: GetAdapters

Adapters

Tato odpověď od **serveru** obsahuje kromě *ver* a *state* atributů seznam adaptérů. Každý prvek seznamu je XML elementem obsahující *id* adaptéru, atribut *name* nesoucí uživatelem definované jméno. Dále pak *role*,který obsahuje definici práv pro daný **adaptér** (kapitola 2.2.4). Posledním atributem je *utc* (Coordinated Universal Time), který obsahuje informaci o tom, v jakém časovém pásmu byl **adaptér** spárován, tzn. kolik hodin je v místě, kde se **adaptér** nachází. Přesněji, tento atribut obsahuje minutový rozdíl proti GTM (Greenwich Mean Time), např. pro Českou Republiku je hodnota 60, značící UTC+1, resp. ECT (European Central Time).

```
<com ver="1.0" state="adapters">
    <adapter id="65260" name="home" role="admin" utc="60"/>
    <adapter id="64206" name="work" role="superuser" utc="120"/>
</com>
```

Zpráva 11: Adapters

ScanMode

```
<com
    ver="1.0"
    bt="42"
    state="scanmode"
    aid="51966"
/>
```

Zpráva 12: ScanMode

ReInitAdapter

```
<com
    ver="1.0"
    bt="42"
    state="reinitadapter"
    oaid="64206"
    naid="51966"
/>
```

Zpráva 13: ReInitAdapter

GetGateInfo

Tento požadavek má stejnou strukturu jako zpráva **GetAdapters**, s tím rozdílem, že obsahuje navíc element *aid*, který definuje data brány,která mají být zaslána.

GateInfo

Informace získané touto zprávou reprezentují požadovaný adaptér. Kromě ustálených elementů, jsou tu elementy jako *role* definující práva uživatele na této bráně, jméno a IP adresa brány, časová zóna, ve které je brána (adaptér) umístěna, či verze systému, který na adaptéru běží. Pro další statistické účely je zde element *nfacs* nesoucí informaci o počtu připojených zařízení a element *nusers*, který označuje počet uživatelů používající tuto bránu.

```
<com
    ver="1.0"
    state="gateinfo"
    aid="12349"
    role="admin"
    aname="adapter frantisek"
    nfacs="3"
    nusers="3"
    ip="192.168.0.1"
    aversion="0"
    utc="10"
</com>
```

Zpráva 14: GateInfo

SetGate

Obsahuje nastavitelné prvky brány jako jsou název a časová zóna.

```
<com
    ver="1.0"
    state="setgate"
    bt="1026877"
    aid="1234"
    aname="adapter1"
    utc="1"
/>
```

Zpráva 15: GateInfo

GetAllDevs

```
<com
    version="1.0"
    bt="42"
    state="getalldevs"
    aid="51966"
/>
```

Zpráva 16: GetAllDevs

GetNewDevs

```
<com
    ver="1.0"
    bt="42"
    state="getnewdevs"
    aid="51699"
/>
```

Zpráva 17: GetNewDevs

GetDevs

Zpráva obsahuje seznam prvků definujících **adaptér** na první úrovni, jeho zařízení (senzory, aktory) v druhé úrovni a navíc, pokud se jedná o multi-zařízení (obsahující více čidel/měřících elementů) pak i výběr jeho specifické části.

```
<com
    ver="1.0"
    bt="42"
    state="getdevs">
    <adapter id="51966">
        <dev id="120:50:FF:000:FFE">
            <part type="1" />
            <part type="2" />
        </dev>
    </adapter>
</com>
```

Zpráva 18: GetDevs

GetLog

Stejně jako v případě předchozí zprávy se jedná o jednu ze složitějších. Kromě klasických atributů, *ver*, *bt*, *state* a *aid*, také *from*, obsahující unixovou časovou značku reprezentující čas, od kterého mají být zalogovaná data poslána. Dále *to* (koncová časová značka), *ftype* (typ agregační funkce, např. avg, sum, atp.), *interval* (šířka časového intervalu, který bude použit pro agregační funkce) a v neposlední řadě *did* (device ID - identifikátor zařízení, obvykle MAC) a *dtype* (identifikace čidla uvnitř senzoru).

```
<com
    ver="1.0"
    bt="42"
    state="getlog"
    from="1377684610"
    to="1377684650"
    ftype="avg"
    interval="86400"
    aid="51966"
    did="120:07:FF:000"
    dtype="1"
/>
```

Zpráva 19: GetLog

AllDevs

Každý prvek seznamu obsahuje identifikaci zařízení *did* (device ID), *lid* (location ID - identifikace místnosti, kde je zařízení umístěno), *refresh* (délka intervalu uspání v sekundách), *batt* (battery - hodnota baterie v procentech), *time* (čas poslední aktualizace hodnot), *rssi* (Received Signal Strenght Indication - síla bezdrátového signálu u daného senzoru) a dále pak seznam čidel, pokud jde o multi-zařízení. Každý z prvků seznamu čidel pak nese informaci o svém typu, atribut *type*, o poslední hodnotě, atribut *val* (value), také atribut *vis* (visibitlity - zda má být čidlo zobrazeno, či nikoliv) a samozřejmě uživatelem definovaný název, atribut *name*. Ukázka zprávy 14 je opět co nejkratší, obsahuje tedy pouze jediné zařízení definované elementem *dev* (device). Obecně se však jedná o velmi velkou zprávu.

```
<com
    ver="1.0"
    state="alldevs"
    aid="51966">
    <dev
        did="100:00:FF:000:FF0"
        lid="5555"
        refresh="5"
        batt="100"
        time="1377684610"
        rssi="52" >
        <part
            type="1"
            vis="1"
            name="Teplota na okně"
            val="30" />
    </dev>
</com>
```

Zpráva 20: AllDevs

Devs

```
<com ver="1.0" state="devs">
    <adapter id="51966">
        <dev
            init="1"
            did="120:00:FF:000:FFE"
            lid="1"
            refresh="10"
            batt="70"
            time="1377684610"
            rssi="52" >
            <part
                type="1"
                vis="1"
                name="Teplota u dveří"
                val="20" />
        </dev>
    </adapter>
</com>
```

Zpráva 21: Devs

LogData

```
<com ver="1.0" state="logdata">
    <row>1377684610 30</row>
    <row repeat="20" interval="3600">1377684610 30</row>
    <row>1377684610 33</row>
</com>
```

Zpráva 22: LogData

SetDevs

```
<com ver="1.0" bt="42" state="setdevs" aid="51966">
    <dev
        init="1"
        did="120:00:FF:000:FFE"
        lid="1"
        refresh="10" >
        <part
            type="1"
            vis="1"
            name="teplota u dveří" />
    </dev>
</com>
```

Zpráva 23: SetDevs

Switch

```
<com
    ver="1.0"
    bt="42"
    state="switch"
    aid="51966"
    dtype="0"
    did="120:00:FF:000:FFE"
    val="1"
/>
```

Zpráva 24: Switch

DelDev

```
<com
    ver="x.x"
    bt="42"
    state="deldev"
    aid="51966"
    did="120:00:FF:000:FFE"
/>
```

Zpráva 25: DelDev

AddRoom

```
<com
    ver="1.0"
    bt="42"
    state="addroom"
    aid="51966"
    ltype="2"
    lname="Obývací pokoj"
/>
```

Zpráva 26: AddRoom

SetRooms

Pokud chce uživatel místnost přejmenovat, použije se právě tato zpráva, která obsahuje kromě běžných atributů, jako jsou *ver*, *bt*, *state* a *aid*, také seznam *loc* (location) elementů, kde každý reprezentuje právě jednu místnost. Každý element obsahuje *id*, *type* a *name* pro popis místnosti. Odpověď **serveru** je opět **TRUE**, nebo **FALSE**.

```
<com
    ver="1.0"
    bt="42"
    state="setrooms"
    aid="51966">
    <loc id="5" type="2" name="Obývací pokoj" />
    <loc id="3" type="3" name="WC" />
</com>
```

Zpráva 27: SetRooms

DelRoom

```
<com
    ver="1.0"
    bt="42"
    state="delroom"
    aid="51966"
    lid="5"
/>
```

Zpráva 28: DelRoom

GetRooms

```
<com
    ver="1.0"
    bt="42"
    state="getrooms"
    aid="51966"
/>
```

Zpráva 29: GetRooms

Rooms

```
<com
    ver="1.0"
    state="rooms"
    aid="51966" >
    <loc id="5" type="1" name="Obývací pokoj" />
    <loc id="6" type="2" name="WC" />
</com>
```

Zpráva 30: Rooms

RoomID

```
<com
    ver="1.0"
    state="roomid"
    lid="5"
/>
```

Zpráva 31: RoomID

AddAccs

```
<com
    ver="1.0"
    bt="77"
    state="addaccs"
    aid="51966">
    <user
        email="user3@gmail.com"
        role="user"/>
</com>
```

Zpráva 32: AddAccs

SetAccs

```
<com
    ver="1.0"
    bt="77"
    state="setaccs"
    aid="51966">
    <user
        email="user3@gmail.com"
        role="user"/>
</com>
```

Zpráva 33: SetAccs

DelAccs

```
<com
    ver="1.0"
    bt="74"
    state="delaccs"
    aid="51966">
    <user email="user3@gmail.com"/>
</com>
```

Zpráva 34: DelAccs

GetAccs

```
<com
    ver="1.0"
    bt="42"
    state="getaccs"
    aid="51966"
/>
```

Zpráva 35: GetAccs

Accounts

```
<com
    ver="1.0"
    state="accounts">
    <user
        email="user@gmail.com"
        role="user"
        name="hanka"
        surname="sroubalova"
        gender="0"/>
</com>
```

Zpráva 36: Accounts

SetTimeZone

```
<com
    ver="1.0"
    bt="42"
    state="settimezone"
    aid="51966"
    utc="-120"
/>
```

Zpráva 37: SetTimeZone

GetTimeZone

```
<com
    ver="1.0"
    bt="42"
    state="gettimezone"
    aid="51966"
/>
```

Zpráva 38: GetTimeZone

TimeZone

```
<com
    ver="1.0"
    state="timezone"
    utc="60"
/>
```

Zpráva 39: TimeZone

SetLocale

```
<com
    ver="1.0"
    bt="42"
    state="setlocale"
    loc="sk"
/>
```

Zpráva 40: SetLocale

Kompletní seznam pohledů vlhkosti

- **GetHumOverview** - požadavek na přehled vlhkosti
- **HumOverview** - přehled vlhkosti domácnosti
- **GetHumMsgDetail** - požadavek na detailní popis události
- **HumMsgDetail** - zpráva s detailem vzniklé události
- **GetHumLocDetail** - požadavek na přesná data vlhkosti v dané místnosti
- **HumLocDetail** - detail vlhkosti požadované místnosti
- **GetHumLocStats** - požadavek na získání statistik vlhkosti v místnosti
- **HumLocStats** - statistiky vlhkosti dané místnosti
- **SetHumNotifs** - nastavení notifikace pro vlhkost
- **GetHumNotifs** - požadavek na získání notifikací
- **HumNotifs** - notifikace ohledně vlhkosti
- **SetHumThreshold** - nastavení prahu pro sledování vlhkosti
- **GetHumThreshold** - získání prahu vlhkosti

Kompletní seznam zpráv pro podmínky a akce

- **AddAlg** - přidání algoritmu
- **AlgCreated** - algoritmus úspěšně přidán
- **GetAllAlgs** - požadavek na všechny algoritmy
- **Algs** - seznam všech definovaných algoritmů (pravidel)
- **GetAlgs** - požadavek na vybrané algoritmy
- **SetAlg** - úprava některého z algoritmů
- **DelAlg** - smazání algoritmu

Kompletní seznam notifikací

- **DelGCMID** - smazání GCMID (Google Cloud Messaging ID)
- **SetGCMID** - nastavení GCMID
- **GetNotifs** - dotaz na seznam notifikací
- **Notifs** - seznam notifikací
- **NotifRead** - označení notifikace za přečtenou
- **PassBorder** - geolokační zpráva oznamující server o změně polohy uživatele

Příloha 2: Obsah CD

src/android/	složka se zdrojovými soubory aplikace pro platformu Android
src/windowsphone/	složka projektem a zdrojovými soubory pro platformu Windows Phone 8.1
build/apk/beeeon.apk	zkompilovaná aplikace pro platformu Android
zprava/zprava.pdf	zpráva ve formátu PDF
zprava/doc/	složka se zdrojovými soubory diplomové práce ve formátu DOCX