VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

INTEGRACE NOVÝCH BEZDRÁTOVÝCH TECHNOLOGIÍ A ZAŘÍZENÍ DO BEEEON BRÁNY

INTEGRATION OF NEW WIRELESS TECHNOLOGIES AND DEVICES

INTO THE BEEEON GATEWAY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. DAVID BEDNAŘÍK**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. JAN KOŘENEK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání diplomové práce

Student: **Bednařík David, Bc.**
Program: Informační technologie Obor: Informační systémy
Název: **Integrace nových bezdrátových technologií a zařízení do BeeeOn brány**
Integration of New Wireless Technologies and Devices into the BeeeOn Gateway
Kategorie: Vestavěné systémy

Zadání:

1. Seznamte se s architekturou brány systému BeeeOn.
2. Nastudujte zařízení podporující technologie Bluetooth LE (systémy Tabu Lumen, Revogi), Sonoff a Conrad HomeMatic.
3. Navrhněte způsob integrace alespoň jedné z výše uvedených technologií do systému BeeeOn.
4. Proveďte implementaci navrženého řešení a ověřte korektnost připojení pro vybranou sadu zařízení.
5. Navrhněte a implementujte kompletní sadu testů pro bránu BeeeOn a pomocí vytvořených testů bránu odlaďte.
6. Diskutujte dosažené výsledky a možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Kořenek Jan, doc. Ing., Ph.D.**
Konzultant: Krejčí Radek, RNDr., CESNET
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 3. června 2020
Datum schválení: 25. října 2019

Abstrakt

Tato diplomová práce pojednává o integraci nových zařízení od výrobců Revogi, Tabu Lumen, Sonoff a HomeMatic do softwaru BeeeOn Gateway. Teoretická část práce se zabývá architekturou softwaru BeeeOn Gateway a popisuje vlastnosti, chování a způsob komunikace se zařízeními od výše zmíněných výrobců. Součásti této části je také popis komunikačních technologií, které zmíněna zařízení využívají. Patří zde Bluetooth Low Energy, WiFi a rádio s frekvencí 868 MHz. Praktická část zmiňuje způsob rozšíření softwaru BeeeOn Gateway o moduly, které implementují podporu inteligentních zařízení. V této části je také popsán způsob ověřování správnosti implementace a testování celého softwaru BeeeOn Gateway. Testování brány je realizováno pomocí jednotkových a integračních testů, které ověřují chování jednotlivých komponent brány, ale také celé brány.

Abstract

This master thesis deals with the integration of new devices from the manufacturers Revogi, Tabu Lumen, Sonoff and HomeMatic into the BeeeOn Gateway software. The theoretical part deals with the architecture of the BeeeOn Gateway software and describes the characteristics, behavior and way of communication with devices from the above mentioned manufacturers. This part of thesis also contains a description of the communication technologies used by these devices. They include Bluetooth Low Energy, the WiFi and the 868 MHz radio. The practical part mentions the way of extension of BeeeOn Gateway software to modules that implement support for smart devices. This section also describes how the correctness of implementation was verified and testing of the entire BeeeOn Gateway software. The testing of gateway is performed by unit and integration tests, which verify the behavior of individual gateway components as well as the whole gateway.

Klíčová slova

Inteligentní domácnost, BeeeOn, Revogi, Tabu Lumen, Sonoff, HomeMatic, BeeeOn Gateway, MQTT, Bluetooth Low Energy, FHEM

Keywords

Smart Home, BeeeOn, Revogi, Tabu Lumen, Sonoff, HomeMatic, BeeeOn Gateway, MQTT, Bluetooth Low Energy, FHEM

Citace

BEDNAŘÍK, David. *Integrace nových bezdrátových technologií a zařízení do BeeeOn brány*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Jan Kořenek, Ph.D.

Integrace nových bezdrátových technologií a zařízení do BeeeOn brány

Prohlášení

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

.....
David Bednařík
26. května 2020

Poděkování

Zde bych rád poděkoval vedoucímu diplomové práce Doc. Ing. Janu Kořenkovi Ph.D. za odborné vedení a konzultace jak při psaní této práce, tak při návrhu a implementaci praktické části práce.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Motivací za vznikem inteligentních budov bylo a stále je, zvýšení komfortu, bezpečí a snížení spotřeby energií. V raných stadiích inteligentních budov se tento koncept více odkazoval na automatizaci než inteligenci. Hlavní rozdíl mezi těmito dvěma termíny je, že automatizace používá komponenty systému pouze pro zajištění optimálního chování, zatímco inteligence se snaží dynamicky měnit parametry systému za účelem předvídat a optimalizovat očekávání uživatele. Historii inteligentních budov lze rozdělit do několika fází. Během první fáze byly inteligentní budovy osazené automatizačním systémem, který měl za úkol šetřit pracovní čas. V další fázi se vývoj inteligentních budov zaměřil na šetření energií a zvýšení bezpečnosti. Rozvoj inteligentních budov pokračoval přidáváním osamocených funkcí (ovládání osvětlení, klimatizace a další), které byly následně integrovány do jednoho systému. Jednou z posledních fází je využití počítačové sítě jako infrastruktury inteligentní budovy. Současné inteligentní budovy můžeme klasifikovat podle velikosti:

- **Malé inteligentní budovy** - typickým příkladem je rodinný dům. V této kategorii inteligentních budov je kladen největší důraz na bezpečnost, komfort a efektivní využití energetických zdrojů.
- **Inteligentní budovy střední velikosti** - mezi tuto kategorii budov lze zařadit nemocnice, školy, hotely a tak dále. V této kategorii budov je důraz kladen na nízké náklady na provoz a opravu, na efektivní využití energetických zdrojů a na bezpečnost.
- **Velké inteligentní budovy a rozsáhlé komplexy budov** - do této kategorie inteligentních budov patří letiště, univerzitní kampusy či velké administrativní komplexy budov. Hlavní motivací u této kategorie budov jsou nízké náklady na provoz a opravu.

Tato práce se primárně zabývá malou inteligentní budovou tzv. chytrou domácností. Jedná se o dům či bytovou jednotku, ve které se nachází skupina zařízení, pomocí které realizujeme požadované služby bezpečnosti, komfortu a efektivního využití energetických zdrojů. Tuto skupinu zařízení lze rozdělit na senzory, aktory a řídící jednotku.

Senzory slouží k zjišťování stavu prostředí, ve kterém se nacházejí. Typickým příkladem může být senzor měřící různé fyzikální veličiny jako například teplota, vlhkost, koncentrace oxidu uhličitého ve vzduchu, ale také zde patří senzor detekující pohyb nebo internetová kamera. Na druhou stranu aktory slouží k změně prostředí, ve kterém se nacházejí. Příkladem aktoru může být zásuvka, kotel nebo alarm.

Jako poslední je zde řídicí jednotka, která má za úkol sbírat data ze senzorů a na základě těchto dat a definovaných pravidel měnit prostředí pomocí aktorů. Odlehčenou variantou řídící jednotky je brána. Ta pouze umožňuje přeposílat data ze senzorů na server a přijímat

a vykonat příkazy ze serveru na změnu stavu aktoru. Server může data uchovávat a na základě nich se rozhodovat jaké akce provede ve formě odeslání příkazu na změnu stavu aktoru na bránu. Rozšiřovaná *BeeeOn* brána v rámci této práce patří do této skupiny odlehčených řídících jednotek.

Bezpečnost domácnosti je možné realizovat pomocí detektoru pohybu, webové kamery, okenních a dveřních senzoru, od kterých řídící jednotka přijímá data a v případě, že detekuje určitou anomálii (například pohyb v domácnosti v určitou dobu, kdy by se nikdo v domácnosti neměl nacházet) může spustit alarm nebo poslat notifikaci uživateli. Zvýšeni komfortu uživatele lze dosáhnout například inteligentní žárovkou nebo zásuvkou, kterou může uživatel ovládat vzdáleně z pohodlí křesla či postele, nebo senzorem měřící koncentraci oxidu uhličitého, na základě kterého řídící jednotka může odesílat uživateli rady v podobě notifikací, aby vyvětral a tím snížil koncentraci oxidu uhličitého na přijatelnou hodnotu. Pro efektivní využití energetických zdrojů je možné použít senzor teploty, okenní senzor a chytrou termostatickou hlavici. Na základě teploty a stavu okna může řídící stanice zapínat a vypínat (když je teplota nad určitou hranici nebo je otevřené okno a venkovní teplota je nižší než vnitřní) vytápění a tím efektivně využívat teplou vodu.

Cílem této práce bylo rozšířit *BeeeOn* bránu o podporu zařízení od výrobců *Revogi*, *Tabu Lumen*, *Sonoff* a *HomeMatic*. Implementaci jednotlivých vytvořených modulů otestovat a ověřit správnost komunikace brány s inteligentními zařízeními. Hlavním přínosem systému *BeeeOn* je integrace zařízení od různých výrobců komunikující různými protokoly a technologiemi. Tento problém také řeší systém *openHAB*[1] implementován v programovacím jazyce Java. Na rozdíl od *BeeeOn* brány systém *openHAB* není možné nasadit na standartní síťový směrovač.

Obsah práce je dělen to několika logických celků. První část práce je zaměřena na architekturu *BeeeOn* brány. Je zde popsáno z jakých částí a služeb se brána skládá a jak jsou jednotlivé části brány propojeny. Další část se věnuje jednotlivým technologiím a zařízením, pro které je vytvářena podpora do *BeeeOn* brány. Součástí popisu každého zařízení, pro které je vytvářena podpora, je chování zařízení, vlastnosti zařízení, způsob komunikace se zařízením a způsob lokalizace zařízení. Čtvrtá a pátá kapitola je zaměřena na návrh a implementace modulů pro podporu nových technologií a zařízení. V kapitolách je obsažen diagram návrhových tříd jednotlivých modulů, implementační detaily a také nástroje a knihovny, které byly při implementaci použity. Předposlední kapitola popisuje jakým způsobem byly nové moduly testovány a jak byla testována brána jako celek. Jsou zde zahrnuty jak jednotkové testy vybraných komponent tak integrační testy celého softwaru *BeeeOn* brány.

[1] https://www.openhab.org

Kapitola 2

BeeeOn brána

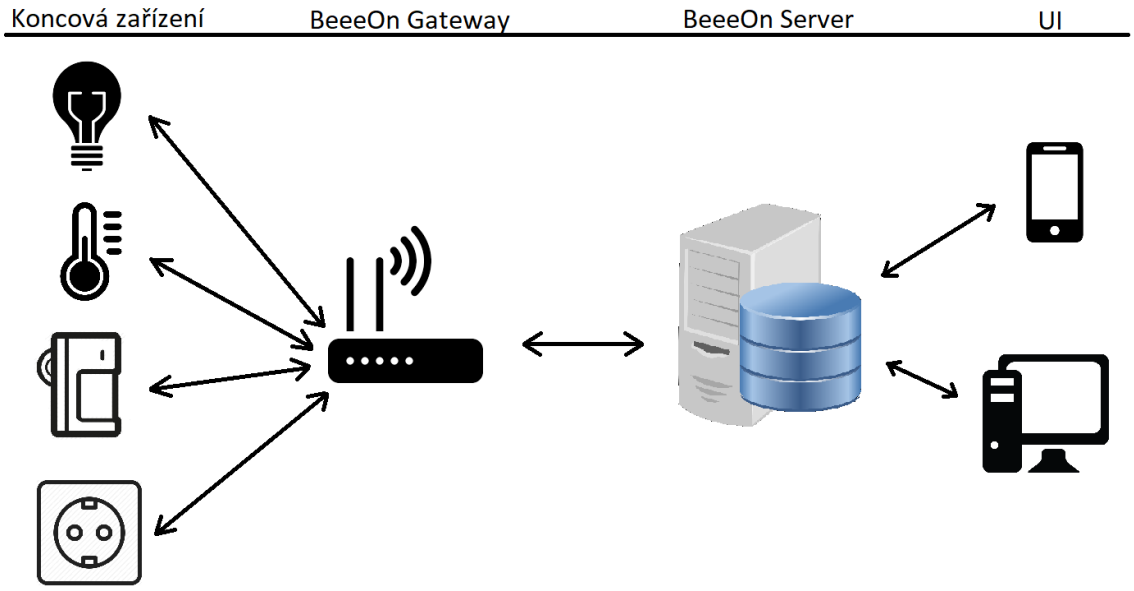
Tato kapitola představuje a podrobně popisuje software *BeeeOn* brány. Brána je zařazena do celého systému *BeeeOn* a ke každé části je dostupný popis funkce, které daná část vykonává. Taktéž je zde zmíněna architektura softwaru brány, tudíž z jakých celků se software brány skládá, co jednotlivé části vykonávají a jaké mezi nimi existují vazby. Software brány *BeeeOn*, jak už název napovídá, slouží jako brána mezi senzorovou a počítačovou síti. Úkolem brány je komunikovat a ovládat zařízení v senzorové síti pomocí různých bezdrátových technologií na základě požadavků uživatele, který s bránou interaguje pomocí počítačové sítě.

Hlavní přínos této brány a celého systému *BeeeOn* je sjednocení podpory velké škály **IoT** zařízení a tím poskytnout uživateli komfortní způsob řízení inteligentní domácnosti, složené ze zařízení různých výrobců, pouze pomocí jedné aplikace. Významnou úlohou brány je také sběr statistických dat o jednotlivých senzorových sítích, na základě kterých lze dále zkoumat a detekovat různé bezpečností anomálie.

2.1 Architektura systému BeeeOn

Architektura systému *BeeeOn* je znázorněna diagramem, jenž je dostupný na obrázku 2.1. Diagram znázorňuj jednotlivé části, ze kterých se systém *BeeeOn* skládá a jak tyto části mezi sebou komunikují. Komunikace mezi částmi je reprezentovaná orientovanými hranami, kdy orientace hrany definuje směr komunikace.

Na nejnižší úrovni jsou koncová zařízení, se kterými komunikuje *BeeeOn* brána pomocí různých protokolů a technologii. Příkladem těchto technologií a protokolů může být *Bluetooth*, *IQRF*, *ZWave* nebo *Wifi*. *BeeeOn* brána je spravovaná *BeeeOn* serverem, se kterým komunikuje pomocí počítačové sítě. Na nejvyšší úrovni jsou uživatelská rozhraní, která taktéž komunikují s *BeeeOn* serverem a zajišťují přístup uživatele do systému *BeeeOn*. Uživatelská rozhraní slouží k zobrazení nasbíraných dat ze senzorů, ale také k odeslání příkazu na změnu stavu zařízení.



Obrázek 2.1: Architektura systému BeeeOn

2.1.1 Koncová zařízení

Na nejnižší úrovni systému *BeeeOn* se nacházejí inteligentní zařízení různých výrobců, která komunikují různými technologiemi. Příkladem může být, jak lze vidět na digramu, senzor, chytré okno nebo bezdrátový spínač. Pomocí koncových fyzických zařízení systém *BeeeOn* zajišťuje služby inteligentní domácnosti. Tyto zařízení se mohou skládat z různých modulů, které lze rozdělit do dvou kategorií:

- **Senzorový modul** – jedná se o pasivní modul zařízení, který většinou nabízí měření různých fyzikálních veličin. Typickým příkladem fyzikálních veličin může být teplota, vlhkost, koncentrace oxidu uhličitého, intenzita osvětlení nebo hladina intenzity zvuku.
- **Aktorový modul** – aktivní modul, který má schopnost měnit svůj stav (otevřeno/-zavřeno, zapnuto/vypnuto, jas nebo barva osvětlení). Tyto moduly umožňují měnit prostředí, ve kterém se nacházejí.

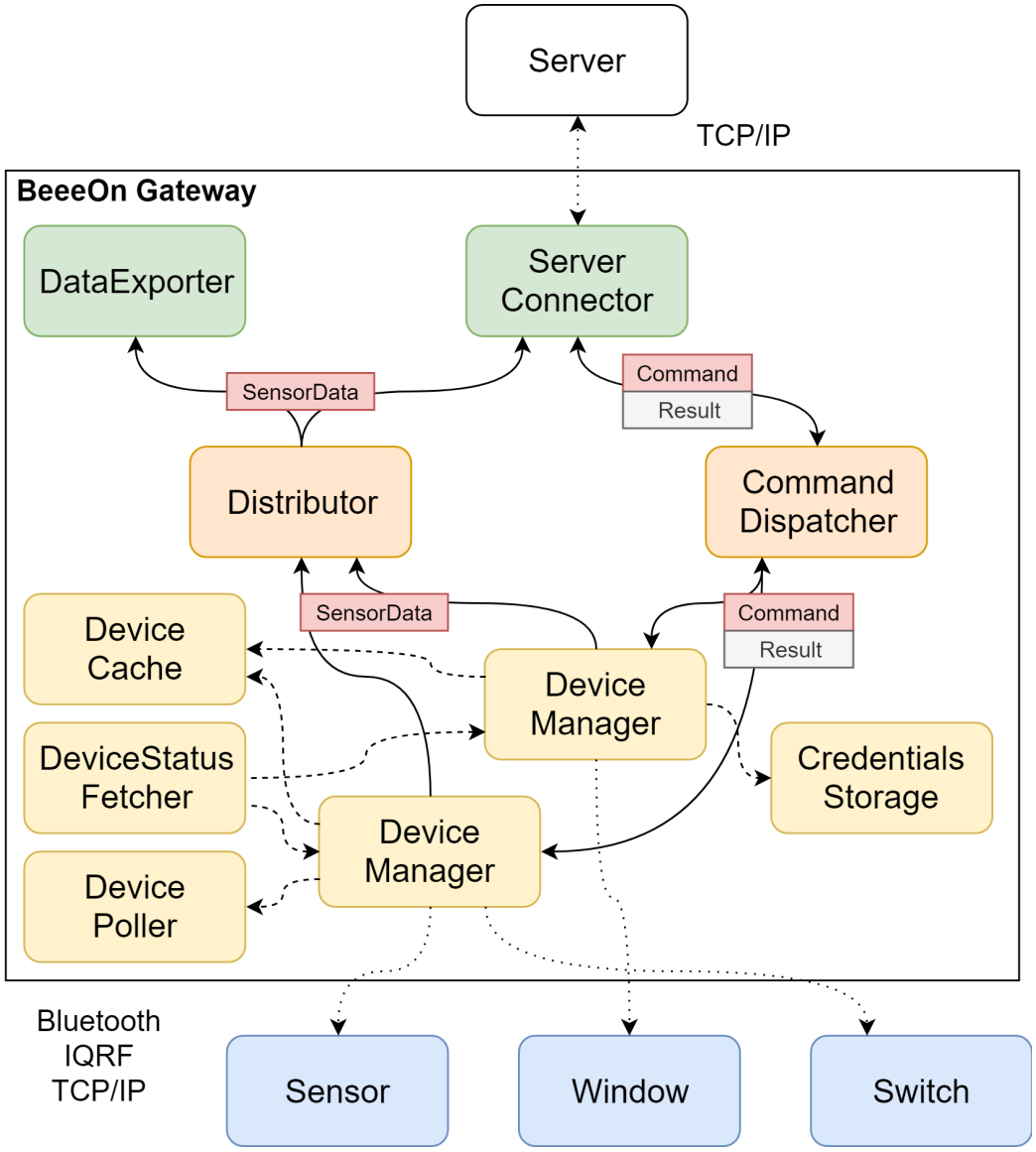
2.1.2 Server

O úroveň výše než je *BeeeOn* brána se nachází *BeeeOn* server. Ten zprostředkovává komunikaci mezi uživatelem a jeho bránami. Úkolem serveru je správa uživatelů, správa připojených bran a inteligentních zařízení a ukládat naměřená data ze všech připojených zařízení. Aby mohl uživatel ovládat svou inteligentní domácnost odkudkoliv z internetu, běží server na veřejné adrese. Uživatel může se serverem interagovat, a tím také ovládat své brány, pomocí mobilní aplikace dostupné pro operační systém *Google Android*. Server disponuje *REST API*, pomocí kterého si mobilní aplikace a server vyměňuji zprávy.

2.2 Architektura BeeeOn brány

O propojení výše zmíněných vrstev systému *BeeeOn* se stará brána. Hardware pro běh této software brány není pevně stanoven. Brána je vyvíjená, tak aby ji bylo možné spustit na libovolné distribuci operačního systému Linux.

Obecná struktura softwaru brány vyobrazena na obrázku 2.2 znázorňuje jednotlivé třídy softwaru brány a komunikaci mezi těmito třídami. Objekty obsažené v diagramu představují *C++* třídy. Součástí diagramu jsou také externí entity, se kterými software brány spolupracuje. Mezi externí entity patří server a inteligentní zařízení chytré domácnosti.



Obrázek 2.2: Architektura softwaru BeeeOn brány

Třídy softwaru brány lze rozdělit do tří úrovní. Třídy nejvyšší úrovně se starají o export nasbíraných senzorových dat a o příjem a odesílání řídicích příkazu. Do této skupiny patří třída **ServerConnector** a skupina tříd **DataExporter**. Třída **ServerConnector**, jak už název napovídá, komunikuje ze serverem, komunikace spočívá ve vzájemném vyměňování

si příkazů, které jsou popsány v sekci 2.3. Kromě příkazů také odesílá získaná senzorová data směrem na server. Třídy typu **DataExporter** slouží k pouhému sdílení získaných dat z připojených koncových zařízení. Aktuálně jsou v bráně implementovány dvě třídy tohoto typu a to **MqttExporter** a **NamedPipeExporter**.

Dále jsou zde třídy prostřední úrovně, které zaručují distribuci senzorových dat všem exportérům a doručení příkazů třídám, pro které jsou určené. Třída **Distributor** se stará o první úkol a tedy odesílání senzorových dat všem registrovaným třídám, které o tyto data mají zájem. Senzorová data jsou v softwaru brány reprezentovány třídou **SensorData**. Objekty této třídy jsou předávány mezi objektem **Distributor** a okolními objekty, mezi které patří třídy exportující data a třídy manažera zařízení. Druhý úkol, přeposílání jednotlivých příkazů mezi moduly, zaručuje třída **CommandDispatcher**. Třída, která chce zpracovávat příkazy, musí implementovat rozhraní **CommandHandler**. Příkazy jsou reprezentovaný třídou **Command**, jejíž objekty jsou předávaný mezi moduly. Ke každému příkazu je vytvořen objekt třídy **Result**, který nese výsledek příkazu.

Na nejnižší úrovni jsou třídy, které pracují s koncovými zařízeními. To znamená, že tyto třídy získávají data ze zařízení, mění jejich stav na základě přijatého příkazu a synchronizují stav zařízení podle serveru. Zde patří třída **DeviceManager**, která zmíněné akce vykonává. Pro každou komunikační technologii nebo výrobce inteligentních zařízení existuje v softwaru brány samostatná třída **DeviceManager**, která implementuje specifickou komunikaci a řízení dané skupiny chytrých zařízení. Fungování manažera zařízení je podrobně popsáno v sekci 2.4.

2.3 Komunikační kanály brány

Aby brána mohla komunikovat s prvky vyšší úrovně systému *BeeeOn*, obsahuje několik komunikačních kanálů, které slouží k exportu nasbíraných senzorových dat a přijímaní/odesílání příkazů.

2.3.1 Export dat

Export dat zajišťují třídy typu **DataExporter**, které získaná senzorová data naformátují do předem definované podoby (příkladem může být JSON[1] formát nebo CSV[2] formát) a exportují je na definované komunikační rozhraní. Momentálně jsou v bráně implementovány dva různé exportéry.

- **NamedPipeExporter** – používá pro předávání senzorových dat pojmenovanou rouru. V rámci konfigurace tohoto exportéru je nutné určit cestu, kde má exportér vytvořit pojmenovanou rouru a v jakém formátu má data odesílat. Ukázka konfigurace tohoto exportéru je dostupná ve výpisu 2.1. Pomocí parametru `pipe.enable` se povoluje či zakazuje spuštění exportéru. Dále parametr `pipe.path` definuje, kde se má vytvořit pojmenovaná roura a parametry `pipe.format` a `pipe.csv.separator` definují formát zapisovaných dat do roury.

  ```
  [exporter]
  pipe.enable = yes
  pipe.path = /var/run/beeeon/gateway/exporter
  pipe.format = CSV
  ```

[1] JavaScript Object Notation
[2] Comma-separated values

```
pipe.csv.separator = ;
```

Výpis 2.1: Konfigurace NamedPipeExporter

- **MqttExporter** – používá pro přenos senzorových dat síťový komunikační protokol *MQTT*. Pro správné fungování exportéru je nutné definovat IP adresu a port *MQTT Brokeru*, ke kterému se exportér připojí a bude pomocí něho sdílet senzorová data. Tyto informace jsou modulu předány pomocí parametrů `mqtt.host` a `mqtt.port` v konfiguračním souboru. Ukázka konfigurace modulu je dostupná ve výpisu 2.2. Další důležitou informací, kterou exportér potřebuje vědět, je na jaké *MQTT* téma má publikovat senzorová data. Tato informace je exportéru předaná pomocí konfiguračního parametru `mqtt.topic`. Mezi další konfigurovatelné parametry patří `mqtt.qos` definující kvalitu služeb, parametr `mqtt.clientID` určující unikátní název *MQTT* klienta a poslední parametr `mqtt.format` definuje stejně jako u **NamedPipeExporter** formát přenášených dat.

```
[exporter]
mqtt.enable = yes
mqtt.host = localhost
mqtt.port = 1883
mqtt.topic = BeeeOnOut
mqtt.qos = 0
mqtt.clientID = Gateway
mqtt.format = JSON
```

Výpis 2.2: Konfigurace MqttExporter

2.3.2 Příjem a odesílání příkazů

Druhá kategorie komunikačních kanálů umožňuje ovládat bránu a zařízení, která jsou k bráně připojena, pomocí příkazů. Primárně zde patří třída **GWSConnector**, která zajišťuje komunikaci mezi bránou a serverem. Do komunikace patří příjem a odesílání příkazů a jejich odpovědi, ale také přenos senzorových dat. Přenos zpráv je realizován pomocí komunikačního protokolu *WebSocket*. Zprávy jsou přenášeny ve formátu JSON. Výčet všech podporovaných příkazů je dostupný v následujícím seznamu:

Registrace brány (gateway-register) – tento příkaz je typicky odesílán bránou po jejím spuštění a navázaní spojení se serverem. Brána se pomocí tohoto příkazu registruje u serveru. Příkaz je odesílán také v případě, kdy dojde k opětovnému navázání spojení se serverem.

Seznam napárováných zařízení (device-list) – každý manažer zařízení si uchovává seznam všech jeho napárovaných zařízení, který může být uložen perzistentně, ale také neperzistentně. Za účelem synchronizace tohoto seznamu je při spuštění brány odeslán tento příkaz pro každého manažera zařízení. Výsledkem tohoto příkazu je seznam zařízení, která jsou z pohledu serveru napárované pro daného manažera zařízení. K nekonzistencím mezi seznamem napárovaných zařízení na bráně a na serveru může dojít například v případě, kdy brána nevratně ztratí svůj seznam napárovaných zařízení.

Poslední hodnota zařízení (last-value) – motivace tohoto příkazu je podobná jako u předchozího příkazu, a to zajištění konzistence. Příkaz umožňuje bráně, dotázat

se serveru na poslední nastavenou hodnotu modulu daného zařízení a následně tuto hodnotu nastavit na fyzickém zařízení. Vhodné využití tohoto příkazu je například, když si zařízení při ztrátě napájení nepamatuje svůj poslední stav a při inicializaci se nastaví na předem definovanou hodnotu. Potom se při obnově napájení brána může dotázat na poslední nastavenou hodnotu a tu na zařízení obnovit a tím zajistit konzistenci.

Objevování zařízení (listen) – příkaz odeslán serverem v případě, když chce uživatel nalézt nová zařízení v senzorové síti. V příkazu je obsažena doba trvání vyhledávání nových zařízení. Brána tento příkaz předá všem spuštěným manažerům zařízení, kteří po definovanou dobu spustí proces hledání zařízení. Manažer zařízení pro každé nalezené zařízení, které ještě není napárováno, odesílá příkaz *new-device* na server. Příkaz *new-device* obsahuje informace o nalezeném zařízení. Mezi informace obsažené v tomto příkazu patří: název zařízení, výrobce zařízení, seznam modulů a další specifické informace jako IP adresa, MAC adresa a sériové číslo.

Hledání konkretního zařízení (device-search) – v případě, kdy server vyžaduje po bráně, aby nalezla pouze jedno konkretní zařízení, je vhodné využít tento příkaz. Tento příkaz není povinně implementován ve všech manažerech zařízení. Příkaz obsahuje:

- identifikaci manažera zařízení, který má hledání konkretního zařízení provést
- identifikace zařízení, které se má najít pomocí IP adresy, MAC adresy nebo sériového čísla
- maximální dobu vyhledávání zařízení

Napárování zařízení (device-accept) – aby manažer zařízení mohl začít ovládat dané zařízeni, je nutné jej nejprve napárovat. K napárování slouží tento příkaz. Příkaz obsahuje 64-bitový identifikátor zařízení, které se má napárovat. Manažer zařízení, který zařízení udržuje, se určí pomocí prvních 8 bitů v identifikátoru zařízení. Po napárování je ze zařízení umožněno získávat data a ovládat jej. Tento příkaz končí úspěchem, když se zařízení povede korektně napárovat. V případě, že proces párování skončí neúspěchem, například manažer zařízení dané zařízení nezná, končí příkaz chybou `FAILED`, která je odeslaná zpět na server.

Odpárování zařízení (unpair) – tento příkaz má opačný efekt než tomu je u příkazu *device-accept*. Přijetím toho příkazu manažer zařízení odstraní dané zařízení rozpoznáno jeho identifikátorem obsaženým v příkazu ze seznamu napárovaných zařízení. Manažer zařízení tedy přestane sbírat data z odpárovaného zařízení a neumožní jej ovládat (měnit jeho stav). Příkaz končí úspěchem, když se povede dané zařízení odpárovat, ale i v případě, kdy dané zařízení manažer zařízení nezná a nemá jej napárované.

Nastavení hodnoty zařízení (set-value) – pro ovládání zařízení je určen tento příkaz. Příkaz obsahuje identifikátor zařízení, identifikátor modulu, hodnotu, na kterou se má daný modul nastavit a časový limit určující maximální dobu, po kterou se má manažer zařízení snažit o nastavení modulu na novou hodnotu. Po přijetí příkazu bránou je příkaz předán manažeru zařízení, který se o dané zařízení stará, což je určeno podle identifikátoru zařízení. Následně se manažer zařízení snaží o nastavení nové hodnoty modulu na zařízení. Příkaz končí neúspěchem v případě, kdy manažer zařízení dané zařízení nezná, nebo není schopen se k němu připojit, nebo nastavení

dané hodnoty modulu na zařízení končí chybou. V takovém případě je předaná chyba `FAILED` v podobě objektu **Result** serveru.

2.3.3 Testovací prostředí

Posledním komunikačním kanálem brány je testovací prostředí **TestingCenter**, které umožňuje přímé ovládání brány bez přítomnosti serveru. Testovací prostředí je určeno pouze k testování brány. Při povolení tohoto modulu, brána otevře port, na kterém začne naslouchat. Po připojení uživatele na tento port se otevře vzdálená konzole, pomocí které lze odesílat příkazy do brány. Konfigurace modulu je dostupná ve výpisu 2.3. Parametrem `center.enable` se povoluje spuštění modulu. Pomocí parametru `center.pairedDevices` lze testovacímu prostředí zadat seznam identifikátoru zařízení, která jsou již napárovaná. Parametry `center.tcp.address` a `center.tcp.port` definují na jaké IP adrese a na jakém portu bude testovací prostředí naslouchat. Testovací prostředí má vlastní interní neperzistentní reprezentaci napárovaných zařízení, to znamená, že když dojde k napárovaní zařízení jiným způsobem než přes **TestingCenter**, tak testovací prostředí dané zařízení nemá označené jako napárované.

```
[testing]
center.enable = yes
center.pairedDevices =
center.tcp.address = 127.0.0.1
center.tcp.port = 6000
```

Výpis 2.3: Konfigurace TestingCenter

TestingCenter umožňuje spravovat napárovaná zařízení a modifikovat jejich stav, spravovat úložiště hesel a přihlašovacích údajů a simulovat databázi zařízení na serveru. Výčet důležitých podporovaných příkazů testovacím prostředím je následující:

- `command listen [<timeout>]` – spustí vyhledávání nových zařízení. Volitelným parametrem příkazu je časový limit, který je implicitně 5 sekund.
- `command device-accept <device-id>` – napáruje zařízení definované jeho identifikátorem.
- `command unpair <device-id>` – odpáruje zařízení definované jeho identifikátorem.
- `command set-value <device-id> <module-id> <value> [<timeout>]`– pokusí se nastavit novou hodnotu zadaného modulu na zařízení definovaného identifikátorem. Volitelným parametrem příkazu je časový limit, který je implicitně 0 sekund.
- `device list-new` – zobrazí všechna zařízení, pro která byl odeslán příkaz `new-device`.
- `device list-paired` – zobrazí všechna zařízení, která má testovací prostředí označena jako napárovaná.
- `credentials set <device-id> pin <pin>` – uloží pin pro dané zařízení do úložiště důvěrných dat.
- `credentials set <device-id> password <username> <password>` – uloží přihlašovací jméno a heslo pro dané zařízení do úložiště důvěrných dat.

- `credentials show <deviceID>` – zobrazí všechna důvěrná data (pin, přihlašovací údaje) pro dané zařízení.
- `credentials save` – perzistentně uloží všechny změny provedené nad úložištěm důvěrných dat.

2.4 Manažer zařízení

Software *BeeeOn* brány se skládá z několika modulů manažera zařízení, kdy každý manažer zařízení obsluhuje určitou technologicky podobnou skupinu zařízení. Implementuje specifické procesy pro hledání zařízení v dané senzorové síti, pro napárování zařízení, pro odpárování zařízení, pro sběr dat z napárovaných zařízení a pro ovládání napárovaných zařízení. Hlavními úkoly modulu manažera zařízení je

- provedení příkazů přijatých od serveru,
- správa dostupných zařízení v senzorové síti,
- komunikace se zařízeními (sběr dat ze zařízení a změna stavu zařízení).

Každý manažer zařízení dědí abstraktní třídu **DeviceManager**, která implementuje společné chování všech manažerů zařízení. Zde patří příjem a zpracování příkazů, vyhodnocení vykonání příkazů, odesílání příkazů a senzorových dat a práce s napárovanými zařízeními. Moduly manažera zařízení jsou spuštěny v samostatném vlákně, ve kterém typicky provádí zpracování přijatých událostí z napárovaných zařízení nebo sběr dat z napárovaných zařízení. Zpracování přijatého příkazu ze serveru nebo z testovacího prostředí je vykonáváno v samostatném vlákně, které vytvoří **CommandDispatcher**. To umožňuje zpracování různých příkazů současně. **DeviceManager** vykovává stejné příkazy sekvenčně ve stejném pořadí, jak byly přijaty bránou.

2.4.1 Podpůrné moduly

V rámci vývoje softwaru brány byly identifikovány často se vyskytující stejné funkcionality, implementované ve většině manažerů zařízení. Tyto funkcionality byly extrahovány z jádra brány do samostatných modulů, čímž se zjednodušila zodpovědnost manažerů zařízení a sjednotil se způsob implementace těchto funkcionalit. Mezi podpůrné moduly manažera zařízení patří:

- **DeviceCache** – úkolem modulu je spravovat veškerá napárovaná zařízení pro všechny manažery zařízení. Existují dvě implementace **DeviceCache**, kdy jedná uchovává napárovaná zařízení v RAM operační paměti a druhá je uchovává perzistentně na pevném disku či paměťové SD kartě. Výhodou druhé implementace **FilesystemDeviceCache** je, že ihned po startu může brána vyhledat již napárovaná zařízení a začít s nimi pracovat.
- **DeviceStatusFetcher** – modul zajišťuje aktualizaci seznamu napárovaných zařízení po startu brány. Při výpadku brány se může seznam napárovaných zařízení na serveru změnit a je nutné tyto změny provést i lokálně na bráně. Proto při startu brány se **DeviceStatusFetcher** dotáže serveru na všechna napárovaná zařízení a v momentě kdy obdrží odpověď předá jednotlivá napárovaná zařízení konkretním manažerům

zařízení. Implicitní zpracování manažerů zařízení je aktualizace **DeviceCache**. Toto chování je ale možné změnit v konkretní implementaci manažera zařízení, pomocí překrytí metody **handleRemoteStatus**.

- **DevicePoller** – tento modul je možné využít v případě, kdy se manažer zařízení periodicky dotazuje svých napárovaných zařízení na jejich stav. **DevicePoller** tedy nabízí periodické sbírání dat ze zařízení. Aby mohl manažer zařízení svá zařízení registrovat u **DevicePoller**, musí třídy reprezentující jednotlivá zařízení, dědit třídu **PollableDevice** a implementovat její metody **poll** a **refresh**. Metodu **poll** potom v intervalech určených návratovou hodnotu metody **refresh** periodicky volá **DevicePoller**.

2.5 Sestavení softwaru BeeeOn brány

Z předchozích sekcí vyplývá, že software brány se skládá z několika modulů, které mají mezi sebou určité závislosti. Aby nebylo nutné při každé změně konfigurace brány (povolení/zakázání modulu, změna závislostí mezi moduly), celý software brány od začátku kompilovat, je použitá technika vkládání závislostí (Dependency injection). Tato technika dynamicky vytváří a spravuje pouze ty služby/moduly, které jsou povoleny při startu brány, a dynamicky vytváří vztahy mezi spuštěnými moduly. Konfigurace jednotlivých modulů je předaná třídě **DependencyInjector** ve formě *XML* konfiguračního souboru. Stejným způsobem jsou také předávány parametry jednotlivých modulů, které jsou do *XML* konfiguračního souboru mapovány z *INI* souborů. Ukázka částí těchto *INI* souborů byla již ukázaná pro konfiguraci jednotlivých komunikačních kanálů brány v sekci 2.3. Třída **DependencyInjector** (poskytovatel vkládání závislostí) po spuštění softwaru brány načte *XML* konfigurační soubory, ze kterých zjistí jaké moduly jsou povoleny a jaké mezi nimi existují vazby. Pro povolené moduly vytvoří instance příslušných tříd a vytvoří mezi nimi definované závislosti. Některé moduly, pro které to je vyžadováno, může poskytovatel vkládání závislosti spustit v samostatném vlákně.

Ukázka konfiguračního souboru je dostupná ve výpisu 2.4. Tento konfigurační soubor spouští software brány s jedním exportérem a s jedním manažerem zařízení. V ukázce lze vidět způsob vytváření závislostí mezi moduly. Příkladem může být nastavení závislosti mezi modulem `deviceManager` a moduly `distributor`, `commandDispatcher` a `deviceCache` s tím, že manažer zařízení zná zmíněné moduly a může s nimi pracovat.

```
<system>
  <factory>
    <instance name="main" class="BeeeOn::LoopRunner">
      <add name="runnables" ref="distributor" />
      <add name="loops" ref="gwsConnector" if-yes="${gws.enable}"/>
      <add name="runnables" ref="deviceManager" if-yes="${dm.enable}"/>
    </instance>
    <instance name="distributor" class="BeeeOn::Distributor">
      <add name="exporters" ref="gwsConnector" if-yes="${gws.enable}" />
    </instance>
    <instance name="commandDispatcher" class="BeeeOn::CommandDispatcher">
      <add name="handlers" ref="gwsConnector" if-yes="${gws.enable}"/>
      <add name="handlers" ref="deviceManager" if-yes="${dm.enable}"/>
    </instance>
```

```
    <instance name="gwsConnector" class="BeeeOn::GWSConnector">
      <set name="commandDispatcher" ref="commandDispatcher"/>
    </instance>
    <instance name="deviceCache" class="BeeeOn::DeviceCache">
      <set name="prepaired" list="" />
    </instance>
    <instance name="deviceManager" class="BeeeOn::DeviceManager">
      <set name="distributor" ref="distributor" />
      <set name="commandDispatcher" ref="commandDispatcher" />
      <set name="deviceCache" ref="deviceCache" />
    </instance>
  </factory>
</system>
```

Výpis 2.4: Ukázka konfiguračního souboru

Každá třída, která má být vytvářená poskytovatelem vkládání závislostí, se musí u něj registrovat. To se provádí pomocí speciálních maker. Ve výpisu 2.5 je ukázka registrace *MQTT* exportéru u poskytovatele vkládání závislostí. Jako první vždy musí být makro `BEEEON_OBJECT_BEGIN`, kterým se definuje jmenný prostor a název třídy. Dále je zde makro `BEEEON_OBJECT_CASTABLE`, které definuje nadřazenou třídu. To znamená, že třída **Exporter** je nadřazenou třídou třídy **MqttExporter** a poskytoval vkládání závislostí může s touto třídou pracovat jako s třídou **Exporter**. Makrem `BEEEON_OBJECT_PROPERTY` se definuje, jaké parametry je možné pro danou třídu konfigurovat. Pro každý parametr je v makru definovaná metoda, pomocí které se provede nastavení daného parametru. Název parametru se musí shodovat s názvem parametru v *XML* konfiguračním souboru. Posledním makrem musí být makro `BEEEON_OBJECT_END` se stejnými argumenty jako první makro.

```
BEEEON_OBJECT_BEGIN(BeeeOn, MqttExporter)
BEEEON_OBJECT_CASTABLE(Exporter)
BEEEON_OBJECT_PROPERTY("topic", &MqttExporter::setTopic)
BEEEON_OBJECT_PROPERTY("qos", &MqttExporter::setQos)
BEEEON_OBJECT_PROPERTY("formatter", &MqttExporter::setFormatter)
BEEEON_OBJECT_PROPERTY("mqttClient", &MqttExporter::setMqttClient)
BEEEON_OBJECT_END(BeeeOn, MqttExporter)
```

Výpis 2.5: Makro pro registraci třídy u poskytovatele vkládání závislostí

Kapitola 3

Komunikační technologie

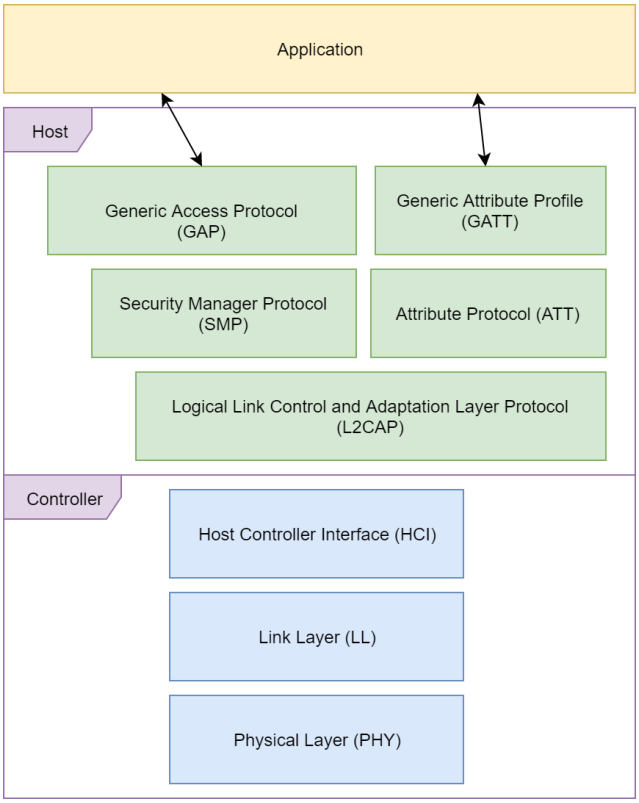
Kapitola obsahuje popis komunikačních technologií a protokolů, pomocí kterých komunikují analyzovaná zařízení v rámci této diplomové práce. Nejprve je popsaná komunikační technologie *Bluetooth Low Energy*, která je využita firmami *Revogi* a `Tabu Lumen` pro komunikaci s jejími zařízeními. Dále je rozebrán komunikační protokol *MQTT* využívající počítačovou síť. Tento protokol je využit ke komunikaci s multisenzorem *Sonoff SC*.

3.1 Bluetooth Low Enegry

Bluetooth Low Energy, také někdy označován jako *Bluetooth Smart*, je podmnožina standartu *Bluetooth*. Přestože se *Bluetooth Low Energy* a klasický *Bluetooth* do určité míry překrývají, *Bluetooth Low Energy* pracuje na trochu jiných principech. *Bluetooth Low Energy* je bezdrátový komunikační protokol, jenž je ve velké míře používán u zařízení pro chytrou domácnost. V dnešní době, pokud navrhujeme moderní mobilní platformou, která má komunikovat z okolním světem, je *Bluetooth Low Energy* jeden z nejjednodušších způsobů. Velkou výhodou této technologie je nízká spotřeba energie a cena, ale naopak je omezen vzdáleností oproti komunikační technologie *LoRa*.

Na obrázku 3.1 lze vidět architekturu softwarového vrstevnatého modelu *Bluetooth Low Energy*. Zásobník je rozdělen do dvou částí a to řadič a hostitel. Na nejnižší úrovni řadiče je fyzická vrstva (Physical Layer), která obsahuje analogové komunikační obvody, jejíž úkol je přenos dat skrz vzduch. O úroveň výše je linková vrstva, která využívá služeb fyzické vrstvy a je zodpovědná za odesílání propagačních zpráv, skenování sítě a vytváření/udržování spojení s jiným zařízením [4].

Následující sekce jsou zaměřeny na nejvýše situované vrstvy *Bluetooth Low Energy* zásobníku, které zajišťují inzerci přítomnosti zařízení v síti, navázaní spojení s jiným zařízením a přenos dat mezi zařízeními. Jedná se o vrstvy části hostitel.



Obrázek 3.1: Bluetooth Low Enegry zásobník

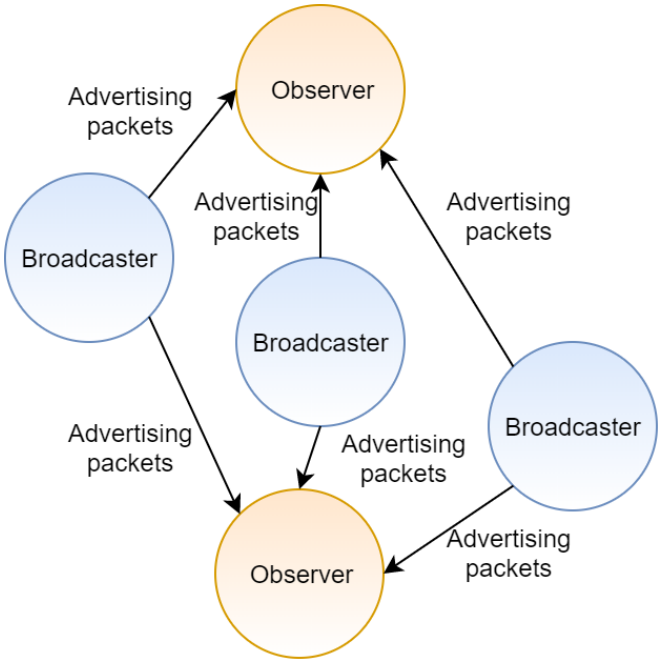
3.1.1 Generic Access Profile

Stavy a procedury této vrstvy jsou základním stavebním kamenem pro ostatní operace *Bluetooth Low Energy*. Tato vrstva zajišťuje objevování nových zařízení na síti, připojování se k dalším zařízením, sdílení dat a vytváření zabezpečených spojení. **GAP** je vrstva, která umožňuje, aby zařízení bylo viditelné ostatním zařízením a definuje jakým způsobem interagují dvě zařízení.

GAP [4, 7] definuje dvě dvojice rolí, které umožňují zařízením komunikovat. Zařízení může fungovat s více rolemi zároveň. První dvojice rolí je vysílatel/pozorovatel jenž umožňuje jednosměrnou komunikaci bez připojení. Tento způsob komunikace dovoluje jednomu zařízení odesílat data více zařízením najednou. Názorná ukázka způsobu komunikace mezi zařízeními je na obrázku 3.2, kde lze vidět, že zařízení v roli vysílatele odesílá zprávy, které jsou doručeny všem zařízením v roli pozorovatele.

- **Vysílatel (Broadcaster)** – zařízení v této roli periodicky odesílá propagační zprávy, které v sobě nesou konkrétní data. Příkladem může být teplotní senzor, který periodicky odesílá naměřená teplotní data.

- **Pozorovatel (Observer)** – tuto roli zastává zařízení, které sbírá a dále zpracovává data z propagační zprávy. V této roli se nachází *BeeeOn* brána, která sbírá data z různých senzorů a provádí nad nimi nějaký výpočet.

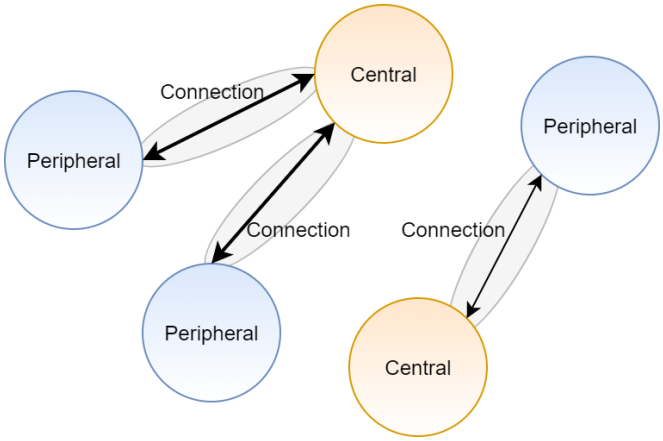


Obrázek 3.2: Komunikace mezi vysílateli a pozorovateli

Druhou dvojicí rolí je periferní/centrální zařízení. Tato dvojice rolí implementuje obousměrnou komunikaci orientovanou na připojení.

- **Periferní zařízení** – obvykle malá, zdrojově omezená zařízení, která nabízí určité služby. Může se jednat o teplotní senzor, dveřní/okenní kontakt a tak podobně.
- **Centrální zařízení** – jsou zařízení s větším výpočetním výkonem a pamětí. Tyto zařízení se mohou k periferním zařízením připojovat a využívat jejich služby. Jedno centrální zařízení může mít v jednom čase ustanovených více spojení.

Periferní zařízení periodicky odesílá propagační zprávy, aby se centrální zařízení dozvědělo o přítomnosti periferního zařízení v síti. Interval odesílání propagačních zpráv je optimalizován, aby nepotřeboval moc výpočetního výkonu a paměti a tím šetřil spotřebu energie zařízením. Druhým způsobem, jak se může centrální zařízení dozvědět o přítomnosti periferního zařízení je odeslání skenovacího požadavku (Scan Request) centrálním zařízením. Periferní zařízení po přijetí skenovacího požadavku ihned odpovídá skenovací odpovědí (Scan Response). Následně po nalezení periferního zařízení centrálním zařízením, může centrální zařízení vytvořit s periferním zařízením spojení a začít vyměňovat zprávy pomocí **GATT**. Periferní zařízení po ustanovení spojení přestane periodicky odesílat propagační zprávy o své přítomnosti na síti, protože může mít pouze jedno ustanovené spojení. Způsob komunikace mezi zařízeními je zobrazeno na obrázku 3.3, kde lze vidět, že centrální zařízení může ustanovit připojení s více periferními zařízeními současně, se kterými si v rámci spojení vyměňuje zprávy.



Obrázek 3.3: Komunikace mezi centrálními a periferními zařízeními

3.1.2 Generic Attribute Profile

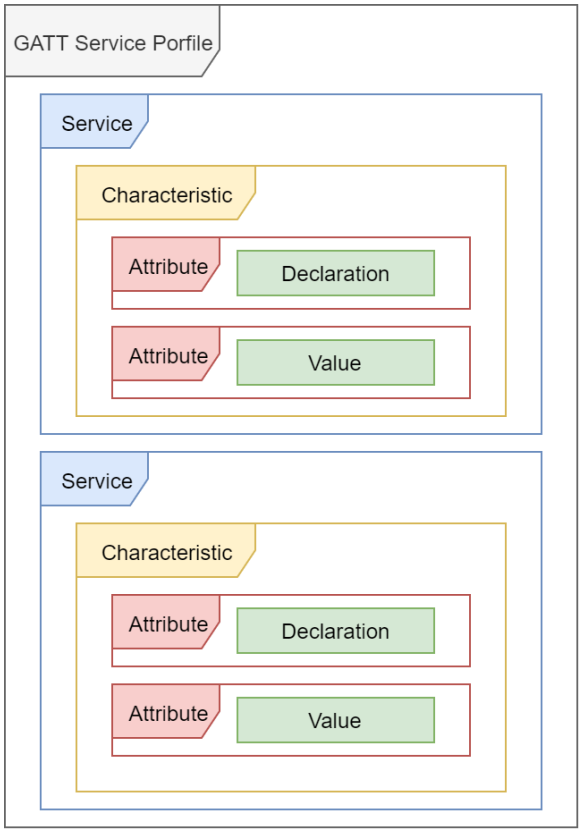
Generic Attribute Profile vrstva [4, 7] společně s vrstvou **Attribute Protocol** ustanovují jak jsou data na *Bluetooth Low Energy* zařízení organizovány a jak jsou vyměňovány v rámci spojení. **GATT** je využíván až tehdy, když je mezi zařízeními ustanoveno spojení. To znamená, že se již provedlo objevování zařízení na síti a vytvoření spojení s konkretním zařízením. Tyto akce jsou řízeny vrstvou **GAP**.

GATT hierarchicky organizuje atributy do služeb (services), vlastností (characteristics) a atributů (attributes). Kdy služba se skládá z několika vlastností a vlastnost se skládá z několika atributů. Ukázka hierarchie **GATT** profilu je na obrázku 3.4, který znázorňuje profil skládající se ze dvou služeb, kde každá služba má jednu vlastnost s dvěma atributy. Služba seskupuje vlastnosti, které mají podobné chování a mají dosáhnout určité funkcionality. Každá služba se od ostatních odlišuje svým identifikátorem, který se nazývá UUID[1]. Identifikátor může být 16-bitový v případě, že služba je veřejná, nebo 128-bitový, když je služba privátní (výrobcem definovaná).

Vlastnost je kontejnerem pro koncová uživatelská data. Každá vlastnost obsahuje minimálně dva atributy. Prvním z nich je *Declaration Attribute*, který obsahuje metadata pro druhý atribut *Value Attribute*, jenž obsahuje samotná data. Třetím volitelným atributem je *Descriptor Attribute*. Ten může obsahovat uživatelsky čitelný popis vlastnosti nebo může sloužit jako přepínač, který povoluje/zakazuje aktualizace iniciovány serverem. Pro každý atribut jsou definovaná oprávnění, která definují, zda je možné atribut číst, modifikovat nebo ani jedno. Stejně jako tomu bylo u služeb, vlastnosti jsou také identifikovány pomocí identifikátoru, který může být 16-bitový nebo 128-bitový.

Organizací *Bluetooth SIG* jsou předdefinovány 16-bitové UUID pro různé typy služeb a vlastností dostupné na této webové stránce https://www.bluetooth.com/specifications/gatt/. Jednotlivé služby, vlastnosti a atributy jsou spravovány **Attribute Protocol**, který data uchovává v jednoduché vyhledávací tabulce, kdy každý záznam je adresován pomocí 16-bitového *handle*.

[1] Universally unique identifier



Obrázek 3.4: Hierarchie GATT profilu

Komunikace mezi dvěma zařízeními je založena na principu server/klient. V roli **GATT** serveru je periferní zařízení, které obsahuje služby a vlastnosti a vyřizuje požadavky od **GATT** klienta. Klientem v komunikaci je centrální zařízení a tedy například mobil nebo právě *BeeeOn* brána. Po úspěšném vytvoření spojení mezi zařízeními, klient odesílá požadavek na server pro nalezení všech služeb a vlastností. Když klient zná nabízené služby a vlastnosti **GATT** serverem, může začít odesílat na něj požadavky pro manipulaci s atributy. Druhý způsob komunikace je iniciován serverem, který odesílá na klienta data v okamžiku, kdy nastane určitá událost.

3.2 Message Queuing Telemetry Transport

MQTT je lehký síťový protokol, který pro výměnu zpráv mezi zařízeními používá princip publikování a odebírání. *MQTT* je navržen tak, aby byl otevřený, jednoduše implementovatelný a umožňoval připojení tisíce lehkých klientů k jednomu serveru. Díky těchto vlastnosti je *MQTT* ideální pro použití v omezených sítích s malou šířkou pásma nebo pro zařízení s malou kapacitou paměti a omezenými výpočetními zdroji. Návrh *MQTT* minimalizuje požadavky na šířku pásma a zároveň se snaží zajistit spolehlivost doručení.

3.2.1 Komunikační model

MQTT protokol definuje dva typy entit v síti, a to zprostředkovatele zpráv (Broker) a řadu klientů. Broker zastává roli serveru, který přijímá veškeré zprávy od připojených klientů a

poté je směruje k relevantním klientům. Klient je jakékoliv zařízení, které může komunikovat s brokerem. Klient může být senzor v prostředí nebo *BeeeOn* brána, která zpracovává senzorová data. Komunikace mezi klienty a brokerem probíhá:

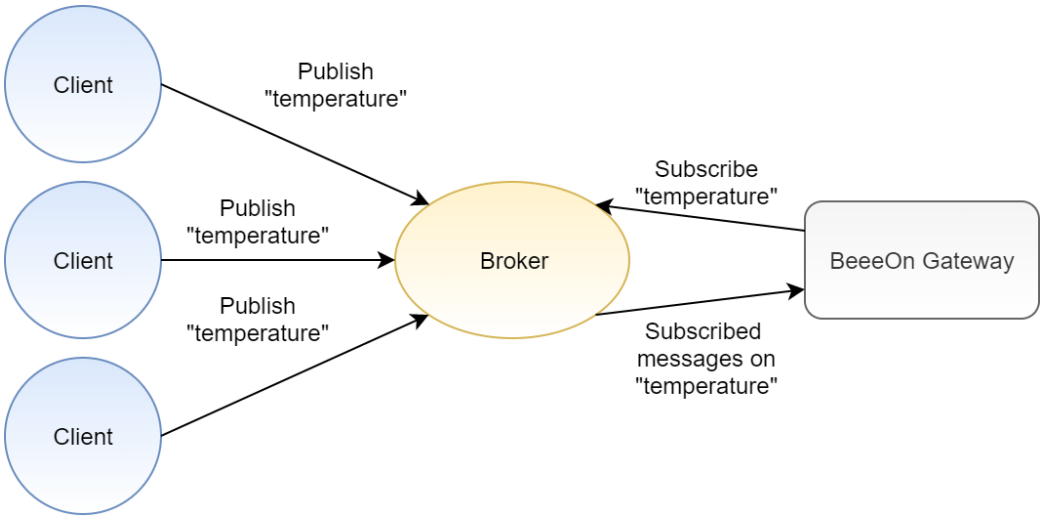
1. Klient se připojí k brokeru a může si u něj registrovat odběr jakéhokoliv tématu (topic). Toto připojení může být nešifrované TCP/IP připojení, nebo šifrované TLS připojení pro citlivé zprávy. Při připojování klienta k brokeru může klient definovat téma a zprávu, kterou má broker poslat na zadané téma, když se klient odpojí (last will).

2. Klient publikuje zprávy pod určitým tématem, zasíláním zpráv a témat na broker.

3. Broker poté přijaté zprávy směruje na klienty, kteří u něj mají registrovaný odběr na téma, na které byla zpráva publikována.

MQTT komunikační model publikování/odebírání umožňuje komunikaci jeden na jednoho, ale také jeden na více. Protokol nedefinuje formát obsahu zpráv a tedy je možné využít libovolného formátu například JSON, XML nebo CSV [9, 2].

Ukázka způsobu komunikace pomocí *MQTT* protokolu je dostupná na obrázku 3.5. Komunikace se účastní tři klienti, jeden broker a *BeeeOn* brána. Brána si po připojení k brokeru registruje odběr na téma `temperature`. Klienti v určitých okamžicích odesílají zprávy na téma `temperature` s naměřenou teplotou na broker. Broker po přijetí zprávy na téma `temperature` zjistí, kteří klienti mají registrovaný odběr na toto téma, v tomto případě se jedná pouze o *BeeeOn* brána, a těm ji přepošle.



Obrázek 3.5: Ukázka *MQTT* komunikace

3.2.2 MQTT téma

V *MQTT* je téma [9, 2] textový řetězec, který broker používá pro směrování zpráv. Téma je hierarchické a skládá se z jednoho nebo více úrovní, podobně jak tomu je u adresářové cesty v souborovém systému. Jednotlivé úrovně tématu jsou oddělovány oddělovacím znakem `/`. Příkladem hierarchického *MQTT* tématu může být `living_room/temperature`. *MQTT* témata podporují dva zástupné znaky:

- **Znak +** – tento zástupný znak odpovídá jakékoliv jedné úrovni tématu. To znamená, když existují dvě témata `ground_floor/living_room/temperature` a `ground_floor/living_room/humidity`, potom téma `ground_floor/living_room/+` odpovídá oběma definovaným tématům.
- **Znak #** – tento zástupný znak odpovídá jakékoliv sekvenci úrovní tématu. Když existují stejná témata jako v předchozím případě, potom téma `ground_floor/#` odpovídá opět oběma definovaným tématům. To však neplatí pro téma `ground_floor/+`, které neodpovídá žádnému tématu.

3.2.3 Kvalita služeb

Protokol *MQTT* podporuje tři úrovně kvality služeb [9, 6]. Kvalita služeb udává způsob jakým broker dodává zprávy příjemcům. Úroveň kvality služeb se definuje při odebírání určitého tématu a při publikování zprávy na určité téma. Kvalita služeb definuje proces výměny zpráv mezi dvěma entitami.

- **Úroveň 0 (Fire and forget)** – v případě této úrovně kvality služeb klient i broker pouze odešle zprávu a dále se o nic nestará.
- **Úroveň 1 (Delivered at least once)** – když odesílatel odesílá zprávu s touto úrovní kvality služeb, tak po odeslání zprávy čeká až příjemce odpoví, že zprávu přijal. Broker přijatou zprávu dále může přeposlat s kvalitou služeb stejnou nebo nižší v případě, že odběratel nepodporuje kvalitu služeb úrovně 1. Chování brokeru se může lišit s implementaci a potvrzovací zprávu může odeslat ihned, jak příjme zprávu nebo až v okamžiku, kdy všichni odběratelé potvrdili přijetí přeposlané zprávy. Tato úroveň kvality služeb zaručuje, že zpráva je odběrateli doručena alespoň jednou. Jedna zpráva, ale může být příjemci doručena vícekrát.
- **Úroveň 2 (Delivered exactly once)** – na rozdíl od úrovně 1 tato úroveň kvality služeb zaručuje, že jedna zpráva je doručena příjemci právě jednou. Odesilatel při odeslání zprávy nejprve čeká, až příjemce potvrdí přijetí zprávy. V případě, že odesílatel neobdrží potvrzení přijetí zprávy, tak odesílá zprávu znovu s příznakem duplicity. Po přijetí potvrzení od příjemce odesílatel odesílá zprávu, která příjemci říká, že si může odstranit veškeré uložené stavy spojené s odeslanou zprávou. Příjemce po zpracování této zprávy odesílá odesílateli potvrzení. Přijetím potvrzeni odesílatelem komunikace končí.

Klienti, při registraci odběru u brokeru, definují úroveň kvality služeb, pomocí, které chtějí dostávat zprávy. V takovém případě, když broker obdrží zprávu na určité téma s kvalitou služeb úrovně 1, ale odběratel má registrovaný odběr na stejné téma s kvalitou služeb úrovně 0, potom broker danému odběrateli odesílá zprávu s kvalitou služeb úrovně 0.

Kapitola 4

Bezdrátová zařízení

V této kapitole jsou popsány jednotlivá zařízení, pro které je v rámci této práce vytvořena podpora do softwaru *BeeeOn* brány. Tato sada zařízení byla vybrána z důvodů rozšíření množiny sbíraných a ovládaných fyzikálních veličin prostředí *BeeeOn* brány, ale také z důvodu dobré dostupnosti na evropském trhu. Pomocí vybraných zařízení je možné ovládat osvětlení v domácnosti prostřednictvím žárovek, snížit spotřebu elektrické energie, kterou měří inteligentní zásuvka a také monitorovat kvalitu prostředí pomocí multisenzoru. Celkově byli analyzovány zařízení od čtyř různých výrobců:

1. **Revogi** – analyzovaná zařízení od výrobce *Revogi* jsou inteligentní žárovka, zásuvka a svíčka. Tento výrobce pro komunikaci se zmíněnými zařízeními používá komunikační bezdrátovou technologii *Bluetooth Low Energy*. Mimo jiné *Revogi* také vyrábí inteligentní zařízení chytré domácnosti komunikující bezdrátovou technologií *WiFi*.

2. **Tabu Lumen** – zařízení od výrobce *Tabu Lumen* stejně jako zařízení od *Revogi* používají pro komunikaci bezdrátovou technologii *Bluetooth Low Energy*. Od tohoto výrobce byla analyzována chytrá RGB žárovka.

3. **Sonoff** – výrobce *Sonoff* používá pro komunikaci se svými zařízeními bezdrátové technologie *WiFi* a *ZigBee*. V této práci byl analyzován chytrý multi-senzor, od tohoto výrobce, umožňující měřit kvalitu ovzduší. Se senzorem se komunikuje pomocí technologie *WiFi*.

4. **HomeMatic** – poslední zmíněný výrobce *HomeMatic* vyrábí zařízení pro inteligentní domácnost, se kterými je možné komunikovat pomocí bezdrátové technologie *WiFi*. Většina zařízení od toho výrobce však používá pro komunikaci proprietární bezdrátovou technologii využívající radiového kmitočtu s frekvencí 868 MHz. Analyzovanými zařízeními od tohoto výrobce jsou inteligentní termostatická hlavice na radiátor, inteligentní zásuvka a inteligentní magnetický dveřní, okenní kontakt. Se všemi těmito zařízeními se komunikuje pomocí rádia s frekvencí 868 MHz.

Způsob ovládání *Bluetooth Low Energy* zařízení jsem zjišťoval zachytáváním komunikace originální mobilní aplikace pro daná zařízení s konkretními zařízeními. Pomocí mobilní aplikace jsem na zařízení odesílal různé příkazy a poté jsem ze zachycené komunikace analyzoval a dekódoval vyměňované zprávy. Tímto způsobem jsem zjistil jak ovládat *Revogi* a *Tabu Lumen* zařízení. Analýzu komunikace *Sonoff* senzoru jsem začal stejným způsobem jako analýzu *Bluetooth Low Energy*, kde jsem zjistil, že senzor nemá žádné aplikační rozhraní pro získávání naměřených dat, a proto byl firmware senzoru nahrazen vlastním firmwarem

disponující *MQTT* rozhraním. V rámci hledání způsobu jak komunikovat s *HomeMatic* zařízeními byl nalezen systém *FHEM*, který umožňuje pomocí *USB* rádiového modulu zařízení ovládat. Pro propojení *FHEM* systému s *BeeeOn* bránou jsem nastudoval aplikační rozhraní *FHEM* systému, které je popsané dále v kapitole.

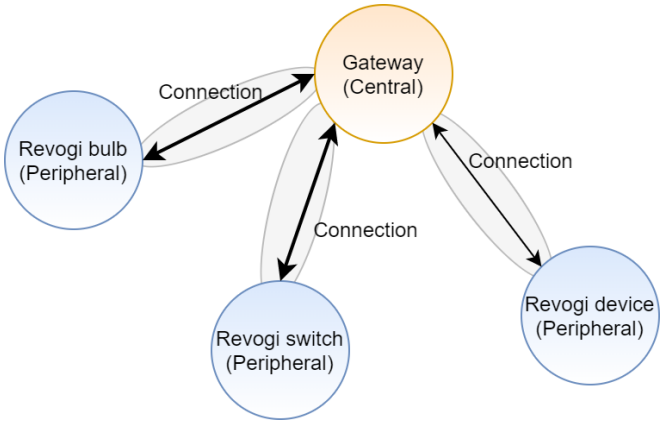
Ke každému zařízení je v dalších sekcích dostupný popis chování, popis jeho funkcí a popis komunikace. V rámci popisu komunikace se zařízením jsou dostupné ukázky zpráv, pomocí kterých se získávají data ze zařízení a případně pomocí kterých se nastavuje stav jejích aktorových modulů na danou hodnotu.

4.1 Revogi

Software *BeeeOn* brány je rozšiřován o podporu inteligentní žárovky, zásuvky a svíčky vyráběné firmou *Revogi*. S těmito zařízeními se komunikuje pomocí bezdrátového protokolu *Bluetooth Low Energy*, jehož fungování je blíže popsáno v sekci 3.1. Aby dané zařízení mohlo s těmito zařízeními komunikovat je nutné, aby zařízení disponovalo *Bluetooth Low Energy* modulem, který může být v zařízení již zabudován, nebo je jej možné dodat ve formě USB modulu.

4.1.1 Komunikace se zařízeními

Revogi inteligentní zařízení jsou, vzhledem k rozdělení rolí v rámci *Bluetooth Low Energy*, periferními zařízeními. To znamená, že pro řízení *Revogi* zařízení se *BeeeOn* brána, která je v roli centrálního zařízení, vždy musí k zařízení připojit. Po úspěšném připojení brány k zařízení může brána začít využívat jeho nabízené **GATT** vlastnosti (například získání aktuálního stavu zařízení nebo změna stavu zařízení). Na obrázku 4.1 je znázorněna komunikace *BeeeOn* brány s *Revogi* zařízeními.



Obrázek 4.1: Komunikace brány s Revogi zařízeními

Revogi nepoužívá pro identifikaci předdefinované **GATT** vlastnosti s UUID `2a24`[1] určenou pro uchování názvu modelu. Tato vlastnost je však na každém *Revogi* zařízení dostupná, ale vždy obsahuje řetězec `Model Number`. Pro identifikaci zařízení *Revogi* používá vlastní privátní **GATT** vlastnost s UUID `0000fff6-0000-1000-8000-00805f9b34fb`. Obsah této vlastnosti je specifický pro každý typ zařízení.

[1] https://www.bluetooth.com/wp-content/uploads/Sitecore-Media-Library/Gatt/Xml/Characteristics/org.bluetooth.characteristic.model_number_string.xml

Formát zpráv vyměňovaných s *Revogi* zařízeními je zobrazen v tabulce 4.1. Zpráva se skládá z hlavičky, zprávy a patičky. Hlavička je čtyři bajty dlouhá a její obsah identifikuje o jakou zprávu se jedná. Délka obsahu zprávy se může lišit podle toho jaké zařízení zprávu odesílá nebo pro jaké zařízení je zpráva určena. Formát této části je popsán pro každé zařízení zvlášť. Poslední části zprávy je patička, která se skládá ze tří bajtů. První bajt obsahuje kontrolní součet, který slouží k ověření korektnosti zprávy a poslední dva bajty nabývají vždy hodnot `0xff 0xff`.

Hlavička	Obsah	Patička
4 B	X B	3 B

Tabulka 4.1: Formát *Revogi* zprávy

Získání aktuálního stavu *Revogi* zařízení probíhá povolením příjímání notifikací z privátní **GATT** vlastnosti s UUID `0000fff4-0000-1000-8000-00805f9b34fb` a následným zapsáním zprávy 4.2 na privátní **GATT** vlastnost s UUID `0000fff3-0000-1000-8000-00-805f-9b34fb`. Zařízení po přijetí této zprávy vygeneruje a odešle notifikaci na zmíněné vlastnosti, která obsahuje aktuální nastavení zařízení. Typ této zprávy je možné identifikovat pomocí hlavičky zprávy, která nabývá hodnot: `0x0f 0x0e 0x04 0x00`. Obsah zprávy obsahuje nastavení jednotlivých modulů zařízení.

Hlavička	Obsah	Patička
`0x0f 0x05 0x04 0x00`	`0x00 0x00`	`0x05 0xff 0xff`

Tabulka 4.2: Zpráva vynucující odeslání zprávy s aktuálním nastavením *Revogi* zařízení

Pro nastavení stavu *Revogi* zařízení slouží privátní **GATT** vlastnost s UUID `0000ff-f3-0000-1000-8000-00805f9b34fb`. Obsah zprávy obsahuje jednotlivé parametry příkazu, které reprezentují nastavovanou hodnotu daného modulu zařízení.

4.1.2 Revogi chytrá žárovka

Prvním zařízením od výrobce *Revogi*, pro které je vytvářena podpora do *BeeeOn* brány je chytrá žárovka. Žárovka disponuje čtyřmi aktorovými moduly. První aktorový modul slouží k změně stavu žárovky ve smyslu zapnuta/vypnuta. Druhý modul umožňuje měnit jas žárovky. Žárovka se může nacházet ve dvou stavech, a to v bílém módu a v barevném módu. Žárovka v bílém módu umožňuje měnit teplotu bílé barvy. Pro nastavení teploty bílé barvy slouží třetí aktorový modul. V rámci barevného módu žárovka může svítit všemi možnými barvami a pro změnu barvy je používán čtvrtý aktorový modul. Žárovka přechází z bílého módu do barveného módu, když je žárovkou přijat požadavek na svícení danou barvou. Podobně žárovka přechází z barevného módu do bílého, když žárovka příjme požadavek na to, aby svítila danou teplotou bílé barvy. Žárovku je možné vidět níže na obrázku 4.2. Žárovka je identifikována pomocí řetězce `Delite-1748`, jenž se nachází v privátní **GATT** vlastnosti, která je určena pro uchování typu *Revogi* zařízení.

Obrázek 4.2: Revogi chytrá žárovka

Získávaní aktuálního nastavení žárovky

Obsah zprávy odeslané žárovkou obsahující aktuální nastavení žárovky má 11 bajtů a její formát je naznačen v tabulce 4.3.

RGB	Jas a stav	Teplota bílé barvy	Mód	Další nastaven
3 B	1B	1 B	1 B	5 B

Tabulka 4.3: Formát obsahu zprávy odesílanou *Revogi* žárovkou

První tři bajty obsahu nesou hodnotu aktuálně nastavené barvy, kterou žárovka svítí, v pořadí červená, zelená a modrá. Čtvrtý bajt obsahuje aktuální nastavený jas a stav žárovky. Žárovka podporuje 200 úrovní jasu. Když je hodnota tohoto bajtu v intervalu 0 až 200, tak je žárovka zapnuta a hodnota udává úroveň jasu žárovky. V případě, kdy je hodnota větší než 200 tak je žárovka vypnuta a jas je nula. Další bajt udává aktuálně nastavenou teplotu bílé barvy. Obdobně jak tomu bylo u jasu, žárovka podporuje 200 úrovní teploty bílé barvy od 2700 do 6500 Kelvinů. Platné hodnoty tohoto bajtu spadají do intervalu 0 až 200. V desátém bajtu je dostupná informace o módu žárovky, jestliže se žárovka nachází v barevném módu bajt obsahuje hodnotu `0x00` v opačném případě když je žárovka v bílém módu bajt obsahuje hodnotu `0x01`. Poslední 5 bajtů obsahu zprávy obsahuje další nastavení žárovky, které je ale v rámci ovládání žárovky nepodstatné.

Ovládání žárovky

Obsah zprávy, odesílané na žárovku za účelem změny stavu žárovky, má 10 bajtů. Poslední čtyři bajty obsahu jsou určeny pro další nastavení žárovky, které se zde nepožívají a implicitně jsou nulové. Hlavička této zprávy musí nabývat hodnot `0x0f 0x0d 0x03 0x00`.

Obsah zprávy pro změnu stavu žárovky na zapnuta/vypnuta je ukázaná v tabulce 4.4. Podle toho v jakém módu se žárovka právě nachází se mění obsah této části. Rozdíl je v posledním bajtu příkazu. Pro zapnutí žárovky `0xNN` je nahrazeno za hodnotu `0xfe` a pro vypnutí žárovky je v tomto bajtu hodnota `0xff`. Aby žárovka zprávu přijala je nutné korektně vypočítat kontrolní součet a přidat jej do patičky. Výpočet kontrolního součtu, pro příkaz na změnu stavu žárovky, je v posledním sloupci tabulky.

Mód	Obsah	Výpočet kontrolního součtu
Bílý mód	`0x00 0x00 0x00 0xNN 0x00 0x01`	`0x04 - (0xff - 0xNN)`
Barevný mód	`0x00 0x00 0x00 0xNN 0x00 0x00`	`0x03 - (0xff - 0xNN)`

Tabulka 4.4: Příkaz pro vypnutí/zapnutí žárovky

Pro změnu jasu žárovky je obsah zprávy zobrazen v tabulce 4.5. Stejně jako v předchozím případě se příkaz mění podle toho v jakém módu se žárovka nachází. V případě, kdy je žárovka v bílém módu, tak bajt `0xNN` nese hodnotu úrovně jasu (hodnota v intervalu 0-200) a bajt `0xYY` nese hodnotu aktuální teploty bílé barvy (hodnota v intervalu 0-200). Když je žárovka v barveném módu je nutné s úrovní jasu (bajt `0xNN`) také zadat aktuální barvu žárovky, která je zadaná v bajtech `0xRR` (červená složka), `0xGG` (zelená složka) a `0xBB` (modrá složka). Výpočet kontrolního součtu v případě, kdy se žárovka nachází v barevném módu lze vidět v posledním sloupci, kde bajt `0xUU` nese hodnotu barevné složky obsažené v příkazu, která je největší. Ve zbývajících bajtech `0xVV` a `0xWW` jsou ostatní barevné složky.

Mód	Obsah	Výpočet kontrolního součtu
Bílý mód	`0xfe 0xf0 0xdc 0xNN 0xYY 0x01`	`0xNN + 0xYY - 0x0131`
Barevný mód	`0xRR 0xGG 0xBB 0xNN 0x00 0x00`	`(0xcb - (0xff - 0xUU) + 0xVV + 0xWW) - 0xc8 + 0xNN`

Tabulka 4.5: Příkaz pro změnu jasu žárovky

Nastavení teploty bílé barvy se provádí odesláním zprávy s obsahem, tak jak je ukázáno v prvním řádku tabulky 4.6. V bajtu `0xNN` je zadána nastavovaná teplota bílé barvy (hodnota v intervalu 0-200). Pro změnu barvy žárovky je zobrazen příkaz v druhém řádku tabulky. Bajty `0xRR`, `0xGG` a `0xBB` nesou jednotlivé složky nastavované barvy v pořadí červená složka, zelená složka a modrá složka. Kontrolní součet pří nastavování barvy žárovky je vypočítáván podobně jako u nastavování jasu. Bajt `0xUU` nese hodnotu největší barevné složky a bajty `0xVV` a `0xWW` obsahují ostatní barevné složky.

Mód	Obsah	Výpočet kontrolního součtu
Bílý mód	`0xfc 0xfc 0xfc 0xc8 0xNN 0x01`	`0x89 - (0xc8 - 0xNN)`
Barevný mód	`0xRR 0xGG 0xBB 0xc8 0x00 0x00`	`0xcb - (0xff - 0xUU) + 0xVV + 0xWW`

Tabulka 4.6: Příkaz pro změnu barvy/teploty bílé žárovky

4.1.3 Revogi chytrá svíčka

Jako druhé zařízení od výrobce *Revogi*, které bylo analyzováno v rámci této práce, je chytrá svíčka. Toto zařízení disponuje třemi akčními moduly, které umožňují měnit stav svíčky, nastavit jas a nastavit barvu, kterou svíčka svítí. Chytrou svíčku lze vidět na obrázku 4.3. Chytrá svíčka na rozdíl od chytré žárovky je napájená pomocí čtyř baterií typu AA. Svíčku lze identifikovat pomocí řetězců `Delite-ED33` a `Delite-0870`, které se nacházejí v privátní **GATT** vlastnosti určenou pro uchování typu *Revogi* zařízení.

Obrázek 4.3: Revogi chytrá svíčka

Získávaní aktuálního nastavení svíčky

Zpráva odesílaná chytrou svíčkou má 18 bajtů. To znamená, že obsah zprávy obsahující aktuální nastavení svíčky má 11 bajtů a její formát je možné vidět v tabulce 4.7.

RGB	Jas a stav	Další nastavení
3 B	1B	7 B

Tabulka 4.7: Formát obsahu zprávy odesílanou *Revogi* svíčkou

Část obsahu zprávy **RGB** je délky 3 bajty a v jednotlivých bajtech jsou barevné složky RGB v pořadí červená, zelená a modrá. Aktuální jas a stav svíčky je dostupný v části obsahu zprávy **Jas a stav**, která je délky jeden bajt. Pokud tento bajt je větší než hodnota 200, tak je svíčka vypnutá a její jas je nulový. V opačném případě je svíčka zapnutá a bajt obsahuje aktuální jas svíčky. Svíčka, tedy stejně jako žárovka, podporuje 200 úrovní jasu.

Ovládání svíčky

Zpráva, pomocí které se ovládá svíčka, má 17 bajtů, což znamená, že obsah zprávy je délky 10 bajtů. Pro změnu stavu, jasu a barvy svíčky slouží první čtyři bajty obsahu zprávy. Zbylých šest bajtů slouží k dalším nastavením svíčky, které jsou v rámci vytváření podpory svíčky do *BeeeOn* brány nepodstatné a implicitně jsou nulové. Hlavička této zprávy musí nabývat hodnot `0x0f 0x0d 0x03 0x00`.

Pro jednotlivé akce je obsah zprávy a výpočet kontrolního součtu zobrazen v tabulce 4.8. Pro zapnutí/vypnutí svíčky je nutné nastavit pouze jeden bajt a ostatní jsou nulové. Pro zapnutí svíčky je `0xNN` nahrazen za hodnotu `0xfe` a pro vypnutí `0xff`. Při nastavování jasu svíčky je nutné do obsahu zprávy zadat aktuální barvu, kterou svíčka svítí a požadovanou úroveň jasu. Aktuální barva svíčky je zadaná po složkách v bajtech `0xRR`, `0xGG` a `0xBB` v pořadí červená, zelená a modrá. Požadovaná úroveň jasu (hodnota v intervalu 0-200) je nahrazena za bajt `0xNN`. Změnu barvy svíčky lze dosáhnout odesláním zprávy, která obsahuje jednotlivé složky nastavované barvy v bajtech `0xRR`, `0xGG` a `0xBB`. Výpočet kontrolního součtu, při nastavování jasu a barvy, je popsán v druhém sloupci tabulky, kde bajt `0xUU` je největší barevná složka obsažena v obsahu zprávy a bajty `0xVV` a `0xWW` jsou ostatní barevné složky.

Akce	Obsah	Výpočet kontrolního součtu
Změna stavu	`0x00 0x00 0x00 0xNN`	`0x03 - (0xff - 0xNN)`
Změna jasu	`0xRR 0xGG 0xBB 0xNN`	`(0xcb - (0xff - 0xUU) + 0xVV + 0xWW) - 0xc8 + 0xNN`
Změna barvy	`0xRR 0xGG 0xBB 0xc8`	`0xcb - (0xff - 0xUU) + 0xVV + 0xWW`

Tabulka 4.8: Příkaz pro změnu nastavení Revogi svíčky

4.1.4 Revogi chytrá zásuvka

Posledním zařízením od výrobce *Revogi*, pro které je vytvářena podpora, je chytrá zásuvka. Toto zařízení se skládá z jednoho aktorového modulu a čtyř senzorových modulů. Aktorový modul slouží k vypínání a zapínání zásuvky. Senzorové moduly umožňují získávat informace o elektrické síti, do které je zásuvka připojena, a o připojeném spotřebiči. Mezi informace sbírané o elektrické síti patří: elektrické napětí, elektrický proud a frekvence. Zásuvka také měří spotřebovanou energii připojeným spotřebičem. Chytrou zásuvku je možné vidět na obrázku 4.4. Zásuvka je identifikovatelná pomocí privátní **GATT** vlastnosti, určenou pro uložení typu *Revogi* zařízení, která v případě zásuvky obsahuje řetězec `MeterPlug-F19F`.

Obrázek 4.4: Revogi chytrá zásuvka

Získávaní dat a aktuálního nastavení zásuvky

Naměřená data a aktuální stav zásuvky odesílá zásuvka na vyžádání ve zprávě, která má 19 bajtů. Obsah zprávy je 12 bajtů dlouhý a formát obsahu je zobrazen v tabulce 4.9.

Stav	Další nastavení	Příkon	Napětí	Proud	Frekvence	Další nastavení
1 B	3 B	2 B	1 B	2 B	1 B	2 B

Tabulka 4.9: Formát obsahu zprávy odesílanou *Revogi* zásuvkou

V prvním bajtu obsahu je aktuální stav zásuvky. Hodnota `0x00` znamená, že je zásuvka vypnuta a hodnota `0x01` reprezentuje stav zapnuta. V části **Příkon** se nachází naměřená hodnota spotřebované elektrické energie spotřebičem v jednotkách miliwatt. V následujícím bajtu se nachází naměřené napětí elektrické sítě, které je ve voltech. Část obsahu zprávy **Proud** obsahuje ve dvou bajtech naměřený elektrický proud sítě. Hodnota proudu je ve

zprávě v jednotkách miliamepéry. Poslední měřenou fyzikální veličinou je frekvence sítě, která se nachází v části obsahu zprávy **Frekvence** a tato hodnota je v jednotkách hertz.

Ovládání zásuvky

Změna stavu zásuvky se provádí odesláním zprávy délky 10 bajtů. Hlavička této zprávy musí nabývat hodnot `0x0f 0x06 0x03 0x00`. Obsah zprávy je dostupný v tabulce 4.10. Pro vypnutí zásuvky musí první bajt obsahu nabývat hodnoty `0x00` a pro zapnutí zásuvky musí tento bajt nabývat hodnoty `0x01`. Výpočet kontrolního součtu, který je doplněn do patičky zprávy, je v druhém sloupci tabulky.

Obsah	Výpočet kontrolního součtu
`0xNN 0x00 0x00`	`0xNN + 0x04`

Tabulka 4.10: Příkaz pro změnu stavu Revogi zásuvky

4.2 Tabu Lumen

Od výrobce *Tabu Lumen* je vytvářena podpora pro chytrou žárovku. Tento výrobce použil pro komunikaci s žárovkou stejnou technologii *Bluetooth Low Enegry* jako výrobce *Revogi*. Bližší popis této technologie je dostupný v sekci 3.1.

4.2.1 Tabu Lumen chytrá žárovka

Jediným zařízením od výrobce *Tabu Lumen*, pro které je vytvářena podpora do *BeeeOn* brány, je chytrá žárovka. Žárovka disponuje třemi aktorovými moduly. První aktorový modul umožňuje měnit stav žárovky (zapnuta/vypnuta). Další aktorový modul slouží k nastavení jasu žárovky a poslední aktorový modul umožňuje měnit barvu, kterou žárovka svítí. Fyzická podoba žárovky je dostupná na obrázku 4.5. Na rozdíl od *Revogi* zařízení je možné tuto žárovku identifikovat na základě standardní **GATT** vlastnosti s UUID `2a24` určenou pro název modelu. *Tabu Lumen* chytrá žárovka má v této **GATT** vlastnosti řetězec `BG521`, který je pro tento typ zařízení unikátní.

Obrázek 4.5: Tabu Lumen chytrá žárovka

Ovládání žárovky

Z pohledu systému komunikace pomocí *Bluetooth Low Energy* je žárovka v roli periferního zařízení. Pro ovládání žárovky slouží privátní **GATT** vlastnost s UUID `0000fff1-00-00-1000-8000-00805f9b34fb`. Změna nastavení žárovky probíhá vždy zapsáním 20 bajtů na tuto **GATT** vlastnost. První bajt zprávy obsahuje kód příkazu a zbytek zprávy slouží k předání parametrů příkazu. Aby žárovka odeslanou zprávu korektně přijala, musí být zpráva správně zašifrována. Zpráva se šifruje pomocí dvou invariantních klíčů, které jsou veřejně dostupné na internetu[2]. Dvojice klíčů a algoritmus šifrovaní je dostupný v příloze C. Vstupem algoritmu je pole dvaceti hodnot reprezentující odesílanou zprávu, která ještě nemá specifikovaný příkaz, což znamená, že první bajt musí být nulový. Příkaz se musí specifikovat až po zašifrovaní zprávy.

Při každém připojení k žárovce se musí připojené centrální zařízení autorizovat. Autorizace probíhá odesláním zprávy, jejíž první bajt má hodnotu `0x08` (příkaz zprávy) a druhý až sedmý bajt nabývá hodnoty `0x55`. Ostatních třináct bajtů je nulových. Když se tato zpráva neodešle do určitého časového intervalu nebo je odeslána jiná první zpráva na žárovku, tak se žárovka automaticky od připojeného zařízení odpojí.

Vypnutí žárovky lze docílit odesláním zprávy s příkazem `0x00` a parametry, které jsou nulové, to znamená, že zbytek zprávy se skládá z nulových bajtů. Pro zapnutí žárovky, změnu jasu a změnu barvy, slouží příkaz s hodnotou `0x01`. V druhém až čtvrtém bajtu zprávy musí být hodnoty jednotlivých složek barvy, kterou má žárovka svítit, v pořadí červená, zelená a modrá. Další bajty zprávy jsou určeny pro další nastavení žárovky a implicitně jsou nulové.

Žárovka samotná neumí měnit svůj jas. Změna jasu je prováděna na základě největší barevné složky. V případě, že největší barevná složka je hodnoty `0xff`, tak to znamená, že jas žárovky je maximální (100 %). Při změně jasu, je nutné znát aktuálně nastavenou barvu a výpočet nově nastavené barvy, která reflektuje nastavovaný jas, se provede:

1. **Normalizace barevných složek** – z aktuálně nastavené barvy žárovky se vypočte aktuální jas, což se provede nalezením největší barevné složky, která je vydělena maximální hodnotu (`0xff`). Získáním aktuálního jasu je možné jednotlivé barevné složky vynásobit rozdílem aktuálního jasu a maximálního jasu. Tím dostaneme všechny barevné složky při maximálním jasu.

2. **Aplikace jasu na barevné složky** – výsledná nastavovaná barva se získá aplikováním nastavovaného jasu na získané barevné složky. To se provede vynásobením všech barevných složek nastavovaným jasem.

4.3 Sonoff

Bylo analyzováno také zařízení od společnosti *Sonoff*. Jedná se o chytrý multisenzor s názvem *Sonoff SC*. Multisenzor nabízí monitorovaní kvality prostředí a to měřením teploty, vlhkosti, prašnosti, hlučnosti a světelnosti. Tento výrobce použil pro komunikaci se zařízením technologii *WiFi*. Výrobcem naprogramovaný multisenzor *Sonoff SC* neobsahoval žádné aplikační rozhraní, pomocí kterého by z něj bylo možné sbírat data. Multisenzor se po spuštění připojí na předem definovanou adresu serveru a pomocí šifrované *HTTP* komunikace si multisenzor se serverem vyměňují data. Z těchto důvodů byl originální firmware nahrazen za jiný, který pro sdílení naměřených dat multisenzorem používá *MQTT* protokol.

[2] https://github.com/sandeepmistry/node-lumen

4.3.1 Multisenzor Sonoff SC

Jak již bylo výše napsáno, od výrobce *Sonoff* je analyzován multisenzor *Sonoff SC*. Tento multisenzor obsahuje pět senzorových modulů, které dokážou monitorovat kvalitu ovzduší. *Sonoff SC* dokáže měřit teplotu, vlhkost, prašnost, hlučnost a světelnost v prostředí, ve kterém se nachází. Aby bylo možné získávat data z multisenzoru byl originální firmware multisenzoru nahrazen novým, který využívá síťový komunikační protokol *MQTT* pro sdílení naměřených dat multisenzorem. Nový firmware multisenzoru vychází z již existujícího firmwaru[3] , který byl upraven pro potřeby systému *BeeeOn*. Multisenzor je možné vidět na obrázku 4.6.

Nastavení multisenzoru

Multisenzor využívá pro komunikaci bezdrátovou technologii *WiFi*. Aby mohl multisenzor začít komunikovat s okolím, je nutné jej korektně nastavit. Při prvním spuštění se multisenzor začne chovat jako přístupový bod. Je potřeba se na něj připojit a poté je možné multisenzoru pomocí webového prohlížeče předat přihlašovací údaje k směrovači, ke kterému se má multisenzor připojit. Po úspěšném připojení multisenzoru k lokální síti je ještě potřeba zadat doménové jméno nebo IP adresu *MQTT* brokeru, na který má multisenzor posílat naměřená data.

Obrázek 4.6: Multisenzor Sonoff SC

Hardware multisenzoru

Multisenzor *Sonoff SC* je osazen dvěma mikrokontroléry a to **ESP8266** a **ATMega328P**. Mikrokontrolér **ESP8266** disponuje *WiFi* modulem, který zajišťuje komunikaci multisenzoru s okolím. Druhý mikrokontrolér **ATMega328P** je součástí multisenzoru, protože mikrokontrolér **ESP8266** obsahuje pouze jeden AD převodník, přičemž multisenzor se skládá ze čtyř senzorů, které jsou z většiny analogové. Samotný mikrokontrolér **ESP8266** by nedokázal v rozumném čase sbírat data ze senzorů. To znamená, že mikrokontrolér **ATMega328P** řídí sběr dat ze senzorů. Komunikace mezi mikrokontroléry je realizována pomocí sériové komunikační linky. Schéma zapojení jednotlivých senzorů a mikrokontrolerů je dostupné v příloze B.

[3] https://github.com/xoseperez/sonoffsc

Senzorová výbava senzoru se skládá ze:

- senzoru **DHT11**, který zajišťuje měření teploty a vlhkosti vzduchu,
- fotorezistoru **GM55**, jenž zajišťuje měření světelnosti,
- elektretového mikrofonu s obvodem zesilovače měřící hlučnost a
- senzoru **GP2Y1010AU0F**, který zajišťuje měření množství prachu ve vzduchu.

Programování multisenzoru

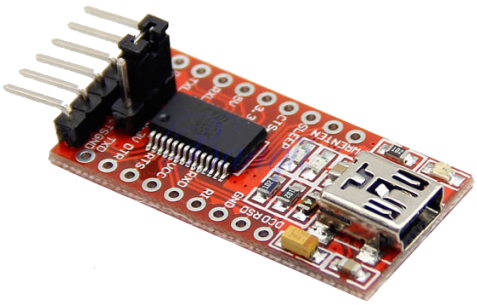
Programování multisenzoru je možné pomocí vývojového prostředí *Visual Studio Code* s modulem *PlatformIO*[4], který rozšiřuje vývojové prostředí o možnost vývoje, nahraní a ladění programů pro vestavěná zařízení.

Zmíněné vývojové prostředí umožňuje programovat dvojici mikrokontrolerů *ESP8266* a *ATMega328P*. Pro programování mikrokontroléru *ESP8266* je nezbytný konvertor *FTD1232*, který konvertuje *USB* na *TTL* sériovou komunikaci. Tento konvertor je zobrazený na obrázku 4.7. Propojení konvertoru *FTD1232* s mikrokontrolérem *ESP8266* je následující:

- **VCC** pin konvertoru je nutné propojit s pinem **3V3** mikrokontroléru,
- **GND** pin konvertoru je nutné propojit s pinem **GND** mikrokontroléru,
- **RX** pin konvertoru je nutné propojit s pinem **TX** mikrokontroléru,
- **TX** pin konvertoru je nutné propojit s pinem **RX** mikrokontroléru.

Pro přepnutí mikrokontroléru *ESP8266* do programovacího módu je nutné při zapnutí multisenzoru držet tlačítko, které se nachází na straně multisenzoru. Během nahrávání programu na mikrokontrolér musí být odpojeny propojky, které propojují oba mikrokontroléry, aby se zabránilo interferenci.

Mikrokontrolér *ATMega328P* je možné programovat pomocí *USBASP AVR* programovacího modulu zobrazeného na obrázku 4.8. Tento programátor s mikrokontrolérem komunikuje pomocí *SPI* protokolu. Propojení programátoru s mikrokontrolérem spočívá v propojení jednotlivých pinů *SPI*, které jsou **5V**, **GND**, **RST**, **SCK**, **MOSI** a **MISO**.

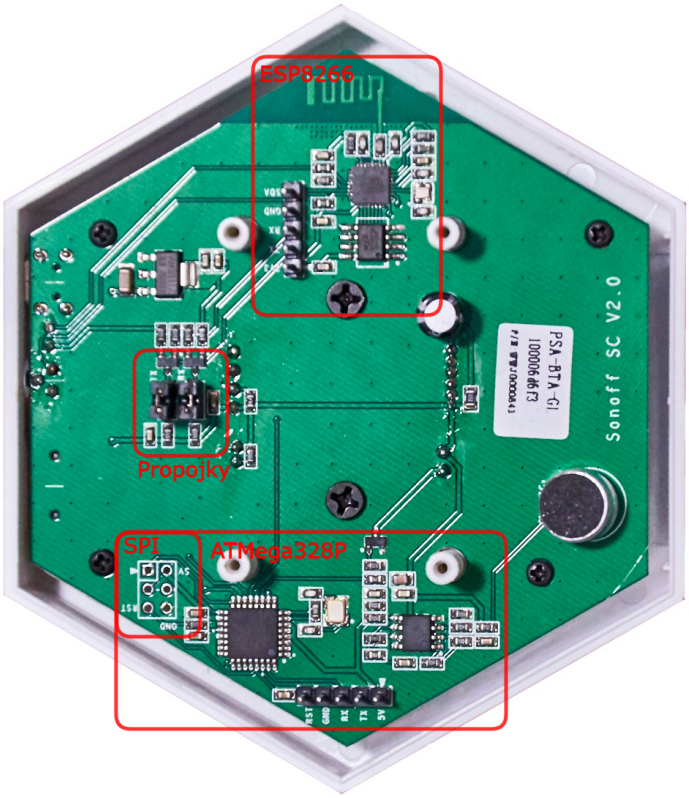


Obrázek 4.7: FTD1232

Obrázek 4.8: USBASP AVR programovací modul

[4] https://platformio.org

Při programování multisenzoru se pracuje se spodní stranou multisenzoru. Tato spodní strana hardware multisenzoru je dostupná na obrázku 4.9. Na obrázku jsou vyznačeny oba mikrokontroléry, propojky, které je nutné během programování mikrokontrolérů odpojit, a SPI rozhraní, pomocí kterého se propojuje *USBASP AVR* programovací modul s mikrokontrolérem *ATMega328P*.



Obrázek 4.9: Spodní strana hardware multisenzoru Sonoff SC

Získávání naměřených dat

Multisenzor začne periodicky sbírat data ze senzorů a publikovat je na definované *MQTT* téma po úspěšném připojení k lokální síti a k *MQTT* brokeru. Výchozí *MQTT* téma, na které multisenzor publikuje senzorová data, je `sonoffsc/data`. Zprávy, publikované na toto téma, jsou ve *JSON* formátu. Ukázka zprávy publikované multisenzorem je zobrazena ve výpisu 4.1.

Každá zpráva obsahuje elementy `time`, `host` a `ip`. Element `time` obsahuje čas, ve kterém byla zpráva vygenerována a je ve formátu `"rok/měsíc/den hodina:minuty:sekundy"`. Další prvek `host` obsahuje unikátní název senzoru, který pro multisenzor `Sonoff SC` vždy začíná prefixem `SONOFFSC_` a pokračuje třemi spodními bajty jeho MAC adresy. Posledním elementem, který je vždy obsažen ve zprávě, je `ip` obsahující IP adresu multisenzoru.

Multisenzor odesílá dva typy zpráv. První typ zprávy slouží k tomu, aby multisenzor dal vědět svému okolí o jeho přítomnosti. Tato zpráva navíc obsahuje prvek `heartbeat`, který obsahuje řetězec `"1"`. Multisenzor tuto zprávu odesílá každých pět minut. Druhý typ zprávy obsahuje naměřená data z jednotlivých senzorů. Periodu sbírání a odesílání naměřených dat umožňuje multisenzor konfigurovat. Výchozí perioda získávání a odesílání naměřených dat je

třicet sekund. Tato zpráva navíc obsahuje elementy `temperature`, `humidity`, `dust`, `light` a `noise`. Teplotu, multisenzor odesílá v elementu `temperature` ve stupních Celsia. Vlhkost je obsažena v elementu `humidity` a je v procentech. Element `dust` obsahuje množství prachu ve vzduchu. Tato hodnota je v jednotkách mikrogram na metr krychlový. Světelnost je obsažena v prvku `light` a je v jednotkách lux. Poslední element `noise` obsahuje hodnotu hlučnosti, která je vyjádřena v jednotkách decibel.

```
{
    "time":"2020/01/01 15:30:00",
    "host":"SONOFFSC_F8B105",
    "ip":"192.168.101.142"
    "temperature":"22.00",
    "humidity":"60",
    "dust":"26.00",
    "light":"35",
    "noise":"59",
}
```

Výpis 4.1: Sonoff zpráva

4.4 HomeMatic

Posledním výrobce analyzovaných zařízení v rámci této diplomové práce je *HomeMatic*. Konkrétně se jedná o chytrou zásuvku, chytrou termostatickou hlavici a magnetický okenní kontakt. Pro komunikaci se zařízeními zvolil výrobce *HomeMatic* rádio s frekvencí 868 MHz. Tato komunikační technologie může být dodána ve formě USB modulu **Busware CC1101-USB-Lite 868MHz CUL**[5], který je použit v rámci vytváření podpory pro tato zařízení do *BeeeOn* brány. Komunikační protokol *HomeMatic* zařízení není volně k dispozici, a proto je pro řízení těchto zařízení využit systém *FHEM*, který implementuje komunikaci s těmito zařízeními pomocí zmíněného USB modulu.

Aby bylo možné začít pracovat s *HomeMatic* zařízením, je jej nejprve nutné napárovat k USB rádiovému modulu. Každé zařízení má specifický způsob, jak se přepíná do párovacího režimu, který je zmíněn v rámci popisu každého zařízení.

4.4.1 FHEM systém

FHEM implementuje server chytré domácnosti pomocí jazyka *Perl*. *FHEM* je modulární a jeden z modulů je právě modul pro komunikaci s *HomeMatic* zařízeními. Tento server lze řídit pomocí webového rozhraní, pomocí aplikace nebo pomocí spojení *telnet*.

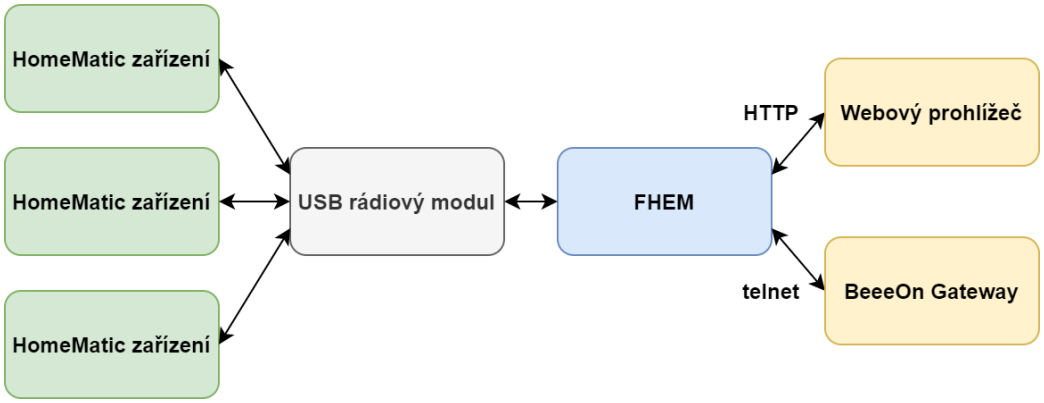
FHEM server je potřeba korektně nastavit, aby správně fungoval. Nastavení serveru se provádí pomocí konfiguračního souboru, kde je z pohledu podpory *HomeMatic* zařízení v *BeeeOn* bráně nutné definovat:

- port, na kterém má *FHEM* server naslouchat na *telnet* spojení
- cestu k USB modulu a jeho nastavení pro komunikaci s *HomeMatic* zařízeními

Část konfiguračního souboru pro *FHEM* je dostupný v příloze D, kde je ukázka konfigurace *telnet* portu a USB rádiového modulu pro komunikaci s *HomeMatic* zařízeními.

[5] http://busware.de/tiki-index.php?page=CUL

Způsob komunikace s *HomeMatic* zařízeními pomocí *FHEM* je znázorněn na obrázku 4.10. S *FHEM* serverem je možné komunikovat pomocí protokolu *telnet* nebo *HTTP*. *FHEM* server komunikuje s *HomeMatic* zařízeními pomocí USB rádiového modulu.



Obrázek 4.10: Komunikace s HomeMatic zařízeními

Používané příkazy pro řízení *HomeMatic* zařízení, které jsou odesílané *BeeeOn* bránou na *FHEM* server pomocí *telnet* spojení, jsou [3]:

`jsonlist2` – účelem příkazu je získat aktuální informace o službách a zařízení. Odpovědí *FHEM* serveru na tento příkaz je *JSON* zpráva, která obsahuje strom všech služeb a zařízení, které na serveru existují. Součástí každé entity jsou její parametry, nastavení a statistické informace. Jednou ze služeb je **ActionDetector**, která nabízí seznam všech známých zařízení *FHEM* serveru. V tomto seznamu je pro každé známé zařízení dostupné jeho jméno, živost a časové razítko poslední aktivity. Část odpovědi je dostupná ve výpisu 4.2, která obsahuje zmíněnou službu **ActionDetector**. V uzlu **"Readings"** se nachází seznam známých zařízení. V ukázce je pouze jedno známé zařízení s názvem `HM_30B0BE`. Na základě jména zařízení je potom možné se dotázat *FHEM* serveru pouze na informace o tomto zařízení. Dotaz, na samostatné zařízení nebo službu, je možný pomocí tohoto příkazu, ke kterému se přidá jako parametr požadovaná služba nebo zařízení. Tedy pro službu **ActionDetector** bude příkaz vypadat `jsonlist2 ActionDetector`.

```
{
  "Name":"ActionDetector",
  "Internals": {
    "FUUID": "5e410459-f33f-7c84-b801-95d2ac9920859610",
    "NAME": "ActionDetector",
    "STATE": "alive:1 dead:0 unkn:0 off:0",
    "TYPE": "CUL_HM",
    "chanNo": "01"
  },
  "Readings": {
    "state": {
        "Value":"alive:1 dead:0 unkn:0 off:0",
        "Time":"2020-02-02 20:20:00"
    },
    "status_HM_30B0BE": {
```

```
            "Value":"alive",
            "Time":"2020-02-02 20:20:00"
        }
      }
    }
```

Výpis 4.2: Část odpovědi *FHEM* serveru na příkaz `jsonlist2`

`set` – tento příkaz lze využít pro přepnutí USB modulu do párovacího režimu a tím hledat nová zařízení v senzorové síti. Parametry toho příkazu pro tento případ jsou název USB modulu (dle D je název modulu `CUL_0`), `hmPairForSec` a počet sekund po jakou dobu má USB modul setrvat v párovacím režimu. Příkaz lze také použít pro nastavení aktorového modulu zařízení do požadovaného stavu.

`delete` – pomocí tohoto příkazu lze odpárovat/odstranit zařízení z *FHEM* serveru. Při použití tohoto příkazu pro odpárování zařízení je parametrem příkazu pouze název odpárovávaného zařízení.

4.4.2 HomeMatic chytrá zásuvka

Prvním analyzovaným zařízení od výrobce *HomeMatic* je chytrá zásuvka. Zásuvka umožňuje měřit frekvenci elektrické sítě, napětí elektrické sítě, spotřebovaný proud a spotřebovaný výkon, ale také dovoluje vzdáleně měnit její stav vypnuto/zapnuto. Zásuvku je možné vidět na obrázku 4.11.

Aby bylo možné zásuvku ovládat a sbírat z ní data, je nutné zásuvku napárovat. Naparování zásuvky probíhá přepnutím zásuvky do párovacího režimu. Zásuvka se přepne do párovacího režimu, když je na zásuvce drženo tlačítko a zároveň je zapojována do elektrické sítě. Když je zásuvka v párovacím režimu, tak její LED bliká oranžově.

Obrázek 4.11: HomeMatic chytrá zásuvka

Odpověď *FHEM* serveru na dotaz na chytrou zásuvku je možné vidět ve výpisu 4.3. Odpověď ve výpisu není kompletní z důvodů její složitosti, ale obsahuje vše důležité. Z odpovědi na zařízení lze zařízení identifikovat pomocí uzlu `"model"`, který v případě zásuvky

obsahuje řetězec `"HM-ES-PMSW1-PL"`. Jednotlivé moduly zařízení *FHEM* server reprezentuje pomocí kanálů. Zásuvka má pět kanálů pro všechny dříve zmíněné moduly. Každý kanál zařízení je ve *FHEM serveru* reprezentován samostatnou entitou, která je adresovatelná pomocí jména kanálu.

- `HM_38D649_Sw` – kanál reprezentuje ovladatelný modul zařízení pro změnu stavu (zapnuta/vypnuta). Změnit stav tohoto modulu zásuvky lze odesláním příkazu `set HM_38D649_Sw {on|off}` na *FHEM* server.
- `HM_38D649_SenPwr` – kanál reprezentuje senzorový modul měřící výkon zásuvky. Naměřeny výkon je poskytován ve wattech.
- `HM_38D649_SenI` – kanál reprezentuje senzorový modul měřící protékající proud zásuvkou. Naměřeny proud je poskytován v miliampérech.
- `HM_38D649_SenU` – kanál reprezentuje senzorový modul měřící napětí elektrické sítě, do které je zásuvka zapojena. Naměřený proud je poskytován ve voltech.
- `HM_38D649_SenF` – kanál reprezentuje senzorový modul měřící frekvenci elektrické sítě, do které je zásuvka zapojena. Naměřený proud je poskytován v hertzích.

Odpověď také obsahuje různé statistické informace o zařízení, mezi které například patří počet přijatých/odeslaných zpráv ze/na zařízení a časové razítko dané události.


```
{
  "Name":"HM_38D649",
  "Internals": {
    "CUL_0_RSSI": "-29",
    "FUUID": "5e410459-f33f-7c84-89ee-5ef6a7f1e1c6140a",
    "NAME": "HM_38D649",
    "STATE": "CMDs_done",
    "channel_01": "HM_38D649_Sw",
    "channel_02": "HM_38D649_SenPwr",
    "channel_03": "HM_38D649_SenI",
    "channel_04": "HM_38D649_SenU",
    "channel_05": "HM_38D649_SenF",
    "protLastRcv": "2020-02-13 07:52:04",
    "protRcv": "1848 last_at:2020-02-13 07:52:04",
    "protSnd": "174 last_at:2020-02-13 07:32:06"
  },
  "Readings": {
    "Activity": {
        "Value":"alive",
        "Time":"2020-02-10 08:21:03"
    },
  },
  "Attributes": {
    "IODev": "CUL_0",
    "actStatus": "alive",
    "model": "HM-ES-PMSW1-PL",
    "serialNr": "MEQ0106579",
```

```
    "subType": "powerMeter"
  }
}
```


Výpis 4.3: Část odpovědi na požadavek na chytrou zásuvku

Část odpovědi na požadavek na kanál `HM_38D649_Sw` chytré zásuvky je dostupný ve výpisu 4.4. Hodnota kanálu se nachází v uzlu `"STATE"`. Dle ukázky odpovědi se zásuvka nachází ve stavu zapnuta.


```
{
  "Name":"HM_38D649_Sw",
  "Internals": {
    "FUUID": "5e410459-f33f-7c84-9c9e-8d22164308119e46",
    "NAME": "HM_38D649_Sw",
    "STATE": "on",
    "chanNo": "01",
    "device": "HM_38D649"
  },
  "Readings": {
    "CommandAccepted": {
        "Value":"yes",
        "Time":"2020-02-11 18:22:40"
    },
    "state": { "Value":"on", "Time":"2020-02-13 07:13:22" }
  },
  "Attributes": {
    "model": "HM-ES-PMSW1-PL"
  }
}
```


Výpis 4.4: Část odpovědi na požadavek na kanál chytré zásuvky

4.4.3 HomeMatic chytrá termostatická hlavice

Druhým zařízením od výrobce *HomeMatic*, které bylo analyzováno je chytrá termostatická hlavice. Termostatická hlavice obsahuje tři moduly, mezi které patří senzorový modul měřící aktuální teplotu vzduchu, aktorový modul určující požadovanou teplotu vzduchu a senzorový modul udávající míru otevření ventilu. Termostatická hlavice je napájena dvěma bateriemi typu **AA**. Fyzický vzhled termostatické hlavice je dostupný na obrázku 4.12.

Pro napárování termostatické hlavice je nutné termostatickou hlavici přepnout do párovacího režimu. To se provede podržením prostředního tlačítka termostatické hlavice do té doby než se na displeji zobrazí odpočet doby, po kterou termostatická hlavice setrvá v párovacím režimu.

Obrázek 4.12: HomeMatic chytrá termostatická hlavice

Odpověď serveru *FHEM* na dotaz na termostatickou hlavici je stejný jako dotaz na zásuvku. Odlišné jsou hodnoty uzlů `"model"` a `"subType"`. Hodnota uzlu `"model"` v případě termostatické hlavice je `"HM-CC-RT-DN"`, kterou lze použít pro identifikaci tohoto zařízení a uzel `"subType"` obsahuje řetězec `"thermostat"`. *FHEM* server k termostatické hlavici přidružuje šest kanálů, přičemž pro základní podporu je důležitý pouze kanál s názvem `HM_36BA59_Clima`. Pomocí toho kanálu lze získávat aktuální hodnoty všech modulů termostatické hlavice a také měnit požadovanou teplotu vzduchu. Část odpovědi na dotaz na tento kanál je ve výpisu 4.5. Hodnoty jednotlivých modulů se nacházejí v uzlu `"STATE"`. Dle ukázky je naměřena teplota vzduchu termostatickou hlavicí 21,5 °C, požadovaná teplota je 17 °C a ventil je otevřen na 40 procent.

Pro nastavení požadované teploty vzduchu slouží příkaz `set HM_36BA59_Clima desired-temp {teplota}`. Nastavovaná teplota musí být v rozmezí od 5 do 30 stupňů Celsia, přičemž minimální přírůstek teploty je 0,5 °C.

```
{
  "Name":"HM_36BA59_Clima",
  "Internals": {
    "FUUID": "5e41612b-f33f-7c84-0061-ee3420c2bd98550c",
    "NAME": "HM_36BA59_Clima",
    "STATE": "T: 21.5 desired: 17.0 valve: 40",
    "TYPE": "CUL_HM",
    "chanNo": "04",
    "device": "HM_36BA59"
  },
  "Readings": {
    "CommandAccepted": { "Value":"yes", "Time":"13:28:01" },
    "ValvePosition": { "Value":"40", "Time":"13:35:14" },
    "desired-temp": { "Value":"17.0", "Time":"13:35:14" },
    "measured-temp": { "Value":"21.4", "Time":"13:35:14" }
  },
  "Attributes": {
```

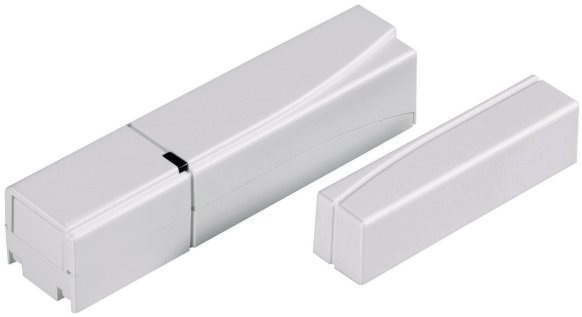
```
    "model": "HM-CC-RT-DN"
  }
}
```

Výpis 4.5: Část odpovědi na požadavek na kanál termostatické hlavice

4.4.4 HomeMatic magnetický kontakt

Posledním zařízením od *HomeMatic*, které bylo analyzováno je magnetický okenní/dveřní kontakt. Pomocí tohoto zařízení je možné měřit stav okna/dveří. Magnetický kontakt se může nacházet ve stavu otevřeno nebo zavřeno. Senzor je napájen pomocí dvou baterií typu **LR44**. Obrázek 4.13 obsahuje fotografii tohoto senzoru.

Stejně jako ostatní zařízení od výrobce *HomeMatic* je nutné magnetický kontakt nejprve napárovat. Pro napárování senzoru je nutné na senzoru stisknout a na chvíli držet tlačítko, které se nachází na zadní straně senzoru pod krytem. Signalizační LED bliká oranžově, když se senzor nachází v párovacím režimu.



Obrázek 4.13: HomeMatic magnetický kontakt

Odpověď *FHEM* serveru na dotaz na tento senzor vypadá obdobně jako odpověď na dotaz na chytrou zásuvky. Senzor je možné identifikovat pomocí uzlu `"model"` v odpovědi, který obsahuje řetězec `"HM-SEC-SC-2"`. Odlišnou hodnotu oproti zásuvky má v případě senzoru také uzel `"sybType"`, který nese hodnotu `"threeStateSensor"`. Senzor má jeden modul, jehož odpovídající kanál je obsažen přímo v odpovědi na dotaz na senzor. Hodnota kanálu se nachází v uzlu `"STATE"` a může nabývat dvou hodnot `"open"` (otevřeno) a `"closed"` (zavřeno).

Kapitola 5

Návrh integrace vybraných technologií do BeeeOn Gateway

Kapitola je zaměřena na návrh jednotlivých modulů pro podporu nových zařízení. Jako první je popsán návrh rozšíření již existujícího modulu pro podporu *Bluetooth Low Energy* o zařízení od výrobců *Revogi* a *Tabu Lumen*. Dále je rozebrán návrh modulu pro podporu multisenzoru *Sonoff SC* a modulu, který zajišťuje řízení zařízení od výrobce *HomeMatic*. Ke každé části je dostupný diagram tříd, který znázorňuje strukturu modulu. Během návrhu jednotlivých modulů se již počítalo s tím, aby přidaní podpory nového zařízení bylo co nejjednodušší.

5.1 Návrh rozšíření Bluetooth LE modulu o Revogi a Tabu Lumen podporu

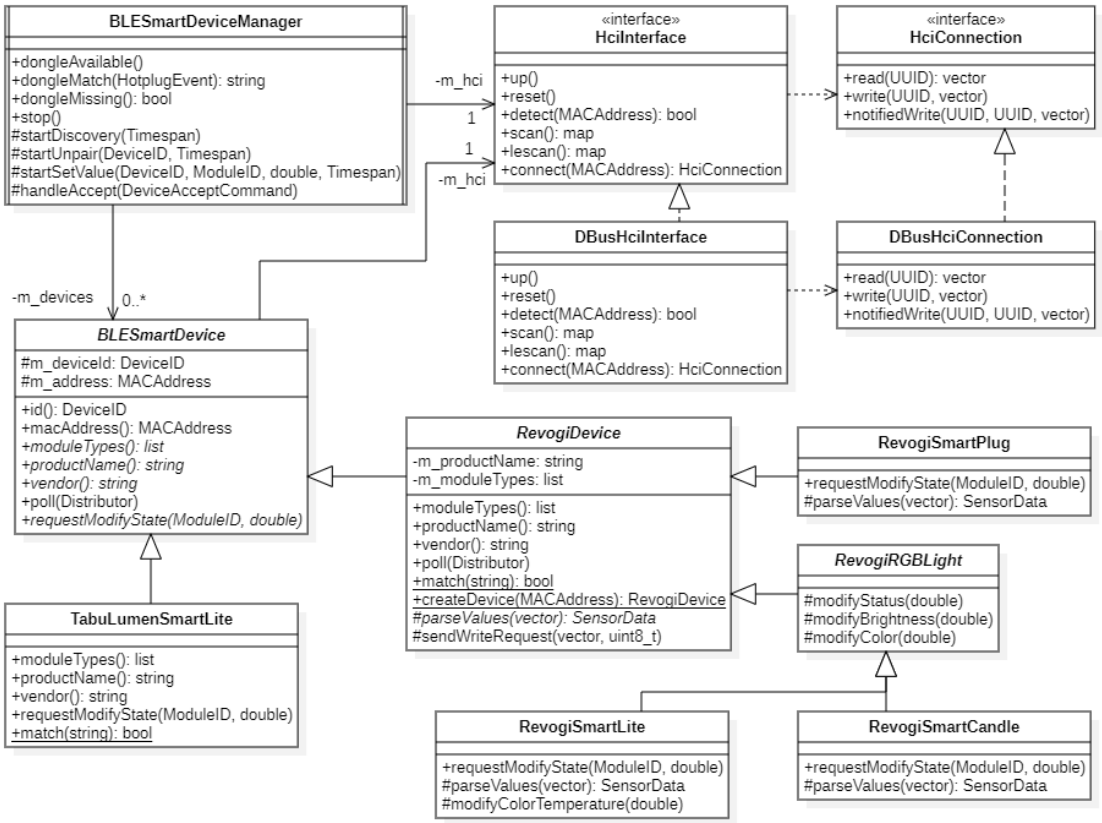
V rámci vytváření podpory pro zařízení od výrobců *Revogi* a *Tabu Lumen* je rozšiřován již existující modul pro podporu *Bluetooth Low Energy* zařízení, který byl mnou vytvořen v rámci bakalářské práci. Výsledný návrh rozšíření tohoto modulu je dostupný v diagramu tříd, zobrazen na obrázku 5.1.

Jádrem modulu pro podporu *Bluetooth Low Energy* zařízení je manažer zařízení, který je reprezentován třídou **BLESmartDeviceManager**. Tato třída dědí z abstraktní třídy **DongelDeviceManager**, která nabízí společnou funkcionalitu všech manažerů zařízení zděděním abstraktní třídy **DeviceManager** a navíc řeší akce spojené s připojitelným zařízením (například USB modul).

Aby třídy využívající *Bluetooth* komunikaci nemuseli být závislé na konkretní implementaci, existují rozhraní **HciInterface** a **HciConnection**. Jednou z implementací těchto rozhraní je **DBusHciInterface** a **DBusHciConnection**, které pro zajištění *Bluetooth* komunikace využívají *Bluetooth démon*, s kterým komunikují pomocí *DBus* sběrnice. **DBusHciInterface** třída implementuje metody pro vyhledávání zařízení na síti *Bluetooth Classic* a *Bluetooth Low Energy*, detekování dostupnosti *Bluetooth Classic* zařízení a vytvoření spojení s *Bluetooth Low Energy* zařízením. Spojení s *Bluetooth Low Energy* zařízením je reprezentováno třídou **DBusHciConnection**. Tato třída implementuje metody pro výměnu dat v rámci spojení. Pro komunikaci se zařízeními pomocí technologie *Bluetooth Low Energy* má manažer zařízení v asociaci rozhraní **HciInterface**. Manažer zařízení je takto pouze závislí na rozhraní a je mu možné jednoduše měnit implementaci *Bluetooth* komunikace.

Pro uniformní práci manažera zařízení se všemi zařízeními je vytvořena abstraktní třída **BLESmartDeviceManager**. Ta definuje rozhraní každé třídy reprezentující zařízení, kam patří metody pro získání informací o zařízení, metoda pro získání stavu zařízení a metoda pro nastavení požadované hodnoty na zařízení. Pro každé zařízení, pro které je vytvářena podpora je reprezentována samostatnou třídou. Inteligentní žárovka od *Tabu Lumen* je reprezentována třídou **TabuLumenSmartLite**. Tato třída implementuje komunikaci a ovládání fyzické *Tabu Lumen* žárovky.

V rámci přidání podpory pro *Revogi* zařízení je vytvořena abstraktní třída **RevogiDevice**, která implementuje společné metody pro všechny *Revogi* zařízení. Mezi ně patří metody pro získání informací o zařízení, metoda pro odeslání požadavku a metoda pro získání aktuálního stavu zařízení, přičemž zpracování příchozí odpovědi na dotaz na aktuální stav zařízení musí implementovat třída konkretního zařízení. Třída **RevogiSmartPlug** představuje *Revogi* chytrou zásuvku a zajišťuje zpracování aktuálních naměřených dat zásuvkou a ovládání zásuvky. *Revogi* inteligentní svíčka a inteligentní žárovka spolu sdílí funkcionalitu, která je abstrahována do abstraktní třídy **RevogiRGBLight**. Zpracování odpovědi na dotaz na aktuální stav a ovládání je implementováno konkrétně pro svíčku v třídě **RevogiSmartCandle** a pro žárovku v třídě **RevogiSmartLite**.



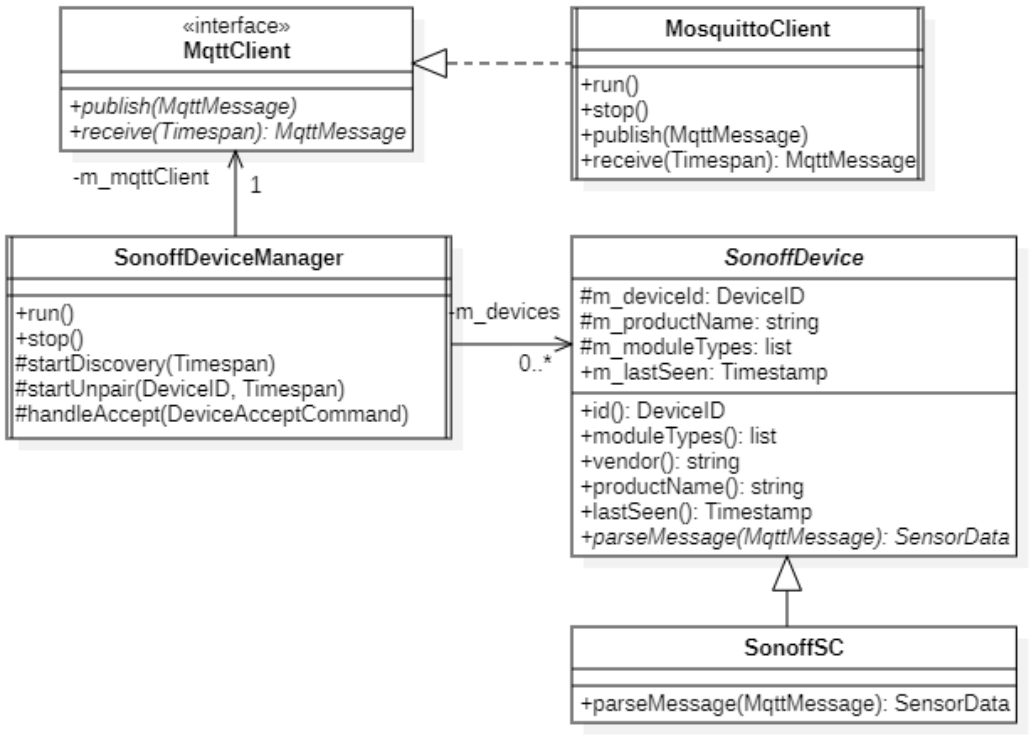
Obrázek 5.1: Návrh rozšíření Bluetooth LE modulu

5.2 Návrh podpory Sonoff

Objektově orientovaný návrh modulu pro podporu *Sonoff* zařízení je dostupný na obrázku 5.2 ve formě diagramu tříd. Hlavní třídou, která řídí veškerou činnost modulu je **SonoffDeviceManager** implementující zodpovědnosti manažera zařízení. Tato třída dědí abstraktní třídu **DeviceManager**, čímž získá společnou funkcionalitu manažerů zařízení. Zde patří metody pro příjem příkazů ze serveru, získání seznamu napárovaných zařízení ze serveru, odeslání naměřených senzorových dat a přístup k podpůrné službě **DeviceCache**.

Komunikace s multisenzory *Sonoff SC* je realizována pomocí *MQTT* protokolu, a proto je **SonoffDeviceManager** asociován s rozhraním **MqttClient**. Díky tohoto rozhraní není manažer zařízení závislý na konkretní implementaci *MQTT* klienta, a zároveň mu může být jednoduše měněna. V *BeeeOn* bráně již existuje jedna implementace rozhraní **MqttClient**, která využívá pro *MQTT* komunikaci knihovnu *mosquitto*. Tato implementace se nachází v třídě **MosquittoClient**. Manažer zařízení pomocí *MQTT* klienta přijímá zprávy od multisenzorů a ty dále zpracovává.

Aby do budoucna bylo jednoduché přidávat podporu nových *Sonoff* zařízení byla navržena abstraktní třída **SonoffDevice**. Ta zajišťuje manažeru zařízení jednotnou práci se všemi podporovanými zařízeními. Abstraktní třída **SonoffDevice** implementuje společné chování *Sonoff* zařízení a definuje rozhraní, které každé *Sonoff* zařízení musí podporovat. Multisenzor *Sonoff SC* je reprezentován třídou **SonoffSC**, která implementuje zpracování příchozích zpráv od fyzického multisenzoru. Pro každý nalezený multisenzor na síti existuje v systému jedna instance třídy **SonoffSC**, která je identifikovatelná pomocí atributu **m_deviceId**.



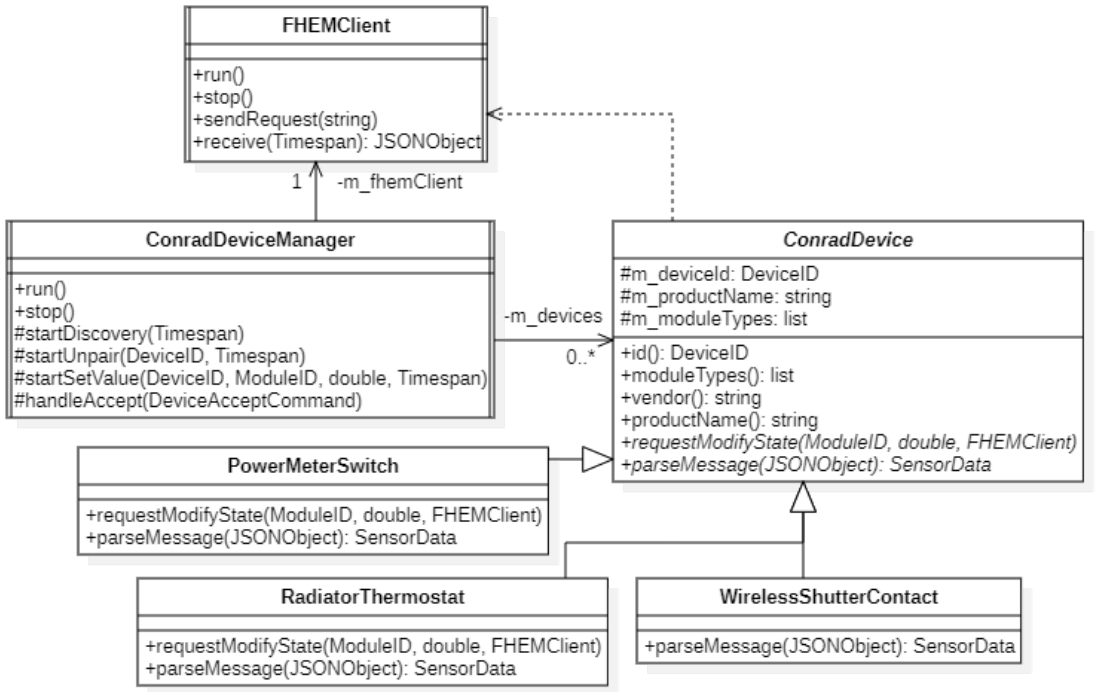
Obrázek 5.2: Objektový návrh Sonoff modulu

5.3 Návrh podpory HomeMatic

Jako poslední byl vytvořen návrh modulu pro podporu *HomeMatic* zařízení, která byla analyzována. Objektově orientovaný návrh modulu je dostupný na obrázku 5.3. Návrh obsahuje třídy, které zajišťují řízení a komunikaci se zařízeními, a vazby mezi těmito třídami.

Struktura modulu je velmi podobná jako u modulu pro podporu *Sonoff* zařízení. Třída **ConradDeviceManager** řídí veškerou činnost modulu. Stejně jako u předchozího modulu **ConradDeviceManager** dědí abstraktní třídu **DeviceManager** a tím získává společnou funkcionalitu manažerů zařízení. Komunikace se zařízeními probíhá pomocí *FHEM* serveru, kterou zajišťuje třída **FHEMClient**. Manažer zařízení modulu vlastní referenci na **FHEMClient** a pomocí něj přijímá a zpracovává vzniklé události na *FHEM* serveru.

Stejně jako u předchozích modulů existuje zde abstraktní třída **ConradDevice**, která reprezentuje generické zařízení a umožňuje uniformní práci manažera zařízení se všemi podporovanými zařízeními v modulu. Abstraktní třída implementuje společné metody mezi všemi *HomeMatic* zařízeními a definuje rozhraní pro změnu stavu zařízení a zpracování naměřených dat zařízením. Pro každý typ zařízení existuje v návrhu samostatná třída. Každá třída představující konkrétní fyzické zařízení dědí abstraktní třídu **ConradDevice** a implementuje metodu pro zpracování událostí určených danému zařízení a případně metodu pro nastavení stavu zařízení pokud dané zařízení umožňuje měnit svůj stav.



Obrázek 5.3: Objektový návrh HomeMatic modulu

Kapitola 6

Implementace podpory nových technologií a zařízení

V této kapitole je popsán způsob implementace jednotlivých navržených modulů pro podporu nových technologií a zařízení. V první části kapitoly jsou zmíněny knihovny, které byly v rámci implementace použity. Dále je rozebrán způsob implementace jednotlivých modulů. V rámci každého modulu existuje popis práce manažera zařízení se zařízeními, vyhledávání nových zařízení a komunikace se zařízeními. Závěr kapitoly se věnuje způsobu překladu nově vytvořených modulů.

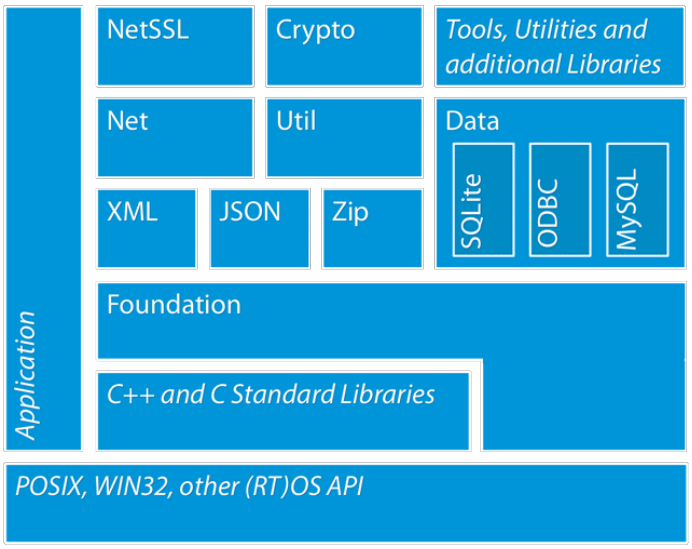
6.1 Použité technologie

Software *BeeeOn* brány je implementována v jazyce *C++*. Tento jazyk je překládaný a podporuje procedurální, ale také objektové paradigma. V rámci vývoje softwaru *BeeeOn* brány a v celém systému *BeeeOn* je primárně využíváno objektové paradigma jazyka *C++*. Implementace nových modulů využívá standardní knihovny jazyka *C++* a také externí knihovny *POCO*, *GLib* a *mosquitto*, které jsou podrobněji popsány v následujících sekcích.

6.1.1 Knihovna POCO

Knihovna *POCO C++* se skládá ze sady open-source knihoven, které vznikly a stále vznikají za účelem zjednodušení a urychlení vývoje síťových a internetových aplikací. Knihovna může být použita pro aplikace běžící na stolním počítači, serveru, mobilu, IoT a vestavěném systému. Struktura sady knihoven *POCO* je dostupná na obrázku 6.1 [1].

Při vytváření nových modulů do softwaru *BeeeOn* brány byly využity **Foundation**, **Net** a **JSON** knihovny ze sady knihoven *POCO*. Knihovna **Foundation** je jádrem *POCO* a tvoří abstraktní vrstvu pro ostatní knihovny. Tato knihovna obsahuje často používané obslužné třídy a funkce. Z knihovny **Foundation** byly použity třídy pro vytváření a správu vláken, pro převod textových řetězců na čísla, pro práci s časovými údaji a pro vytváření/odchytávání výjimek. Knihovna **Net** nabízí třídy pro přenos dat a může být využita pro jednoduchou *TCP/UDP* komunikaci, ale také pro komunikaci pomocí vysokoúrovňových aplikačních protokolů jako například *HTTP*. Tato knihovna je konkrétně využita pro komunikaci s *FHEM* serverem pomocí protokolu *telnet*. Poslední použita knihovna **JSON** implementuje čtení, zpracování a vytváření zpráv ve formátu *JSON*. Zprávy vyměňované se *Sonoff* multisenzorem a s *FHEM* serverem jsou ve formátu *JSON* a pro práci s těmito zprávami je využita právě tato knihovna.



Obrázek 6.1: Struktura knihovny POCO

6.1.2 Knihovna GLib

GLib knihovna se skládá z balíku nízkoúrovňových systémových knihoven. Tato knihovna je napsána v programovacím jazyce *C*. Knihovna nabízí různé pokročilé datové struktury, implementuje práci s vlákny a procesy, poskytuje synchronizační nástroje, umožňuje uživateli využít hlavní smyčku, pomocí které realizuje zpracování callbacků připojených k signálům nebo časovačům, a další. [5].

Z pohledu implementace nových modulu je tato knihovna využita v modulu pro podporu *Bluetooth Low Energy* zařízení. Tento modul z této knihovny využívá *DBus* aplikační rozhraní, pomocí kterého komunikuje s *Bluetooth démonem*, jenž realizuje přenos dat pomocí technologie *Bluetooth*. Dále modul také využívá hlavní smyčku knihovny, kde se vykonávají registrované callbacky na události spojené s komunikací *Bluetooth*. Mezi tyto události patří například změna RSSI zařízení, na základě, kterého se detekuje jeho živost, nebo nastavení příznaku u zařízení indikující ustanovení připojení s tímto zařízením.

6.1.3 Eclipse Mosquitto

Eclipse Mosquitto nabízí open-source zprostředkovatele zpráv implementující *MQTT* protokol. Dále taky nabízí klientské služby pro odebírání zpráv `mosquitto_sub` a sdílení zpráv `mosquitto_pub`. Nejdůležitější části *Eclipse Mosquitto* z pohledu implementace nových modulů je knihovna *libmosquitto* implementována v jazyce C, která zjednodušuje vytváření vlastních *MQTT* klientů.

Knihovna pro vývoj vlastního *MQTT* klienta *libmosquitto* je využita v modulu pro podporu *Sonoff* zařízení. Pomocí této knihovny je implementován *MQTT* klient *BeeeOn* brány, který zajišťuje komunikaci s multisenzory od výrobce *Sonoff*. Implementace *MQTT* klient je také využita v jiných modulech brány, kde příkladem může být modul pro export senzorových dat **MqttExporter**.

6.2 Implementace jednotlivých modulů

Jak už bylo zmíněno pří návrhu jednotlivých modulů, řídící třídou modulu je manažer zařízení. Každý manažer zařízení běží ve vlastním vlákně, ve kterém provádí konkrétní řídící činnost. Tou například může být příjem událostí od *FHEM* serveru nebo příjem zpráv z *MQTT* brokeru.

Manažer zařízení všech modulů musí dědit abstraktní třídu **DeviceManager**, což zaručuje, že je možně se všemi manažery zařízení zacházet stejně. Abstraktní třída **DeviceManager** implementuje rozhraní **CommandHandler** a to potom umožňuje, že každý manažer může být registrován u služby **CommandDispatcher**, který distribuuje přijaté příkazy bránou mezi moduly. Stejným způsobem, implementováním rozhraní **DeviceStatusHandler**, je zaručeno, že každý manažer zařízení, může být registrován u služby **DeviceStatusFetcher**, která získává napárovaná zařízení a jejich poslední stav ze serveru pro jednotlivé manažery zařízení.

Zděděním abstraktní třídy **DeviceManager** také každý manažer zařízení dostane implementaci metody pro nastavení reference na služby **Distributor** a **DeviceCache**, pro odesílání příkazů, pro odesílání senzorových dat a pro obecné zpracování příchozích příkazů. Abstraktní třída definuje abstraktní metody pro zpracování příkazů, které jsou implementovány konkretním manažerem konkretním způsobem. Manažer zařízení nemusí implementovat zpracování všech příkazů, ale pouze těch, které podporuje. Takže, když se manažer zařízení stará pouze o senzory, které neumožňují měnit svůj stav, tak manažer zařízení neimplementuje zpracování příkazu `set-value`.

Každý nový/rozšiřovaný manažer zařízení si udržuje nalezená zařízení v kontejneru standardní knihovny *C++* **std::map**. Tento kontejner udržuje dvojice, kde prvním prvkem je identifikátor zařízení **DeviceID**, který je unikátní v rámci kontejneru, a druhým prvek je reference na konkrétní instanci třídy reprezentující zařízení. Zařízení může manažer zařízení vnímat jako napárované nebo nenapárované. Pro rozlišení stavu napárovanosti zařízení používá manažer zařízení službu **DeviceCache**, která uchovává identifikátory zařízení, která jsou napárovaná.

Všechny moduly abstrahují práci se všemi podporovanými zařízení pomocí abstraktní třídy představující generické zařízení. Tato abstraktní třída definuje jaké metody musí každá třída zařízení nabízet. Patří zde metody pro získání parametrů zařízení jako identifikátor zařízení, název zařízení, výrobce zařízení, moduly zařízení. Výsledky těchto metod jsou využity při odesílání příkazu `new-device` na server, který obsahuje všechny tyto údaje o zařízení. Dále musí každá třída zařízení poskytovat metody pro získání aktuálních hodnot jednotlivých modulů a pro nastavení hodnoty na konkretní hodnotu.

6.2.1 Modul pro podporu Revogi a Tabu Lumen

V rámci implementace podpory pro zařízení od *Revogi* a *Tabu Lumen* byl využit již existující modul pro podporu *BeeWi* zařízení, která stejně jako *Revogi* a *Tabu Lumen* zařízení komunikují pomocí technologie *Bluetooth Low Energy*. Tento modul pro podporu *BeeWi* zařízení byl vytvořen mnou v rámci bakalářské práce[1]. Pro vytvoření podpory pro zařízení od *Revogi* a *Tabu Lumen* byl tento modul zobecněn na modul podporující obecně všechna inteligentní zařízení komunikující pomocí *Bluetooth Low Energy*. Modul je řízen manažerem zařízení, který je implementován třídou **BLESmartDeviceManager**. Technologie *Bluetooth* je do brány dodána pomocí *USB* modulu a tedy manažer zařízení se musí

[1] https://www.fit.vut.cz/study/thesis-file/21068/21068.pdf

kromě zařízení, sběru dat ze zařízení a zpracování příkazů ze serveru, také starat o problémy spojené s přítomnosti *USB* modulu. Tyto problémy pomáhá řešit abstraktní třída **DongleDeviceManager**, kterou **BLESmartDeviceManager** dědí a implementuje.

O *Bluetooth* komunikaci se starají třídy **DBusHciInterface** a **DBusHciConnection**. Tato implementace *Bluetooth* komunikace používá *Bluetooth* démona, se kterým komunikuje pomocí *DBus* sběrnice. *Bluetooth* démon je v roli *DBus* serveru a skládá se z několika objektů. Relevantními objekty jsou `Adapter1`, `Device1` a `GattCharacteristic1`. Každý *DBus* objekt se skládá z metod, atributů a signálů, ke kterým je možné připojit callbacky. Objekt `Adapter1` reprezentuje *Bluetooth* rozhraní a umožňuje rozhraní přepnout do vyhledávacího režimu a naopak, vypnout a zapnout rozhraní a další. *Bluetooth* zařízení je reprezentováno objektem `Device1`, který nabízí metody pro práci se zařízením. Zde patří metody pro získání jména zařízení, pro získání služeb zařízení, pro vytvoření/ukončení připojení se zařízením a další. *Bluetooth Low Energy* zařízení poskytuje určité služby, které se dále dělí na vlastnosti. Tyto vlastnosti jsou reprezentovány objektem `GattCharacteristic1`. Tento objekt nabízí metody pro čtení hodnoty vlastnosti a zápis hodnoty do vlastnosti.

DBus objekty jsou popsány pomocí *XML* dokumentů. Pro zjednodušení komunikace a práce s těmito *DBus* objekty je použit nástroj **gbus-codegen**, který vygeneruje, na základě *XML* dokumentu popisující *DBus* objekt, modul v jazyce *C* využívající *DBus* aplikační rozhraní z knihovny *GLib* pro veškerou práci s daným *DBus* objektem.

Vyhledávání zařízení

O vyhledávání zařízení na *Bluetooth* síti se stará třída **DBusHciInterface**. Tato třída po spuštění *BeeeOn* brány přepne aktivní *Bluetooth* rozhraní do vyhledávacího režimu a registruje si u něj callback na událost vytvoření nového objektu `Device1` (nalezení nového zařízení). Při nalezení nového zařízení se zavolá zmíněný callback, ve kterém se registruje callback k novému zařízení na změnu RSSI hodnoty, a také se vytvoří vnitřní reprezentace zařízení uložená v **DBusHciInterface** obsahující časové razítko poslední změny hodnoty RSSI. V připojeném callbacku k zařízení na změnu RSSI hodnoty se aktualizuje časové razítko poslední změny RSSI hodnoty daného zařízení. Metoda třídy **DBusHciInterface**, pro navracení dostupných *Bluetooth Low Energy* zařízení, vrátí seznam zařízení, pro které platí, že poslední změna RSSI hodnoty proběhla maximálně před určitou dobou. RSSI hodnoty aktivních zařízení se v *Bluetooth* démonovi mění pouze pokud je odpovídající *Bluetooth* rozhraní v režimu objevování, proto třída **DBusHciInterface** udržuje *Bluetooth* rozhraní v tomto režimu.

Výsledkem metody pro navrácení všech dostupných *Bluetooth Low Energy* zařízení je seznam MAC adres. Každou MAC adresu je nutné dále prozkoumat a zjistit jaký typ zařízení identifikuje. Pro většinu zařízení platí, že typ zařízení se nachází ve standardní **GATT** vlastnosti pro název modelu. Manažer zařízení na základě názvu modelu zařízení instanciuje konkrétní třídu zařízení.

Práce se zařízeními

Oproti původnímu rozložení modulu byla vytvořena nová abstraktní třída **BLESmartDevice**, která abstrahuje všechny podporované zařízení modulem. Tato abstraktní třída dědí rozhraní **PollableDevice**, což umožní manažeru zařízení použít podpůrnou službu **DevicePoller**, která se periodicky dotazuje registrovaných zařízení na jejich stav a odesílá získaná senzorová data na server.

Pro každé podporované fyzické zařízení existuje samostatná třída, která dědí a implementuje abstraktní třídu **BLESmartDevice**. Hierarchie tříd reprezentující zařízení odpovídá návrhu. Pro inteligentní žárovku od *Tabu Lumen* byla vytvořena třída *TabuLumenSmartLite*, která implementuje komunikační protokol pro komunikaci s fyzickou žárovkou. Inteligentní zásuvka od *Revogi* je reprezentována třídou **RevogiSmartPlug**. *Revogi* žárovku a svíčku představují třídy **RevogiSmartLite** a **RevogiSmarCandle**. Jejich společná funkcionalita je vyňata do abstraktní třídy **RevogiRGBLight**. Stejně jak bylo popsáno v návrhu, byla vytvořena abstraktní třída **RevogiDevice**, která implementuje společnou funkcionalitu všech *Revogi* zařízení. Aby více vláken nekomunikovalo s jedním zařízením zároveň, dědí **BLESmartDevice** třídu **Poco::SynchronizedObject**, která umožňuje instancím zařízení se zamknout, a tím zajistit výlučný přístup k zařízení.

Konfigurace modulu

Mezi konfigurovatelné parametry modulu patří `enable`, který povoluje spuštění modulu pro podporu *Bluetooth Low Energy* zařízení. Dalším parametrem je `device.timeout`, který určuje časový limit pro práci se zařízením. Zde patří připojení k zařízení, načtení služeb zařízení a čtení/zápis z/do vlastnosti. Doba, po kterou bude metoda pro nalezení dostupných zařízení blokující určuje parametr `le.scanTime`. Dále parametr `refresh` určuje časový interval získávání senzorových dat z napárovaných zařízení. Počet vláken, která jsou využita pro identifikaci dostupných zařízení, určuje parametr `numberOfExaminationThreads`. Parametrem `hci.impl` se určuje implementace *Bluetooth* komunikace. Současně je možné také konfigurovat třídu **DBusHciInterface** a to konkretně pomocí parametru `le.maxAgeRssi` určující maximální stáří hodnoty RSSI, aby bylo zařízení považováno za dostupné. Část konfiguračního souboru *BeeeOn* brány, konfigurující tento modul, je dostupná ve výpisu 6.1.

```
[bluetooth]
le.maxAgeRssi = 90 s

[blesmart]
enable = yes
device.timeout = 5
s~le.scanTime = 5
s~refresh = 120
s~numberOfExaminationThreads = 3
hci.impl = dbus
```

Výpis 6.1: Konfigurace BLESmart modulu

6.2.2 Modul pro podporu Sonoff

Integrací tohoto modulu do *BeeeOn* brány je přidána podpora pro zařízení od výrobce *Sonoff*. Stejně jako u všech modulů je řídící třídou modulu manažer zařízení implementován ve třídě **SonoffDeviceManager**. Manažerem zařízení je aktuálně podporováno pouze jedno zařízení, které neumožňuje měnit stav žádného jeho modulu, a proto není manažerem zařízení podporovaný příkaz `set-value`.

Komunikace mezi manažerem zařízení a zařízeními probíhá pomocí *MQTT* protokolu. Pro příjem respektive odesílání zpráv manažer zařízení využívá již existující třídu **MosquittoClient**, která je manažeru zařízení dodána pomocí reference skrz rozhraní **MqttClient**.

Tímto způsobem je zajištěno, že manažer zařízení není závislý na jedné konkretní implementaci *MQTT* klienta. Třída **MosquittoClient** běží ve vlastním vlákně, ve kterém přijímá a zpracovává příchozí zprávy na odebíraná *MQTT* témata. Každou přijatou zprávu vloží do fronty zpráv, ze které jsou poté zprávy ve stejném pořadí odebírány a předány manažeru zařízení, když se manažer zařízení dotáže na další zprávu.

Vyhledávání zařízení

Manažer zařízení provádí vyhledávání nových zařízení po celý čas jeho spuštění. Podporovaný multisenzor *Sonoff SC* iniciuje periodické odesílání zpráv, které manažer zařízení přijímá a v případě, že se jedná o multisenzor, jenž je pro manažera zařízení neznámý, vytvoří pro něj instanci třídy reprezentující tento multisenzor. Každá instance třídy reprezentující multisenzor obsahuje časové razítko poslední přijaté zprávy od daného fyzického multisenzoru. Když manažer zařízení přijme příkaz od serveru na hledání nových zařízení, tak projde svůj kontejner obsahující všechna známá zařízení a odešle příkaz `new-device` pro každé zařízení, od kterého byla přijata zpráva před definovanou maximální dobou.

Práce se zařízeními

Stejně jak bylo popsáno v návrhu tohoto modulu, aby manažer zařízení mohl pracovat se všemi podporovanými zařízeními jednotně, byla vytvořena abstraktní třída **SonoffDevice** definující rozhraní každé třídy představující zařízení. Abstraktní třída implementuje metody pro získání veškerých informací o zařízení, které jsou potřebné při vytváření příkazu `new-device`. Mezi tyto informace patří identifikátor zařízení, název zařízení, název výrobce zařízení, moduly zařízení a čas obnovení. Abstraktní třída také definuje metodu pro zpracování příchozí zprávy od fyzického zařízení, kterou musí implementovat konkretní třída představující zařízení.

Momentálně je modulem podporovaný pouze multisenzor *Sonoff SC*, který je v modulu reprezentován třídou **SonoffSC**. Tato třída implementuje transformaci příchozí zprávy od fyzického multisenzoru na senzorová data, která jsou poté manažerem zařízení odeslána na server.

Zjištěni, kterému zařízení náleží příchozí zpráva se provádí na základě elementu `host` JSON zprávy. Tento element obsahuje řetězec začínající prefixem `SONOFFSC_` a pokračuje třemi spodními bajty MAC adresy multisenzoru zapsané v hexadecimálním tvaru. Prefix je použit pro identifikaci typu zařízení, a část MAC adresy je použita pro vytvoření identifikátoru zařízení.

Konfigurace modulu

Konfigurace modulu, vyňata z konfiguračního souboru *BeeeOn* brány, je dostupná ve výpisu 6.2. Stejně jako pro většinu modulů je možné modul povolit či zakázat pomocí parametru `enable`. Dále pomocí parametru `maxLastSeen` lze určit maximální dobu neaktivity multisenzoru, aby byl multisenzor reportován serveru manažerem zařízení jako dostupné zařízení. Parametrem `subscribeTopics` se zadávají *MQTT* klientu jaké *MQTT* témata má odebírat. Doménové jméno a port *MQTT* brokeru, na který se má *MQTT* klient připojit, se konfiguruje pomocí parametrů `mqtt.host` a `mqtt.port`. Posledním parametrem `mqtt.clientID` je možné konfigurovat identifikátor *MQTT* klienta.

```
[sonoff]
enable = yes
```

```
maxLastSeen = 600
s~subscribeTopics = sonoffsc/#
mqtt.host = localhost
mqtt.port = 1883
mqtt.clientID = SonoffClient
```

Výpis 6.2: Konfigurace Sonoff modulu

6.2.3 Modul pro podporu HomeMatic

Posledním modulem vytvářeným v rámci této práce je modul pro podporu *HomeMatic* zařízení. Řídící třídou modulu je manažer zařízení, jehož zodpovědnosti implementuje třída **ConradDeviceManager**. Implementace modulu se držela jeho návrhu a tedy struktura modulu odpovídá jeho návrhu.

Modul pro komunikaci se zařízeními používá *FHEM* server, se kterým zajišťuje komunikaci třída **FHEMClient**. Tato třída se periodicky dotazuje *FHEM* serveru na všechna známá *HomeMatic* zařízení a jejich stav, a na základě odpovědi *FHEM* serveru generuje události, které ve formě *JSON* objektu ukládá do fronty událostí. Události jsou pak ve stejném pořadí jak vznikaly, poskytovány manažeru zařízení. **FHEMClient** dokáže generovat tři typy událostí a to událost nalezení nového zařízení, událost změny stavu zařízení a statistickou událost. V případě, že **FHEMClient** obdrží v odpovědi nové *HomeMatic* zařízení, generuje událost o novém zařízení obsahující název zařízení, typ zařízení, model zařízení a sériové číslo zařízení. *FHEM* klient si o každém zařízení uchovává časové razítko poslední přijaté zprávy od fyzického zařízení a počet odeslaných a přijatých zpráv. Tyto informace poskytuje o zařízení *FHEM* server. V případě, že **FHEMClient** detekuje aktualizaci časového razítka poslední přijaté zprávy od zařízení, generuje událost změny stavu zařízení, která obsahuje informace o zařízení a stav jednotlivých kanálu zařízení. Podobným způsobem se generuje statistická událost, která je generována když **FHEMClient** zjistí, že se počet přijatých/odeslaných zpráv daným zařízením změnil.

Vyhledávání zařízení

Manažer zařízení, po přijetí příkazu na objevování nových zařízení, odešle na *FHEM* server příkaz na přepnutí *USB* modulu do párovacího režimu. Doba, po kterou *USB* modul setrvá v párovacím režimu, je dána hodnotou obsaženou v příkazu `listen`. Hledání nových zařízení probíhá v hlavním vlákně manažera zařízení, kde se dotazuje třídy **FHEMClient** na vzniklé události. V případě, že manažer zařízení příjme událost nalezení nového zařízení, tak vytvoří instanci konkretní třídy zařízení a odešle příkaz `new-device` na server. K odeslání příkazu `new-device` dochází také v případě, kdy se příjme událost na změnu stavu zařízení, které ještě manažer zařízení nezná.

Práce se zařízeními

Pro každé podporované *HomeMatic* zařízení byla vytvořena vlastní třída, která implementuje specifické zpracování události vytvořené *FHEM* klientem. Taktéž třída reprezentující konkretní zařízení implementuje metodu pro změnu stavu modulu v případě, že zařízení podporuje nastavení hodnoty nějakého modulu. Společné metody a atributy tříd zařízení jsou vyjmuty do abstraktní třídy **ConradDevice**, kterou každá korektní třída dědí a implementuje.

Tento modul aktuálně podporuje tři *HomeMatic* zařízení. Prvním zařízením je *HomeMatic* chytrá zásuvka, která je reprezentována třídou **PowerMeterSwitch**. Dalším podporovaným zařízením je *HomeMatic* inteligentní termostatická hlavice. O transformaci událostí na senzorová data z tohoto zařízení a odesílání příkazů na nastavení hodnoty modulu se stará třída **RadiatorThermostat**. Třetím a posledním zařízením je *HomeMatic* magnetický kontakt, jenž je v modulu reprezentován třídou **WirelessShutterContact**.

Konfigurace modulu

V rámci konfigurace modulu je možné modul zakázat či povolit pomocí parametru `enable`. Parametr `fhem.address` specifikuje IP adresu a port, na kterém je dostupná služba `telnet` poskytována *FHEM* serverem. Jak často se bude **FHEMClient** dotazovat *FHEM* serveru na stav zařízení určuje parametr `fhem.refreshTime`. Maximální dobu, po kterou se bude čekat na odpověď od *FHEM* serveru lze konfigurovat pomocí parametru `fhem.receiveTimeout`. Pomocí parametru `fhem.reconnectTime` lze definovat doba, po které se má **FHEMClient** znovu zkusit připojit k *FHEM* serveru, když se připojení k *FHEM* serveru nezdařilo. Ukázka konfigurace modulu je dostupná ve výpisu 6.3.

```
[conrad]
enable = yes
fhem.address = 127.0.0.1:7072
fhem.refreshTime = 5
fhem.receiveTimeout = 2
fhem.reconnectTime = 5
```

Výpis 6.3: Konfigurace Conrad modulu

6.3 Překlad a spuštění

O překlad softwaru *BeeeOn* brány se stará nástroj **cmake**[2], který na základě textového souboru *CMakeLists.txt* vygeneruje soubory *Makefile*. Tyto soubory jsou poté použity pro samostatný překlad jednotlivých modulů *BeeeOn* brány a následné spojení do jednoho spustitelného binárního souboru. Pro překlad nově vytvořených modulů a zdrojových souborů byl soubor *CMakeLists.txt* o ně rozšířen. V rámci překladu je využita funkce nástroje **cmake**, která dovoluje konfigurovat jaké moduly se mají přeložit a jaké nikoliv. Každý povolený modul *BeeeOn* brány je přeložen a je z něj vytvořena statická knihovna. Tyto statické knihovny jsou poté propojeny do binárního souboru. Aby byl překlad nových modulů úspěšný, je nutné mít správně nainstalované knihovny a nástroje zmíněné v sekci 6.1.

Pro spuštění nových modulů je nutné rozšířit konfigurační soubory *BeeeOn* brány. První sada konfiguračních souborů udává strukturu softwaru brány. Tyto konfigurační soubory přijímá poskytovatel závislosti, který na základě nich vytvoří vlákna a instanciuje povolené služby, kterým předá nakonfigurované parametry. Ukázka těchto konfiguračních souborů je dostupná ve výpisu 2.4. Druhá sada konfiguračních souborů obsahuje hodnoty jednotlivých konfiguračních parametrů modulů. V rámci sekce zabývající se implementací jednotlivých modulů je dostupná ukázka těchto konfiguračních souborů.

[2] https://cmake.org

Kapitola 7

Testování BeeeOn brány

Obsahem kapitoly je způsob testování nově vytvořených modulů do softwaru *BeeeOn* brány, ale také způsob ověření správnosti chování *BeeeOn* brány jako celku. Testování *BeeeOn* brány se provádí pomocí jednotkových testů a integračních testů. Jednotkové testy z pravidla ověřují správnost výstupů metod jednotlivých tříd na základě jejich vstupu. Zatímco integrační testy testují správnost chování celé *BeeeOn* brány. Spolčený rys integračních testů je spuštění softwaru *BeeeOn* brány a testování chování konkrétní části brány (například jeden specifický manažer zařízení). Pro automatické spuštění veškerých testů, jak jednotkových, tak integračních je využit open-source automatizační server *Jenkins*[1].

Kromě implementace jednotkových a integračních testů bylo také provedeno pilotní testování *BeeeOn* brány. *BeeeOn* brána byla instalována do domácnosti, která byla vybavena inteligentními zařízeními podporovanými systémem *BeeeOn*. Uživatelé domácnosti pomocí systému *BeeeOn* ovládali osvětlení, spotřebiče připojené k inteligentním zásuvkám, okna pomocí inteligentního otvírače oken a získávali informace o kvalitě ovzduší v domácnosti. Pilotní testování bylo prováděno po dobu jednoho měsíce, přičemž *BeeeOn* brána fungovala korektně.

7.1 Jednotkové testy

V rámci vývoje softwaru *BeeeOn* brány jsou také vytvářeny jednotkové testy. Pro zjednodušení implementaci jednotkových testů je využit framework *CppUnit*. Testy jsou vytvářeny pro jednotlivé moduly nebo třídy. Jednotkou testovanou těmito testy je obvykle implementace metody třídy. Účelem jednotkových testů je ověřit správnost výstupu a chování testovaných metod tříd.

V rámci nové vytvořených modulů jsou pro každý modul vytvořeny jednotkové testy, které ověřují správnost transformace získaných naměřených dat ze zařízení na exportována senzorová data. To znamená, že jsou testovány třídy reprezentující zařízení, která mají tuto transformaci na starost. Takto jsou otestovány třídy **RevogiSmartPlug**, **RevogiSmartLite**, **RevogiSmartCandle**, **SonoffSC**, **PowerMeterSwitch**, **RadiatorTermostat** a **WirelessShutterContact**. Celkově bylo vytvořeno 14 nových jednotkových testů.

[1] https://www.jenkins.io

7.2 Integrační testy

O ověření správnosti chování modulů, vnitřní komunikace mezi moduly brány, komunikace modulů s jinými aplikacemi a komunikačních kanálů, se starají integrační testy. Pro implementaci integračních testů je použit programovací jazyk *Python3* s frameworkem pro jednotkové testy *unittest*. Integrační testy pro software *BeeeOn* brány byly navrženy a implementovaný v rámci diplomové práce Richarda Wolferta [8]. Tyto integrační testy jsou rozšířeny o testy modulů brány, které nebyly pokryty původními testy.

Architektura integračních testů je rozdělena do modulů, kde každý modul implementuje určitou funkcionalitu. Základním modulem je **GatewayBaseModule**, který dědí testovací třídu frameworku **TestCase**, implementující spuštění testovací instance softwaru *BeeeOn* brány. Všechny další moduly dědí tento modul, protože všechny testy očekávají běžící instanci softwaru *BeeeOn* brány, na které ověřují výsledek různých příkazů s očekávaným výsledkem.

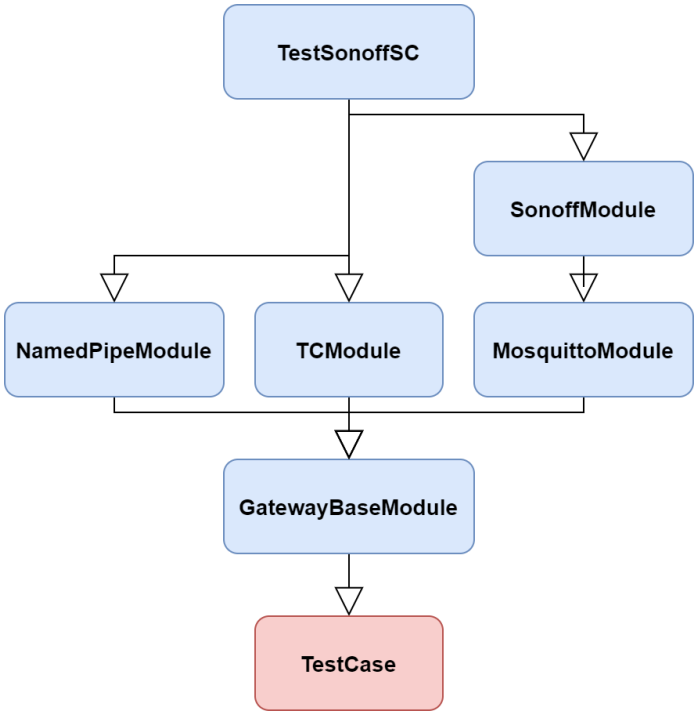
Pro získávání exportovaných dat *BeeeOn* bránou existuje modul **NamedPipeModule**, který data získává pomocí pojmenované roury. Tento modul také upravuje konfigurační soubor, s kterým je *BeeeOn* brána spuštěna, tak aby byl spuštěn exportér **NamePipeExporter** a měl nastavenou cestu k pojmenované rouře stejně jako testovací modul.

Modul **TCModule** zajišťuje vytvoření spojení s testovacím centrem brány a odesílání příkazů na bránu. Modul také modifikuje konfiguraci *BeeeOn* brány a to povolením testovacího centra a nastavení portu, na kterém testovací centrum přijímá nová spojení.

Posledním důležitým modulem, v rámci rozšiřovaní integračních testů, je **MosquittoModule**, který zajišťuje spuštění instance *MQTT* brokeru, jenž je využit pro komunikaci brány se zařízeními.

Ukázka architektury integračních testů s jedním konkretním testem, ověřujícím správnost práce *BeeeOn* brány s *Sonoff* multisenzorem, je dostupná na obrázku 7.1. Jak je i z obrázku níže vidět, v rámci testování modulů *BeeeOn* brány vždy existuje testovací modul, který daný testovací modul povoluje, konfiguruje a případně spouští další potřebné služby pro testování. To znamená, že v rámci testovaní *Sonoff* modulu existuje testovací modul **SonoffModule**. Tento modul povoluje spuštění manažera zařízení, který řídí *Sonoff* zařízení, a umožňuje spustit emulátory *Sonoff* zařízení. Koncový test **TestSonoffSC** implementuje testované příkazy a ověřuje správnost práce *Sonoff* modulu se *Sonoff* multisenzorem. Testovací třídy dědí ty testovací moduly, jejíž funkcionalitu vyžaduje pro testování.

Do původní architektury integračních testů bylo nutné přidat modul, který zajišťuje spuštění *MQTT* brokeru, jenž je využíván v rámci testování modulů pro podporu zařízení od výrobců *Sonoff* a *Vektiva*. Dále byly integrační testy rozšířeny o moduly implementující testované príkazy pro ověření správnosti práce *BeeeOn* brány se zařízeními od *Philips Hue*, *HomeMatic*, *Sonoff* a *Vektiva*. Ve výsledku byly integrační testy rozšířeny o 27 nových testů.



Obrázek 7.1: Architektura integračních testů

7.2.1 Emulátory zařízení

Pro účely automatického testování manažerů zařízení byly vytvořeny emulátory zařízení, které umožňují testovat manažery zařízení bez fyzické přítomnosti podporovaných zařízení. Emulátory zařízení jsou vytvořeny pouze pro ty zařízení, jejíž komunikace lze jednoduše emulovat. Pro emulátory je použit programovací jazyk *Python3*.

Emulátory zařízení napodobují chování a aplikační rozhraní fyzického zařízení. Dokáží odpovídat na požadavky *BeeeOn* brány nebo periodicky generovat senzorová data a ty odesílat do sítě. Emulátory existují pro moduly:

- **Belkin WeMo** – pro inteligentní zásuvku (`bewemuSwitch`), stmívač (`bewemuDimmer`) a *Belkin WeMo Link* (`bewemuLink`), který spravuje inteligentní žárovky.
- **Philips Hue** – pro *Philips Hue Bridge* (`phuemuBridge`) spravující inteligentní žárovky.
- **Vektiva** – pro inteligentní otvírač oken (`smarwiEmulator`).
- **Sonoff** – pro inteligentní multisenzor (`sonoffscEmulator`).
- **HomeMatic** – pro *FHEM* server (`fhemEmulator`), který umožňuje emulovat inteligentní zásuvku a okenní magnetický kontakt.

7.2.2 Testované příkazy

Testované příkazy, společné pro všechny moduly spravující koncová zařízení, jejíž účel je ověřit správnost zpracování příkazů manažerem zařízení, jsou:

1. Odeslání příkazu `listen` na *BeeeOn* bránu. V rámci toho testovaného příkazu se ověřuje korektní zpracování příkazu `listen` manažerem zařízení.

2. Odeslání dvou po sobě jdoucích příkazu `listen`. Tento příkaz ověřuje, zda manažer zařízení korektně zpracuje druhý příkaz `listen`. Manažer zařízení by měl druhý příkaz `listen` ignorovat v případě, že délka objevování je kratší než doba, po kterou má ještě manažer zařízení provádět objevování nových zařízení.

3. Odeslání příkazu `device-accept` na napárování neexistujícího zařízení. Manažer zařízení v takové situaci má vykonávání příkazu `device-accept` ukončit neúspěchem.

4. Odeslání příkazu `unpair` na odpárování neexistujícího zařízení. Příkaz `unpair` pro neexistující zařízení má manažer zařízení vyhodnotit úspěšně.

5. Odeslání příkazu `set-value` na nastavení hodnoty modulu neexistujícího zařízení. Zpracování příkazu `set-value` pro neexistující zařízení má manažer zařízení ukončit chybou.

Na modulech, pro které existují emulátory zařízení, jsou kromě výše zmíněných testovaných příkazů také testovány následující příkazy:

1. Odeslání příkazu `listen` na *BeeeOn* bránu. Manažer zařízení má nalézt zařízení emulované emulátorem na síti.

2. Odeslání příkazu `device-accept` na napárování nalezeného zařízení v předchozím bodě. Manažer zařízení má korektně napárovat zařízení a v případě, že zařízení periodicky odesílá senzorová data nebo manažer zařízení se periodicky dotazuje zařízení na jeho stav, má brána exportovat senzorová data.

3. Odeslání příkazu `set-value` na nastavení hodnoty modulu napárovaného zařízení. Tento testovaný příkaz se provádí v případě, že emulátor emuluje zařízení, které disponuje aktorovým modulem. V této situaci má manažer zařízení úspěšně provést přijatý příkaz a exportovat senzorová data, která potvrzují nastavenou hodnotu.

4. Odeslání příkazu `unpair` na odpárování zařízení. Manažer zařízení po úspěšném odpárování zařízení nemá povolit nastavení hodnoty modulu odpárovaného zařízení a také nemá exportovat žádná data spojená s odpárovaným zařízením.

Tyto příkazy pokrývají integrační testy pro moduly spravující *Belkin WeMo* zařízení, *Philips Hue* zařízení, *Vektiva* zařízení, *Sonoff* zařízení, *HomeMatic* zařízení, virtuální zařízení a interní tlakový senzor.

Moduly, pro které neexistují emulátory zařízení byly a jsou testovány manuálně s přítomnosti fyzického zařízení na síti. Mezi tyto moduly patří moduly pro správu *Bluetooth Low Energy* zařízení, *ZWave* zařízení, *Jablotron* zařízení a *IQRF* zařízení. V rámci manuálního testování těchto modulů jsou prováděné stejné příkazy, jaké provádí automaticky integrační testy.

Kapitola 8

Závěr

Primárním cílem této diplomové práce bylo rozšířit software *BeeeOn* brány, která je součástí systému *BeeeOn* o zařízení od výrobců *Revogi*, *Tabu Lumen*, *Sonoff* a *HomeMatic*. Dalším cílem bylo software *BeeeOn* brány otestovat a ověřit správnost fungování a správnost komunikace jednotlivých modulů s koncovými zařízeními. Všechny stanovené cíle diplomové práce byly splněny.

První část práce se zaměřuje na popis systému *BeeeOn* s důrazem na software *BeeeOn* brány, který je hlavním předmětem práce. V této části je analyzována *BeeeOn* brána spolu s popisem architektury a jednotlivých komponent softwaru brány. Další část práce analyzuje zařízení, o jejíž podporu je software *BeeeOn* brány rozšiřován. V této části je dostupná charakteristika komunikačních technologií, které využívají analyzovaná zařízení. Část také obsahuje popis chování a popis komunikace jednotlivých zařízení. Následně je rozebrán návrh a implementace modulů pro řízení zařízení od zmíněných výrobců. Poslední část práce je zaměřena na testování vytvořených modulů a celého softwaru *BeeeOn* brány.

Ve výsledku tato práce rozšířila software *BeeeOn* brány o tři nové moduly, které rozšířily množinu podporovaných zařízení systémem *BeeeOn* o osm zařízení komunikujících pomocí tří různých technologií. Systém *BeeeOn* nově umožňuje sbírat data o kvalitě ovzduší pomocí multisenzoru *Sonoff SC*, řídit teplotu v domácnosti pomocí *HomeMatic* termostatické hlavice a ovládat osvětlení a jiné spotřebiče pomocí inteligentních žárovek a zásuvek od *Revogi*, *Tabu Lumen* a *HomeMatic*. Vytvořena rozšíření do systému *BeeeOn* jsou také využita v projektu *Zabezpečená brána pro internet věcí (SIoT)*[1]. *BeeeOn* brána poskytuje data o senzorových sítích, nad kterými provádí analýzu a detekci bezpečnostních událostí.

Součásti výsledku práce je také otestování softwaru *BeeeOn* brány. V rámci testování brány byly rozšířeny jednotkové testy a integrační testy softwaru *BeeeOn* brány, které testují, jak nově vytvořené moduly, tak již existující, pro které však integrační testy chyběly. Pro automatizaci jednotkových a integračních testů byl využit automatizační server *Jenkins*. Nad softwarem *BeeeOn* brány bylo také provedeno pilotní testování, jenž probíhalo v reálné domácnosti vybavenou zařízeními podporovanými systémem *BeeeOn*.

Nově vytvořené moduly softwaru *BeeeOn* brány pro podporu zařízení byly navrhnuty a implementovaný s ohledem na jednoduché rozšíření o podporu nových zařízení. Všichni výrobci inteligentních zařízení pro chytrou domácnost, od kterých byly v této práci zařízení analyzována, nabízí další zařízení, která můžou být dále v rámci projektu analyzována a může pro ně být vytvořena podpora. Další možné pokračování projektu může být ve vytvoření emulátorů zařízení pro moduly, pro které neexistují, a rozšíření integračních testů.

[1] https://starfos.tacr.cz/cs/project/VI20172020079

Literatura

[1] GmbH, A. I. S. E. a Contributors. *A Guided Tour Of The POCO C++ Libraries* [online]. 2020 [cit. 2020-02-19]. Dostupné z: https://pocoproject.org/docs/00100-GuidedTour.html.

[2] Jaffey, T. *MQTT and CoAP, IoT Protocols* [online]. 2014 [cit. 2019-12-27]. Dostupné z: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php.

[3] König, R. *FHEM* [online]. 2020 [cit. 2020-02-18]. Dostupné z: https://fhem.de.

[4] Microchip Technology, I. *Introduction to Bluetooth® Low Energy* [online]. 2020 [cit. 2019-12-10]. Dostupné z: https://microchipdeveloper.com/wireless:ble-introduction.

[5] Project, T. G. *Referenční příručka k API* [online]. 2014 [cit. 2018-04-18]. Dostupné z: https://developer.gnome.org/references.

[6] Team, T. H. *Quality of Service 0,1 & 2 - MQTT Essentials: Part 6* [online]. 2015 [cit. 2019-12-27]. Dostupné z: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/.

[7] TOWNSEND, K. *Introduction to Bluetooth Low Energy* [online]. 2019 [cit. 2019-12-10]. Dostupné z: https://cdn-learn.adafruit.com/downloads/pdf/introduction-to-bluetooth-low-energy.pdf.

[8] Wolfert, R. *Automatické testování systému BeeeOn*. Brno, CZ, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/21069/.

[9] Yuan, M. *Getting to know MQTT* [online]. 2020 [cit. 2019-12-27]. Dostupné z: https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/.

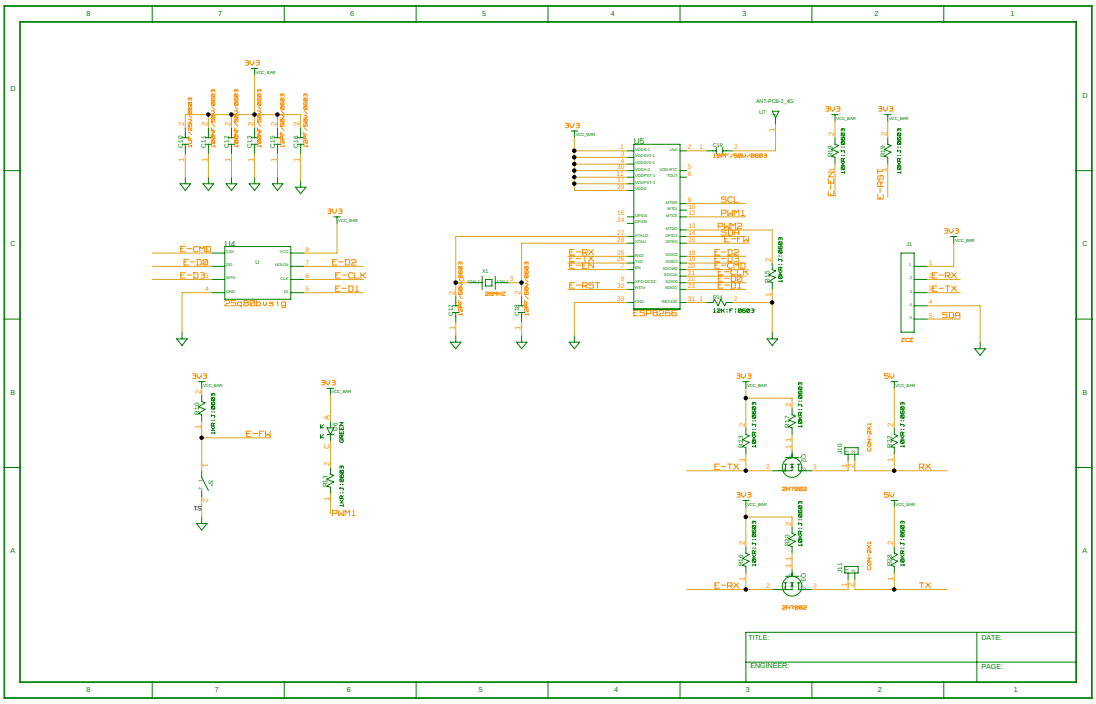
Příloha A

Obsah přiloženého paměťového média

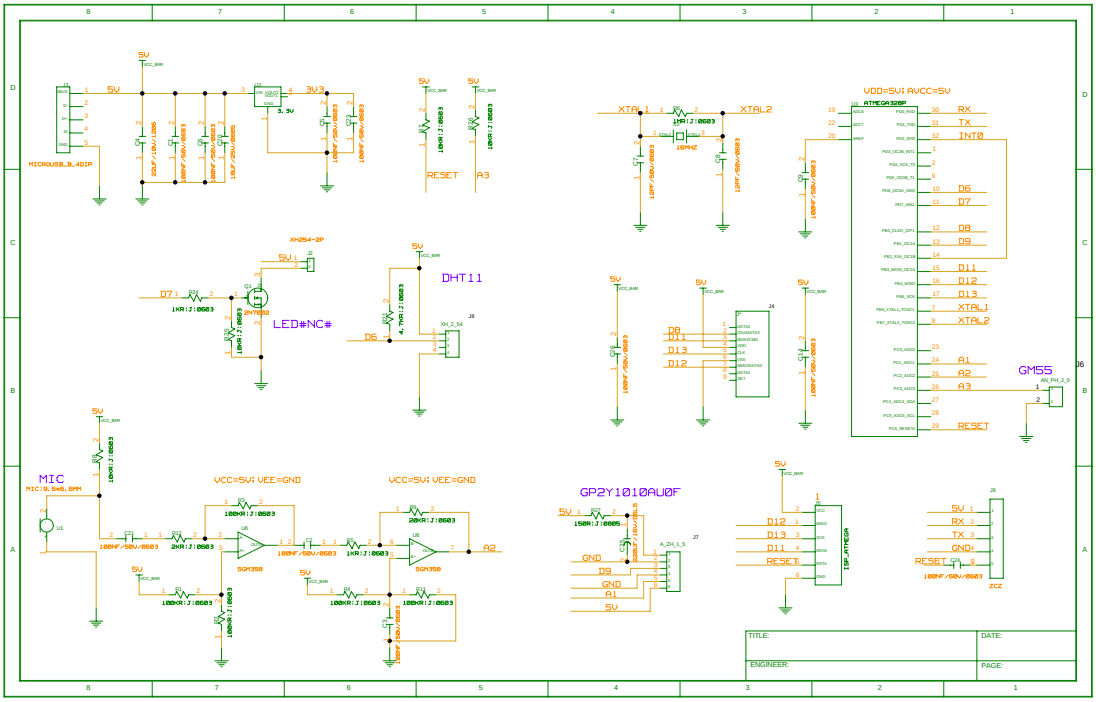
- **createdFiles/** – vytvořené soubory v rámci diplomové práce (nepřeložitelné)
- **original/** – zdrojové kódy softwaru BeeeOn Gateway, testy softwaru BeeeOn Gateway a firmware pro multisenzor Sonoff SC
- **tex/** – zdrojové soubory textu diplomové práce
- **CREATED** – seznam vytvořených/rozšířených souborů v rámci diplomové práce
- **README** – instrukce pro překlad a spuštění zdrojových kódů
- **xbedna62.pdf** – text diplomové práce

Příloha B

Schéma Sonoff SC



Obrázek B.1: Schéma multisenzoru Sonoff SC - část 1.



Obrázek B.2: Schéma multisenzoru Sonoff SC - část 2.

Příloha C

Šifrovací algoritmus Tabu Lumen žárovky

```
ADD_KEY = [
    0x00, 0xf4, 0xe5, 0xd6, 0xa3, 0xb2, 0xa3, 0xb2, 0xc1,0xf4,
    0xe5, 0xd6, 0xa3, 0xb2, 0xc1, 0xf4, 0xe5, 0xd6, 0xa3, 0xb2
]
XOR_KEY = [
    0x00, 0x2b, 0x3c, 0x4d, 0x5e, 0x6f, 0xf7, 0xe8, 0xd9, 0xca,
    0xbb, 0xac, 0x9d, 0x8e, 0x7f, 0x5e, 0x6f, 0xf7, 0xe8, 0xd9
]
c = 0

for j in range(len(data) - 1, 0):
    k~= c + data[j] + ADD_KEY[j]
    if k~>= 256:
        c = k~>> 8
        k~-= 256
    else:
        c = 0
    data[j] = k

for i from in range(0, len(data):
    data[i] = data[i] ^ XOR_KEY[i]
```

Výpis C.1: Šifrovací algoritmus

Příloha D

Konfigurace FHEM serveru

```
define telnetPort telnet 7072 global
setuuid telnetPort 5d318b76-f33f-7c84-c847-34f264e70f2bc9ed

define CUL_0 CUL /dev/ttyACM0@9600 1034
setuuid CUL_0 5d318b79-f33f-7c84-bd65-4cca529366d903ad
attr CUL_0 hmId F11034
attr CUL_0 rfmode HomeMatic
```

Výpis D.1: Část konfiguračního souboru pro *FHEM*