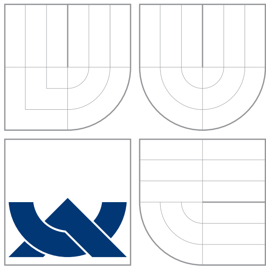
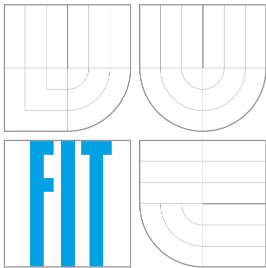


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



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

ROZŠIŘITELNÝ SYSTÉM PRO AUTOMATIZACI DOMÁCNOSTÍ

EXTENSIBLE HOME AUTOMATION SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN NOVÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PAVEL VAMPOLA

BRNO 2016

Zadání bakalářské práce/18896/2015/xnovak1c

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Novák Martin**
Obor: Informační technologie
Téma: **Rozšiřitelný systém pro automatizaci domácností**
Extensible Home Automation System
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s problematikou Internetu věcí a se systémem inteligentní domácnosti vyvíjeným na FIT VUT.
2. Identifikujte příležitosti pro automatizaci řízení domácností tímto systémem, zaměřte se na problematiku šetření energie a zvyšování uživatelského komfortu.
3. Navrhněte snadno rozšiřitelný systém pro integraci automatizačních úloh do serverové části systému inteligentní domácnosti.
4. Navržený systém implementujte, integrujte do systému a ověřte na několika různých automatizačních úlohách.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1, 2 a 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Vampola Pavel, Ing.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
L.S.
612 66 Brno, Božetěchova 2

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá návrhem a vývojem systému, jehož cílem je v projektu inteligentní domácnosti BeeeOn umožnit automatizované zpracování dat získávaných z domácností. Práce popisuje architekturu projektu, do kterého má být systém integrován. Je v ní identifikováno několik příležitosti pro automatizaci, jež slouží k zvyšování komfortu uživatele a k snižování energetické náročnosti domácností. Na jejich základě je navržen rozšiřitelný systém, který umožňuje tyto automatizace realizovat a spravovat na BeeeOn serveru. Práce rovněž obsahuje popis implementace tohoto systému a několika konkrétních automatizací. Nakonec je přiblíženo, jak byl systém testován, integrován na cílový serverový počítač a je popsáno několik návrhů pro budoucí rozšíření tohoto systému.

Abstract

This bachelor thesis is focused on design and development of a system for a project smart home BeeeOn that aims to enable automated processing of data acquired from homes. The thesis describes the project's architecture to which the system should be integrated. It identifies several opportunities for automation to increase user comfort and to reduce the energy demands of households. Based on them is designed extensible system enabling implementation and management of the automations on BeeeOn server. This thesis also focuses on implementation of this system and several specific automations. At the end is described the testing of the system, its integration to the target server, and several designs of the system's future extensions.

Klíčová slova

automatizace, BeeeOn, inteligentní domácnost, Internet věcí, rozšiřitelný systém

Keywords

automation, BeeeOn, smart home, Internet of Things, extensible system

Citace

NOVÁK, Martin. *Rozšiřitelný systém pro automatizaci domácností.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Vampola Pavel.

Rozšiřitelný systém pro automatizaci domácností

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Pavla Vampoly. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

....
Martin Novák
16. května 2016

Poděkování

Tímto bych chtěl poděkovat vedoucímu této práce panu Ing. Pavlu Vampolovi za jeho čas a ochotu a Ing. Viktoru Pušovi, Ph.D. za jeho cenné rady při psaní této práce.

© Martin Novák, 2016.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod **3**

2 Systém BeeeOn **5**
2.1 Architektura . . . 5
2.2 Koncová zařízení . . . 6
2.3 Brána (Adaptér) . . . 6
2.4 Server . . . 7
2.5 Uživatelská rozhraní . . . 8

3 Automatizace řízení domácností **9**
3.1 Příležitosti pro automatizaci . . . 9
3.2 Automatizační úlohy . . . 11
3.3 Návrh konkrétních automatizačních úloh . . . 13

4 Návrh rozšiřitelného systému **15**
4.1 Požadavky na rozšiřitelný systém . . . 15
4.2 Vnitřní struktura systému . . . 16
4.3 Rozšíření databáze . . . 20

5 Implementace rozšiřitelného systému **21**
5.1 Použité technologie . . . 21
5.2 Reprezentace automatizačních úloh . . . 23
5.3 Řídicí komponenty . . . 26
5.4 Komunikační rozhraní . . . 29

6 Implementace konkrétních automatizačních úloh **32**
6.1 Obecný hlídač (Watchdog) . . . 32
6.2 Kontrola živosti brány a koncových zařízení (AliveCheck) . . . 33

7 Testování a integrace **35**

8 Závěr **37**

Literatura **38**

Přílohy **39**
Seznam příloh . . . 40

A Obsah CD **41**

B Schéma databáze **42**

C Ukázky komunikačních protokolů **43**

D Ukázky konfiguračních souborů **44**

E Ukázky uživatelských požadavků **45**

Kapitola 1

Úvod

V uplynulých několika desetiletích *Internet* exponenciálně narostl z malé výzkumné sítě o několika málo uživatelích na celosvětovou síť, která ke konci roku 2015 slouží více než třem miliardám uživatelů[1].

S další miniaturizací a snižováním ceny elektronických zařízení je možné *Internet* rozšířit do další oblasti — *chytrých věcí*. Tento pojem neznamená nic jiného než rozšíření běžných věcí o malá elektronická zařízení. Ta jim poskytují funkcionalitu k získávání a zpracování informací z jejich okolního prostředí a připojení k síti *Internet*. Takové zařízení poté slouží jako propojení fyzického světa se světem informačních technologií. V několika posledních letech získaly sítě sdružující tato zařízení označení *Internet věcí*.

Příkladem *chytré věci* může být například chytrá lednice, která kromě běžné činnosti, jakou je uchování potravin v chladu, dokáže také identifikovat uskladněné potraviny a jejich množství. Poté může v případě, že množství určitého zboží kleslo pod uživatelem nastavenou hranici, takové zboží přes *Internet* objednat v nejbližším obchodě.

Vývoji *Internetu věcí* se věnují na Fakultě informačních technologií VUT v Brně v open-source projektu *BeeeOn* členové Výzkumné skupiny akcelerovaných síťových technologií, zkráceně ANT (Accelerated Network Technologies). Cílem systému vyvíjeném v rámci tohoto projektu je vytvořit modulární, snadno rozšiřitelný, uživatelsky přívětivý a bezpečený systém pro řízení a automatizaci domácností.

Předmětem této práce je navrhnout a implementovat rozšiřitelný systém, který přidá do systému *BeeeOn* možnost automatizovaného zpracování informací získaných z fyzického světa a také možnosti jak na ně reagovat. Příklady takových automatizačních případů jsou například sepnutí větrání, když je v domácnosti naměřena příliš vysoká vlhkost, nebo varování uživatele, když je detekován vysoký atmosférický tlak.

Navrhovaný systém by měl zajistit jednotné prostředí pro vývoj, správu a řízení většího množství automatizačních úloh, které umožní realizaci různých automatizačních případů. Tento systém by měl být snadno rozšiřitelný o další automatizační úlohy a uživateli by mělo být umožněno jejich spouštění a konfigurace podle potřeb jeho vlastní domácnosti.

V následující kapitole této práce s číslem 2 je přiblížena architektura systému inteligentní domácnosti *BeeeOn*. Kapitola číslo 3 je zaměřena na popis různých příležitostí automatizace domácnosti. V této kapitole je také navrženo několik konkrétních automatizačních úloh, které slouží k ověření správného fungování rozšiřitelného systému. V kapitole s číslem 4 je popsán návrh rozšiřitelného systému a všech jeho komponent. Následující kapitola číslo 5 se věnuje tomu, jak byl navržený rozšiřitelný systém implementován. Kapitola

[1] World Internet User Statistics - http://www.internetworldstats.com/stats.htm

číslo 6 si klade za cíl popsat implementaci navržených konkrétních automatizačních úloh. V kapitole s číslem 7 je obsaženo jak byl výsledný rozšiřitelný systém testován a následně integrován do serverové části systému *BeeeOn*. Poslední kapitola číslo 8 je věnována dosaženým výsledkům, shrnutí této práce a návrhům budoucí možné funkcionality pro rozšiřitelný systém.

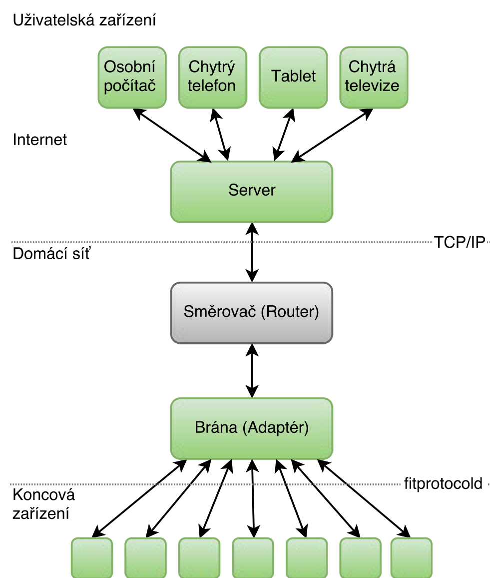
Kapitola 2

Systém BeeeOn

Tato kapitola se věnuje popisu architektury systému inteligentní domácnosti *BeeeOn* a všech jeho součástí, tak jak byl realizován na jaře roku 2016.

2.1 Architektura

Architektura systému *BeeeOn*, zobrazená v obrázku číslo 2.1, se skládá z koncových zařízení (složených ze senzorů a aktuátorů) na nejnižší vrstvě. Ta jsou poté bezdrátově připojena k bráně, která jim umožňuje komunikovat se zbytkem systému. Na další vrstvě je brána připojena ke směrovacímu zařízení, přes které je sítí *Internet* spojena se serverem. Se serverem následně komunikují uživatelská zařízení, na něž jsou implementována uživatelská rozhraní k ovládání systému.



Obrázek 2.1: Architektura systému BeeeOn

2.2 Koncová zařízení

Koncová zařízení jsou fyzická zařízení, bezdrátově připojitelná a komunikující s bránou. Tato zařízení se skládají z jednoho či více senzorů a aktuátorů. V rámci koncového zařízení jsou pak všechny senzory a aktuátory, ze kterých se skládá, nazývány souhrnně jako moduly.

Senzory slouží k získávání dat z okolního prostředí. Měřená data jsou většinou fyzikální veličiny jako teplota, vlhkost, tlak vzduchu apod. Uživatel si může sám v uživatelském rozhraní systému určit, v jakém časovém intervalu má koncové zařízení měřit senzorická data a zasílat je bráně.

Aktuátory jsou součásti koncových zařízení sloužící k nastavování nebo spínání jiných zařízení a tak umožňují systému fyzicky ovlivňovat okolní prostředí. Aktuátorem tedy může být například spínač elektrické zásuvky, ovladač domovního osvětlení čí řízení centrálního topení nebo klimatizace.

2.2.1 Konkrétní koncová zařízení

V rámci projektu *BeeeOn* bylo navrženo a vyrobeno koncové zařízení *BeeeOn senzor*, jenž po připojení do systému dokáže měřit teplotu na interním senzoru, teplotu na externím senzoru a vlhkost okolního vzduchu.

Brána také obsahuje přímo na své základní desce koncové zařízení, jehož součástí je jeden senzor měřící atmosférický tlak. I když je toto zařízení přímo fyzicky obsažené v bráně, tak aby mohlo zasílat bráně senzorická data, musí se k ní rovněž nejdříve připojit.

V době psaní této práce také pracují někteří členové týmu *BeeeOn* na integraci koncových zařízení a implementaci komunikačních protokolů třetích stran. Jmenovitě třeba zařízení firem *Jablotron*, *Conrad* a *Homematic* a protokoly *Z-Wave* a *IQRF*.

2.3 Brána (Adaptér)

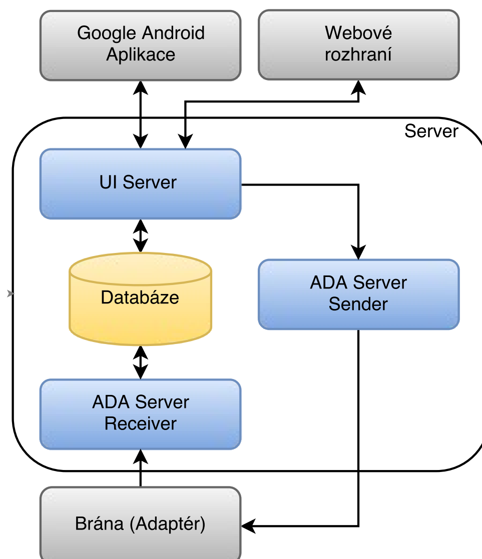
Brána — historicky v projektu nazývána také jako adaptér — je fyzické zařízení připojené k *Internetu*, které za pomoci protokolů TCP/IP komunikuje se serverem. Slouží převážně ke sběru senzorických dat z připojených koncových zařízení, k jejich přeposílání na server a k přijímání požadavků ke změně stavů aktuátorů na připojených koncových zařízeních.

Ke každé bráně je možné připojit libovolný počet koncových zařízení za pomoci jejich jednoznačných identifikátorů. Pokud by chtěl uživatel již připojené koncové zařízení připojit k jiné bráně, tak jej nejdříve musí od původní brány odpojit. Není tudíž možné, aby jedno koncové zařízení zasílalo svá senzorická data více branám a přijímalo od nich požadavky ke změně stavů aktuátorů.

Hardware brány je založen na open-hardware základní desce od firmy *Olimex* s procesorem architektury ARM. Jako operační systém byl zvolen GNU/Linux, hlavně pro jeho velkou podporu na platformě, univerzálnost a rozšiřitelnost.

2.4 Server

Server je výkonný počítač s architekturou x86-64 a operačním systémem s jádrem GNU/Linux, dostupný na veřejné IP adrese v síti *Internet*. Funguje jako mezivrstva zprostředkovávající komunikaci mezi uživatelskými zařízeními a bránami.



Obrázek 2.2: Aktuální schéma serveru

Jedním z jeho účelů je také datové úložiště. K tomuto účelu je na serveru umístěn systém řízení báze dat *PostgreSQL*, s nímž komunikují některé služby serveru. V této databázi jsou uchovávána všechna data potřebná pro fungování systému *BeeeOn*. Schéma základních tabulek databáze se nachází v příloze B.1.

Na serveru jsou spuštěny (kromě databáze) také dvě hlavní služby, které momentálně zajišťují chod celého serveru. Tyto služby, jež mezi sebou komunikují pomocí unixových soketů, jsou popsány v následujících dvou sekcích:

2.4.1 ADA Server

Tato serverová služba slouží k zajištění komunikace s připojenými bránami. V momentě, kdy se brána k ADA Serveru připojí, je uchováno aktivní spojení, aby mohla komunikace probíhat i ze serveru směrem k bráně. Tato funkcionalita je nutná, protože kdyby brána byla v síti využívající technologii NAT[1], mohlo by být znemožněno zasílání požadavků sloužících ke změnám stavů aktuátorů.

Jak je viditelné v obrázku schématu serveru 2.2, služba je sémanticky rozdělitelná na dvě části, jež na serveru běží jako samostatné procesy:

ADA Server Receiver funguje jako serverová aplikace a pasivně čeká na příjem senzorických dat od bran. Tato přijatá data poté ukládá do příslušných tabulek v databázi, aby je později bylo možné zobrazit na uživatelských zařízeních.

ADA Server Sender má opačnou funkci. Slouží převážně k zasílání požadavků ke změně stavů aktuátorů na koncových zařízeních bránám nebo ke změně intervalů zasílání

[1] Network Address Translation

senzorických dat.

V době psaní této práce se současně pracuje na rozšíření komunikačního protokolu mezi *ADA Serverem* a bránami, aby bylo možné ze serveru přímo zjišťovat informace o bráně. Bude tak například možné ověřit dostupnost brány v síti *Internet*.

2.4.2 UI Server

Tato serverová služba zajišťuje komunikaci serveru a uživatelských rozhraní na zařízeních uživatelů. Využívá protokolu založeného na jazyce XML[1], jenž má definované zprávy sloužící k ovládání systému *BeeeOn* a získávání zobrazitelných dat z databáze.

Do budoucna se očekává postupné nahrazení současného protokolu jeho alternativou v jazyce JSON[2] a nakonec nahrazení celé služby *UI Server* službou využívající ke svému fungování technologii REST API[3] [6]. K této změně dochází hlavně kvůli budoucímu vývoji dalších uživatelských rozhraní pro jiné platformy. Pro ty je sjednocení komunikace, které právě technologie REST API nabízí, přívětivější než aktuální forma *UI Serveru*.

2.5 Uživatelská rozhraní

Pro uživatele systému *BeeeOn* bylo vyvinuto několik uživatelských rozhraní ve formě aplikací pro uživatelská zařízení. Pro všechna rozhraní je společné, že uživateli umožňují provádět základní operace s jím vlastněnými bránami a koncovými zařízeními. Umožňují mu také zobrazovat koncovými zařízeními naměřená senzorická data a měnit stavy připojených aktuátorů.

2.5.1 Google Android

Aplikace pro převážně mobilní zařízení s operačním systémem *Google Android* je momentálně hlavním uživatelským rozhraním pro ovládání *BeeeOn* systému. Kromě základní funkcionality popsané výše je cílem této aplikace poskytnout také uživateli:

- Přizpůsobitelný ovládací panel, který mu dá rychlý přehled nad jeho *BeeeOn* systémem.
- Kompletní seznam všech jeho bran a připojených koncových zařízení.
- Lokalizaci do několika jazyků (angličtina, čeština a slovenština).

2.5.2 Webové rozhraní

Webové rozhraní je prozatím pouze ve vývojové fázi a není ještě dostupné širší veřejnosti. Klade si za cíl poskytnout uživateli totožnou funkcionalitu a možnosti jako *Google Android* aplikace. Jeho vzhled a rozložení jsou navrženy tak, aby korespondovaly s mobilní aplikací, byly modulární a co nejvíce uživatelsky přívětivé.

[1] Extensible Markup Language
[2] JavaScript Object Notation
[3] Representational State Transfer Application Programming Interface

Kapitola 3

Automatizace řízení domácností

Jak bylo popsáno v předchozí kapitole, systém *BeeeOn*, tak jak je aktuálně realizován, slouží hlavně ke sběru a uchovávání senzorických dat a ke zpracování změn stavů aktuátorů vyvolaných přímo uživatelem. Systému však zatím chybí schopnost automatizovaně analyzovat uchovaná senzorická data nebo přímo na základě získaných naměřených dat provádět nějakou specifickou činnost. Tato kapitola se proto věnuje způsobům, jakými lze některé činnosti v systému automatizovat a následně také realizovat.

3.1 Příležitosti pro automatizaci

V systému *BeeeOn* bylo definováno několik druhů senzorů, které se liší převážně měřenými fyzikálními veličinami. Ačkoliv ještě nejsou v systému fyzicky dostupné všechny definované senzory, počítá se s jejich brzkou integrací v rámci rozšiřování *BeeeOn* architektury o koncová zařízení třetích stran.

Následující výčet se zaměřuje na stručný popis vlivů měřených fyzikálních veličin na komfort uživatelů a na energetickou náročnost domácnosti. U každé fyzikální veličiny je také navrženo několik případů, jak by se dané veličiny daly využít k automatizovanému řízení domácnosti. Je také uvažováno, že k elektronickým zařízením, která slouží jako aktivní prvky k přímému ovlivňování uživatelského komfortu a energetické náročnosti, budou připojeny aktuátory, které umožňují jejich spuštění nebo ovládání.

- **Teplota vzduchu** – Teplota v domácnosti je do jisté míry individuální záležitost, protože každý člověk vnímá teplotu vzduchu trochu jinak.

  Jednou z možných automatizací týkajících se teploty je například jednoduchý termostat, jenž by udržoval v místnosti uživatelem zvolenou stálou teplotu. V případě, že by teplota klesla pod požadovanou hranici, by systém mohl provést spuštění topného tělesa, a v momentě, kdy by stoupla nad požadovanou hranici, by mohl naopak sepnout klimatizaci.

- **Vlhkost vzduchu** – Ideální vlhkost v domácnosti je podle studie kritérií vystavení člověka vlhkosti v obytných budovách z roku 1985 mezi 40-60%. Vlhkost mimo toto rozmezí může u lidí způsobovat problémy s dýchacími cestami a umožňuje snadnější a rychlejší šíření virů, bakterií a plísní [8].

  Příliš nízká vlhkost v interiéru také dokáže negativně ovlivňovat spotřebu energie potřebné k vytápění domácnosti. Příliš nízká vlhkost může způsobit, že je teplota

vzduchu uživatelem vnímána jako teplota menší, což jej nutí zbytečně domácnost vytápět na vyšší teplotu. [11].

Jednou z příležitostí pro automatizaci s touto veličinou je kupříkladu hlídání, že se vlhkost v místnosti pohybuje v popsaném ideálním rozmezí. V případě, že by klesla pod hranici tohoto rozmezí, dojde k sepnutí zařízení na zvlhčování vzduchu, a v případě, že by stoupla nad rozmezí by systém mohl sepnout ventilaci, nebo vysoušeč vzduchu.

V případě, že by byl umístěn senzor vlhkosti vzduchu ve vnějším prostředí, dalo by se jím detekovat podle úrovně vlhkosti, jestli venku prší, nebo je naopak příliš sucho. Na základě této informace by poté systém mohl zareagovat například zavřením oken v domě pro ochranu před deštěm, nebo spuštěním zavlažovačů trávníku.

- **Atmosférický tlak** – Atmosférický tlak je jedna z fyzikálních veličin, která není reálně dostupnými prostředky v domácnosti ovlivnitelná. Bohužel může přímo svými změnami způsobovat fyzicky oslabeným jedincům zdravotní potíže.

  Automatizační případ využívající tuto veličinu by ji však mohl hlídat a případně varovat uživatele při příliš nízkých, nebo vysokých hodnotách, že může docházet k vysoké zátěži pro osoby s nemocemi dýchacího ústrojí, kardiaky a seniory.

- **Intenzita osvětlení** – Stejně jako teplotu vzduchu i intenzitu světla může vnímat každý člověk odlišně. Za pomoci senzoru intenzity osvětlení by však mohlo být umožněno udržování světelnosti v místnosti na uživatelem zvolené úrovni přímým ovládáním domácího osvětlení.

- **Míra hlučnosti** – Senzor měřící hlučnost v decibelech, dokáže sloužit jako jednoduchý detektor lidské aktivity v místnosti. Koncové zařízení s tímto senzorem je tak například možné umístit do pokoje s kojencem a v případě, že by dítě začalo plakat, by mohl být uživatel upozorněn na jeho uživatelském zařízení.

  Dalším možnost pro využití této veličiny nastává, když uživatel opouští domácnost. V tomto případě míra hlučnosti slouží jako jednoduchý alarm detekující nežádoucí osoby v domácnosti. V momentě, kdy by byla detekována zvýšená zvuková aktivita, by byl uživatel varován skrze jeho uživatelské zařízení.

- **Množství CO_2 ve vzduchu** – Množství částic CO_2 ve vzduchu v domácnosti by podle standardu skupiny ASHRAE[1] číslo 62 z roku 1999 nemělo dlouhodobě přesahovat hranici 1000 ppm (počet částic na milion jiných částic). Při zvýšené koncentraci může být u lidí vyvoláno zhoršení dýchání a bolesti hlavy. Velmi vysoké koncentrace nad 50 000 ppm mohou dokonce způsobit u člověka bezvědomí.[1]

  Automatizační případ s touto veličinou by kontroloval hladinu částic CO_2 v ovzduší místnosti a při její zvýšené koncentraci varuje uživatele, případně rovnou automaticky sepne ventilátor a místnost vyvětrá.

[1] American Society of Heating, Refrigerating, and Air-Conditioning Engineers.

- **Spotřeba elektrické energie** – Senzorická data značící spotřebu elektřiny získaná ze zásuvek v domácnosti by mohla být automatizovaně analyzována. Uživateli by následně byly poskytnuty informace popisující, kdy je v domácnosti největší a nejnižší spotřeba a které zásuvky vykazují největší zátěž na elektrickou síť. Díky tomu by bylo pro uživatele zjednodušeno zajištění ideálnější spotřeby elektrické energie potřebné pro chod jeho domácnosti.

Systém by kromě jednoduchých případů automatizace měl také počítat se složitějšími případy, jež vznikají v momentě, kdy by docházelo ke zpracování více druhů fyzikálních veličin v jednom automatizačním případu. Senzory měřící tyto veličiny by také mohly být umístěny v odlišných částech domácnosti, dokonce i venku.

- **Teplota & vlhkost uvnitř** – Na základě těchto dvou veličin a zadaných izolačních vlastností okna je možné vypočítat a předpovídat rosný bod okna [10]. Při hodnotách teploty a vlhkosti, které by mohly způsobit srážení vody na okenních deskách, by pak systém mohl včasně varovat uživatele, nebo rovnou sepnout větrání a zamezit tak vzniku a šíření plísní.
- **Teplota & vlhkost uvnitř s teplotou & vlhkostí venku** – Na základě těchto teplot a vlhkostí v místnosti a venku lze vypočítat, kdy je vhodné místnost větrat, aby se vlhkost v místnosti držela v ideálním rozmezí a teplota neklesla pod uživatelem nastavenou komfortní hodnotu.

Speciálními příležitostmi pro automatizaci jsou případy, jež ke své činnosti využívají pouze čas. Ačkoliv se čas nedá považovat za formu fyzikální veličiny, byl pro úplnost do tohoto výčtu také zahrnut.

- **Čas** – Jednou z automatizací využívající pouze čas může být případ, kdy uživatel opouští na delší dobu svou domácnosti, ale chtěl by, aby se nadále okolí jevila jako obydlená. V tomto případě by systém v noci automatizovaně spínal osvětlení a elektronické spotřebiče, aby odradil zloděje.

  Správným načasování může být také usnadněna uživatelova ranní rutina. V přesně zvolenou dobu by mohly být automaticky otevřeny žaluzie, aby byl uživatel probuzen slunečními paprsky a zároveň by mohla být inicializována příprava kávy v kávovaru.

3.2 Automatizační úlohy

V předchozí sekci byly popsány různé příležitosti, jak by mohla být naměřená senzorická data automatizovaně zpracovávaná a jak by na ně systém mohl reagovat. V rámci serverové části systému *BeeeOn* však zatím neexistuje žádný prvek, jenž by takové automatizační činnosti umožnil realizovat.

Pro rozšíření funkcionality *BeeeOn* systému o automatizaci je nutné do jeho architektury přidat automatizační úlohy. Taková úloha by měla být na serveru spustitelná a konfigurovatelná přímo uživatelem z uživatelského zařízení.

3.2.1 Druhy automatizačních úloh

Každá automatizační úloha v systému musí být nějakým způsobem upozorněna na to, že má být spuštěna a začít provádět svou činnost. Proto byly na základě případů pro automatizaci domácnosti popsaných v sekci 3.1, definovány autorem této práce tři druhy automatizačních úloh, které by měly umožňovat různé způsoby spouštění:

- **Automatizační úloha spouštěná přijetím datové zprávy** – Jak již bylo popsáno v popisu brány v kapitole 2.3, brána odesílá vždy v nějakém zvoleném časovém intervalu na server naměřená senzorická data z koncových zařízení.

  V případě, že chceme, aby automatizační úloha okamžitě reagovala na nově přijatá data, je třeba, aby byla tato data předána úloze přímo při jejím spuštění. Následně by nad těmito daty provedla výpočet a adekvátně tak na ně zareagovala.

  Takovým druhem automatizační úlohy je například automatizační případ hlídající hladinu CO_2 ve vzduchu, jenž musí při zjištění nárůstu této hladiny v místnosti nad únosnou mez okamžitě začít větrat.

  Jelikož je název *Automatizační úloha spouštěná přijetím datové zprávy* velmi dlouhý, byl pro tento druh automatizační úlohy zvolen v rámci této bakalářské práce také zkrácený alternativní název využívající anglického slova *trigger* (ve významu kohoutek, spoušť). V dalším textu tohoto dokumentu se proto bude o tomto druhu úlohy referovat jako o *trigger* automatizační úloze.

- **Časovaná automatizační úloha** – Některé automatizační úlohy ke své činnosti nepotřebují reagovat okamžitě na přijatá senzorická data, ale spíše vyžadují, aby byly spouštěny v nějakou předem určenou dobu.

  Pro tyto případy je vhodné do systému přidat druh úlohy, která má možnost naplánovat své spuštění na určitý čas, nebo s nějakou časovou periodou.

  Tento druh úlohy se kromě automatizačních případů vyžadujících pro svou činnost pouze čas, jako je třeba případ pro odrazení zlodějů při nepřítomnosti uživatele, také hodí pro různé periodicky spouštěné statistické a kontrolní výpočty nad daty uloženými v databázi na serveru.

- **Kombinovaná úloha** – Některé automatizační případy mohou mít takové požadavky a složitost, že již nebudou pro jejich realizaci stačit předchozí dva typy automatizačních úloh. V případě, že by automatizační úloha potřebovala jak okamžitě reagovat na přijatá senzorická data, tak být spouštěna s nějakým časovým intervalem, je potřeba přidat i druh, který kombinuje časovanou a *trigger* automatizační úlohu dohromady.

  Tento druh úlohy může být vhodný pro realizaci automatizačních případů, jež pro svoji činnost potřebují mít možnost se učit nad daty získanými za nějaký časový úsek. Úloha by tak měla mít možnost s určitou časovou periodou vylepšovat svou bázi znalostí. S příjmem senzorických dat by pak na ně bylo adekvátně zareagováno i s přihlédnutím k naučeným znalostem.

3.3 Návrh konkrétních automatizačních úloh

Tato sekce se zaměřuje na návrh několika konkrétních automatizačních úloh na základě automatizačních případů popsaných ve výčtu v sekci číslo 3.1.

Jejich návrh je zaměřen hlavně na činnost, kterou budou automatizační úlohy realizovat, jakého druhu musejí automatizační úlohy být a jakým způsobem si bude uživatel úlohu moci nakonfigurovat.

3.3.1 Obecný hlídač

Účelem této automatizační úlohy je porovnávat podle uživatelem zvoleného operátoru senzorická data z jednoho senzoru s uživatelem zvolenou hodnotou. V případě splnění podmínky zvoleného operátoru pak úloha zareaguje uživatelem vybraným způsobem.

Tato automatizační úloha by měla být schopna přijímat a porovnávat senzorická data z libovolného senzoru, který uživatel vlastní. Tudíž je její použití dosti univerzální a vhodnou konfigurací by jí mohlo být realizováno velké množství automatizačních případů.

K realizaci obecného hlídače je nejvhodnější druh automatizační úlohy *trigger*, protože ke své činnosti potřebuje získávat co nejaktuálnější senzorická data, okamžitě je zpracovávat a podle zvoleného způsobu na ně reagovat.

K jeho činnosti je nutných několik základních konfiguračních údajů:

- Identifikace senzoru, jehož hodnota má být porovnávána.
- Operátor porovnávání.
- Hodnota, se kterou mají být nově přijatá data ze senzoru porovnávána.

Jelikož většina senzorů svá data ukládá a dále je do systému *BeeeOn* distribuuje jako čísla s plovoucí řádovou čárkou, jsou dostačujícími zvolitelnými operátory porovnávání pouze: větší než a menší než. Použití operátorů na rovnost je v tomto případě nepraktické, protože by ke splnění jejich podmínky nikdy nemuselo dojít.

Způsobů jak může tato automatizační úloha reagovat na splnění podmínky uživatelem zvoleného operátoru je několik:

- Hlídač zašle při splnění podmínky uživatelem zvolenou notifikační zprávu na jeho uživatelské zařízení.
- Hlídač změní stav uživatelem zvoleného aktuátoru. Identifikace aktuátoru a hodnota, která má být na něm nastavena, by měly být tudíž také součástí konfigurace.
- Hlídač provede obě předchozí činnosti.

S jakou konfigurací a za jakým účelem tuto úlohu uživatel spustí, by již mělo být pouze na něm. Použitím více hlídačů se dá přímo realizovat větší množství automatizačních případů popsaných v sekci 3.1.

Kombinací dvou hlídačů lze například realizovat jednoduchý termostat, kdy první hlídač bude nastaven tak, aby při překročení určité teploty vypnul topné těleso, a druhý zase při klesnutí teploty pod určitou hranici těleso zapnul.

3.3.2 Kontrola živosti brány a koncových zařízení

Návrh této automatizační úlohy je specifický pro potřeby systému *BeeeOn*. Neexistuje v něm totiž momentálně způsob jak explicitně oznámit uživateli, že jeho koncová zařízení nebo brána, jsou v systému nedostupná.

Když se tedy například stane, že koncovému zařízení dojde energie v bateriích, nebo je brána odpojena od *Internetu*, může na tuto skutečnost uživatel přijít až po dlouhé době, kdy zjistí, že mu do systému nebyla ukládána žádná senzorická data.

Proto je v systému *BeeeOn* potřeba prvek, jenž by tuto kontrolu prováděl. Vhodným způsobem jak tento problém řešit je časovaná automatizační úloha. Tato úloha tak s určitou časovou periodou ověřuje, jestli jsou brána a její koncová zařízení dostupná.

Kontrola živosti bude vždy prováděna v rámci jedné brány a všech jejích připojených koncových zařízení. K spuštění této automatizační úlohy bude docházet v momentě, kdy je poprvé brána přidána do systému *BeeeOn*, a ukončena bude pouze až s odpojením brány ze systému.

K zjištění, že je brána nedostupná, by mělo v budoucnu sloužit právě vyvíjené rozšíření komunikace mezi serverem a bránou, jež umožní ze serveru pomocí specifické zprávy zjistit, jestli je brána dostupná v síti *Internet*.

K zjištění, jestli je nedostupné koncové zařízení, neexistuje v *BeeeOn* systému zatím žádná přímá funkcionalita. Na základě uživatelských zkušeností tak bylo jako nejvhodnější řešení tohoto problému zvoleno, že každé koncové zařízení, které ve třech intervalech kdy mělo zaslat senzorická data, tato data nezaslalo, bude označeno jako nedostupné.

V momentě, kdy bude zjištěno, že některé zařízení není v systému dostupné, bude v uživatelském rozhraní u takového zařízení zobrazena informace o jeho nedostupnosti. Pokud si uživatel v konfiguraci této automatizační úlohy navíc zvolí možnost zasílání notifikací, bude mu na jeho uživatelské zařízení také zaslána zpráva upozorňující ho na nedostupnost zařízení.

Kapitola 4

Návrh rozšiřitelného systému

V předchozí kapitole bylo vysvětleno, co je to v rámci této práce myšleno automatizační úlohou a byly také popsány některé konkrétní případy. Vhodným umístěním pro činnost těchto automatizačních úloh je server systému *BeeeOn*, jelikož jsou na něj směrována a také na něm ukládána všechna senzorická data z koncových zařízení.

Nebylo by však příliš vhodné a ani dobře udržovatelné, aby každá automatizační úloha byla realizována a spouštěna jako samostatná služba. Vhodným způsobem řešení je tudíž realizace rozšiřitelného systému, který bude sloužit ke správě a řízení více automatizačních úloh najednou.

Tato kapitola se věnuje návrhu právě tohoto systému, pro nějž byl v rámci *BeeeOn* architektury zvolen zkrácený název BAF (*BeeeOn* Automation Framework).

4.1 Požadavky na rozšiřitelný systém

Hlavním požadavkem pro systém je oddělení implementace rozšiřitelného systému samotného a automatizačních úloh, a tak umožnění dalšího jednoduchého rozšiřování systému o nové automatizační úlohy. Jedním z hlavních důvodů je fakt, že se budou dalším vývojem jednotlivých automatizačních úloh zabývat — kromě autora této práce — také jiní vývojáři.

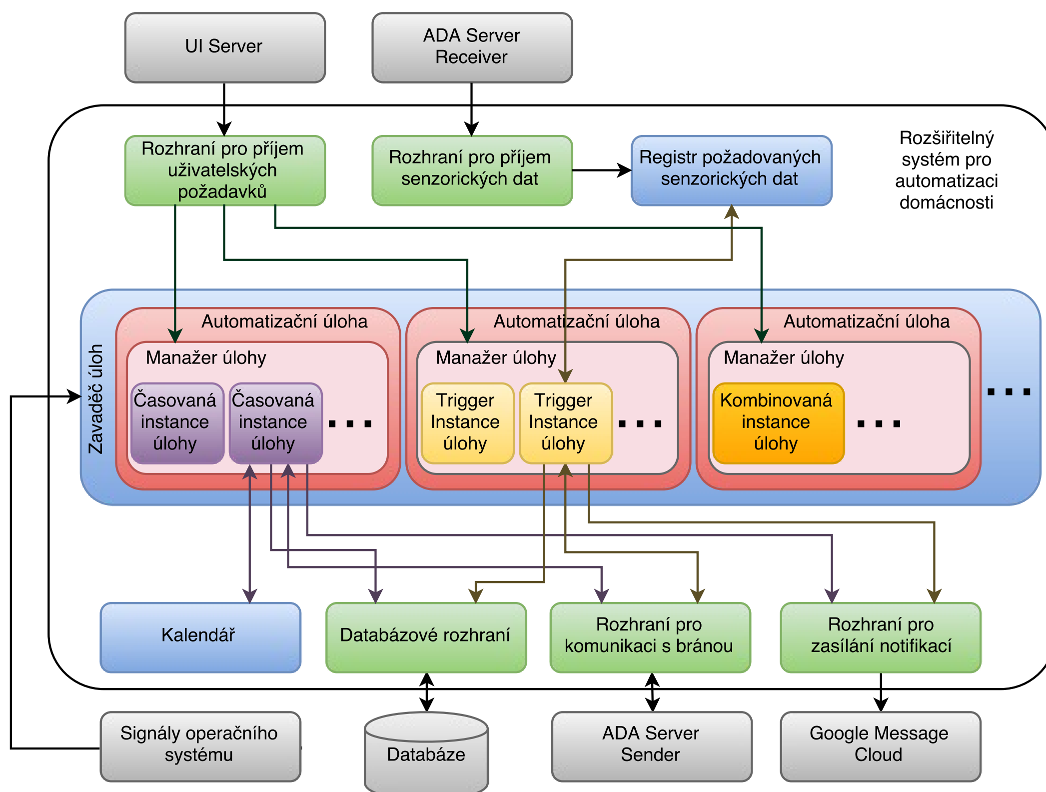
Rozšiřitelný systém tudíž musí poskytnout programátorovi automatizační úlohy nástroje a komponenty potřebné k implementaci a řízení automatizačních úloh, komunikační rozhraní pro příjem uživatelských požadavků a senzorických dat, komunikační rozhraní sloužící k ovlivňování ostatních částí *BeeeOn* systému (přepínání stavů aktuátorů, zasílání notifikací na uživatelské zařízení atd.) a způsob jak vytvořenou automatizační úlohu načíst do rozšiřitelného systému.

Systém by měl být navržen s přihlédnutím ke snadné udržovatelnosti, protože se počítá s jeho nasazením do aktivního provozu a s dalším případným rozšiřováním jeho funkcionality. Z toho vyplývá, že musí být kladen důraz na přehlednou architekturu a objektový návrh.

4.2 Vnitřní struktura systému

Komponenty navrhovaného rozšiřitelného systému je možné rozdělit do tří skupin:

- Reprezentace automatizační úlohy.
- Řídící komponenty.
- Komunikační rozhraní.



Obrázek 4.1: Návrh schématu rozšiřitelného systému

Ve schématu navrhovaného rozšiřitelného systému v obrázku s číslem 4.1 jsou zelenou barvou zvýrazněna komunikační rozhraní. Modrou barvou jsou vyobrazeny řídící komponenty. Odstínem barvy červené, fialovou, žlutou a oranžovou jsou označeny komponenty reprezentující automatizační úlohy. Externí služby, s nimiž rozšiřitelný systém komunikuje, mají barvu šedou.

Vztahy vyobrazené ve schématu 4.1 šipkami reprezentují které komponenty, případně služby serveru a v jakém směru mezi sebou komunikují. Vztahy spojené s komponentou kombinované instance úlohy byly však záměrně ze schématu pro celkovou větší přehlednost vynechány. Jedná se však pouze o sjednocení všech vztahů časované a *trigger* instance úlohy.

4.2.1 Reprezentace automatizační úlohy

Každá automatizační úloha je v rozšiřitelném systému zastoupena jednou, samostatně přeloženou, dynamickou knihovnou. Modularita a snadná rozšiřitelnost navrhovaného systému

o další automatizační úlohy bude poté umožněna tím, že všechny automatizační úlohy budou ve formě těchto dynamických knihoven do rozšiřitelného systému načítány až při jeho spuštění, nebo případně přímo za jeho běhu.

Z toho vyplývá, že v případě potřeby načíst novou automatizační úlohu do rozšiřitelného systému nebude nutné rozšiřitelný systém restartovat, nebo dokonce znova celý překládat.

Každá dynamická knihovna reprezentující automatizační úlohu musí obsahovat následující komponenty, které vývojářům poskytnou možnosti k realizaci konkrétních automatizačních úloh:

Instance úlohy

Tato komponenta reprezentuje v rozšiřitelném systému jedno konkrétní spuštění jedné automatizační úlohy. To znamená, že když si uživatel ve svém uživatelském rozhraní nějakou automatizační úlohu spustí, bude mu v rozšiřitelném systému vytvořena jedna instance této úlohy. Díky tomu je možné, aby měl uživatel spuštěno více instancí jedné automatizační úlohy najednou.

Druhy instancí úlohy odpovídají druhům automatizačních úloh tak, jak byly popsány v sekci *Druhy automatizačních úloh* 3.2.1. Tudíž jsou v systému také tři druhy instancí úloh: časovaná, trigger a kombinovaná instance. Výběrem jednoho druhu této komponenty pro realizaci konkrétní automatizační úlohy je pak přímo specifikován její druh.

Manažer úlohy

Každá automatizační úloha musí v rozšiřitelném systému obsahovat právě jednoho manažera úlohy. Tato komponenta slouží hlavně jako správce instancí úlohy dané automatizační úlohy. Jeho hlavní součástí je kolekce obsahující všechny vytvořené a spuštěné instance úlohy dané automatizační úlohy.

Ke spravování instancí úlohy tato komponenta také obsahuje rozhraní, jež musí obsahovat funkcionalitu sloužící k vytváření nových instancí úlohy, změně konfigurace již existujících instancí úlohy, mazání existujících instancí úlohy a k získávání informací o existujících instancích úlohy.

4.2.2 Řídící komponenty

Řídící komponenty v rozšiřitelném systému slouží k načítání automatizačních úloh do rozšiřitelného systému a k zajištění správného spuštění jednotlivých instancí úloh. Jedním z důležitých předpokladů je, že každá řídící komponenta bude v systému vždy spuštěna pouze jednou.

Zavaděč úloh

Každou automatizační úlohu ve formě dynamické knihovny je za běhu rozšiřitelného systému potřeba načíst a také v systému udržovat. K řízení této činnosti slouží komponenta *Zavaděč úloh*.

Informace o automatizačních úlohách jsou obsaženy ve formě konfiguračního souboru. Tento soubor musí vždy obsahovat o každé automatizační úloze následující informace: jednoznačný identifikátor v rámci všech automatizačních úloh, její verzi, název, druh a hlavně cestu v rámci souborového systému k dynamické knihovně, v níž je automatizační úloha implementována. Tento soubor je při spuštění rozšiřitelného systému touto komponentou

zpracován a všechny v něm obsažené automatizační úlohy jsou do rozšiřitelného systému načteny.

V případě že systém obdrží během svého běhu signál `SIGUSR1` od operačního systému vyvolaného správcem serveru, je tato komponenta upozorněna a provede nové zpracování konfiguračního souboru. Poté načte úlohy, jejichž identifikátory jsou v rámci rozšiřitelného systému nové. Díky této funkcionalitě nebude nutné pro přidání nové automatizační úlohy rozšiřitelný systém vypínat nebo restartovat.

Kalendář

Tato řídící komponenta slouží k řízení spouštění časovaných a případně také kombinovaných, instancí úloh. Samotná instance úlohy, jež má možnost pracovat s časem, může libovolně plánovat do kalendáře, v jaké časy má být přesně spuštěna, případně svá naplánování mazat.

Na základě naplánovaných spuštění poté v reálném čase algoritmus kalendáře, paralelně spuštěný ke všem ostatním procesům v rozšiřitelném systému, postupně spouští naplánované instance úloh.

Registr požadovaných senzorických dat

Tato komponenta je určena k zajištění spouštění *trigger* a kombinovaných instancí úloh, jež mají být spouštěny s přijetím senzorických dat.

Její součástí je kolekce obsahující záznamy o tom, ze kterých koncových zařízení chtějí jednotlivé instance úloh získávat senzorická data. Ty mohou libovolně příslušné instance do této komponenty vkládat nebo je z ní také mazat. Na základě těchto záznamů jsou poté po přijetí příslušné zprávy obsahující požadovaná data instance úloh spouštěny.

4.2.3 Komunikační rozhraní

Pro zajištění toho, aby mezi sebou mohly komunikovat rozšiřitelný systém s automatizačními úlohami a ostatní části *BeeeOn* systému, je potřeba navrhnout a realizovat několik komunikačních rozhraní.

Rozhraní pro příjem senzorických dat

Tomuto rozhraní jsou přeposílána senzorická data ve formě datových zpráv, jež obdržela od bran služba *ADA Server Receiver*. Tímto rozhraním je poté tato zpráva zpracována a k dalšímu zpracování předána *Registru požadovaných senzorických dat*. Tím je vyvoláno spuštění instancí úloh, které přijatá senzorická data ke své činnosti vyžadují.

Rozhraní pro příjem uživatelských požadavků

Toto rozhraní slouží k přijímání požadavků od uživatele určených k ovládání rozšiřitelného systému. Tyto požadavky jsou přijímány ve formě zpráv, jež tomuto rozhraní přeposílá serverová služba sloužící ke komunikaci s uživatelskými zařízeními, která je nastavena tak, aby detekovala a přeposílala pouze zprávy, jež jsou určeny rozšiřitelnému systému.

U zpracování každé zprávy, která nějak mění již existující instanci úlohy, musí být zajištěné, že uživatel opravdu danou instanci úlohy vlastní, aby tak nemohlo dojít ke zneužití funkcionality rozšiřitelného systému.

Pro ovládání rozšiřitelného systému ze strany uživatele je navrženo šest následujících požadavků:

- **Vytvoření nové instance úlohy** – Rozšiřitelný systém po přijetí tohoto požadavku vytvoří a spustí novou instanci úlohy. Je nutné, aby v této zprávě byl obsažen identifikátor spouštěné automatizační úlohy a celá konfigurace potřebná k běhu instance úlohy. Odpovědí po správném zpracování tohoto požadavku je identifikátor nově vytvořené instance úlohy.
- **Změněna konfigurace existující instance úlohy** – Tento požadavek musí obsahovat hlavně jednoznačný identifikátor instance úlohy, jejíž konfigurace má být změněna, a konfigurační údaje, jež mají být měněny. Tyto údaje nemusí být na rozdíl od požadavku pro vytvoření obsaženy všechny.
- **Smazání existující instance** – Po přijetí tohoto požadavku rozšiřitelný systém provede smazání instance úlohy, jejíž identifikátor se v požadavku nachází.
- **Získání všech identifikátorů instancí konkrétní úlohy** – Tento požadavek musí obsahovat identifikátor automatizační úlohy, jejíž instance se mají v systému vyhledat. Uživateli je poté zaslána odpověď obsahující identifikátory všech instancí dané úlohy, které uživatel vlastní.
- **Získání konfigurace instance úlohy** – Tento požadavek musí obsahovat identifikátor instance úlohy, jejíž konfiguraci chce uživatel získat. Uživateli je následně odpovězeno kompletní konfigurací dané instance úlohy. Hlavním využití této zprávy je k umožnění zobrazení získané konfigurace v uživatelském rozhraní.
- **Získání dat z instance úlohy** – Tento požadavek je navržen pro získání jakýchkoliv dat z instance úlohy, jež mohou být zobrazeny v uživatelském rozhraní (například grafy hodnot počítaných instancí úlohy). Musí obsahovat parametry, které musí manažer konkrétní automatizační úlohy umět identifikovat, korektně zpracovat a na jejichž základě musí adekvátně přímo vytvořit odpověď.

Databázové rozhraní

Toto rozhraní slouží k zajištění komunikace rozšiřitelného systému a všech jeho komponent se systémem řízení báze dat, ve kterém jsou uchovávány všechny informace o činnosti celého systému *BeeeOn*.

Konkrétní implementace tohoto rozhraní musí být dimenzována tak, aby bylo možné najednou obsloužit i větší množství dotazů na databázový systém. Také toto rozhraní nesmí nijak omezovat vývojáře automatizačních úloh a mělo by jim umožnit používání libovolných dotazů v jazyce SQL.

Rozhraní pro komunikaci s bránou

Toto rozhraní umožňuje instancím úloh komunikovat s bránami a k nim připojenými koncovými zařízeními. Prostředníkem pro zasílání příkazů bránám je pro toto rozhraní část *Sender* serverové služby *ADA Server*. Ta již nyní slouží k přijímání příkazů od uživatelů, jež jí jsou přeposílány pomocí interního protokolu od služby *UI Server*.

Tento protokol má definovaných více druhů příkazů, ale pro účely tohoto rozhraní jsou nejdůležitější příkazy `SWITCH` a `PING`. První umožňuje zadání pokynu ke změně stavu aktuátoru na libovolné bráně, druhý slouží k zjištění dostupnosti brány v síti *Internet*. Ukázka těchto zpráv se nachází v příloze číslo C.2.

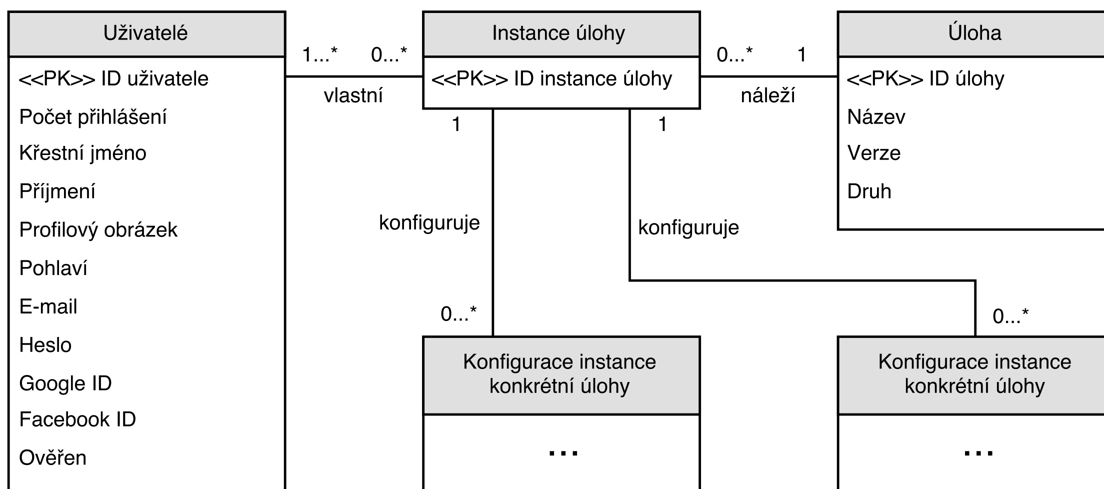
Rozhraní pro zasílání notifikací

Kromě změn stavů aktuátorů na koncových zařízení by mělo být automatizačním úlohám umožněno případné výsledky své činnosti zasílat uživateli sítí *Internet* na jeho uživatelské zařízení ve formě notifikačních zpráv.

Jelikož jediným uživatelským rozhraním, jež je momentálně schopné přijímat a zobrazovat notifikační zprávy, je aplikace určená pro zařízení s operačním systémem *Google Android*, bude toto rozhraní sloužit převážně k zasílání notifikací právě na tato zařízení pomocí externí služby *Google Cloud Messaging*. Tato služba také zajišťuje, že bude notifikace uživateli zobrazena i v případě, že v době zaslání nebylo jeho zařízení připojené k síti *Internet*.

4.3 Rozšíření databáze

Pro potřeby rozšiřitelného systému je také potřeba rozšířit i databázové schéma na serveru o několik tabulek. Tyto tabulky by měly sloužit k uchování dat popisujících automatizační úlohy a jejich vytvořené instance úloh.



Obrázek 4.2: Diagram entitních vztahů navrhovaných tabulek rozšiřitelného systému

V diagramu entitních vztahů 4.2 jsou tyto tabulky a vztahy mezi nimi znázorněny. Součástí tohoto diagramu je také již existující databázová tabulka s údaji o uživatelích *BeeeOn* systému.

V návrhu se také počítá s tím, že konfigurace každé konkrétní instance úlohy bude ukládána v databázi do samostatné tabulky. Tím bude možná kontrola konzistence ukládaných dat pomocí cizích klíčů. Když bude například součástí konfigurace automatizační úlohy identifikátor nějakého koncového zařízení, bude již systémem řízení báze dat kontrolována existence tohoto identifikátoru v databázi.

Kapitola 5

Implementace rozšiřitelného systému

Tato kapitola se podrobněji věnuje tomu, jak byl implementován rozšiřitelný systém pro automatizaci domácností včetně jeho jednotlivých komponent.

Jelikož jsou všechny *BeeeOn* serverové služby napsány v jazyce C++, byl tento jazyk zvolen i pro implementaci tohoto systému. Tento programovací jazyk je kompilovaný, tudíž by měl být v něm implementovaný systém dostatečně výkonný, aby mohl plynule běžet na cílovém serverovém počítači. Součástí standardní knihovny zvolené verze tohoto jazyka z roku 2011 je také velké množství užitečných tříd a funkcí, jež usnadňují a zpřehledňují implementaci.

Cílovým operačním systémem pro nasazení je GNU/Linux, který je primárním operačním systémem na *BeeeOn* serveru, na němž má po integraci rozšiřitelný systém provozovat svou činnost.

Z návrhu rozšiřitelného systému vyplývá také skutečnost, že by měl být schopen v jeden moment zpracovávat větší množství požadavků, ať již od jednotlivých automatizačních úloh, nebo přímo od uživatelů. Z toho důvodu byl systém implementován tak, aby dokázal různé požadavky obsluhovat ve více vláknech, čímž je urychleno jejich zpracování.

Cesta ke konfiguračnímu souboru, jež obsahuje údaje potřebné pro fungování většiny komponent rozšiřitelného systému, je jediným požadovaným argumentem, který je potřeba předat rozšiřitelnému systému při jeho spouštění. Ukázka tohoto souboru je v příloze D.1. V něm byly záměrně z bezpečnostních důvodů vynechány údaje sloužící k přihlášení do databázovému systému umístěném na *BeeeOn* serveru.

5.1 Použité technologie

Kromě standardní knihovny jazyka C++ je zapotřebí k zajištění správného fungování některých komponent rozšiřitelného systému také specifická funkcionalita, kterou poskytují následující vybrané knihovny pro jazyk C a C++.

Knihovna dlfcn.h

Tato standardní *POSIX*[1] knihovna umožňuje jednoduché a efektivní načítání dynamických knihoven za běhu programu a následné načtení symbolů jejich funkcí, jež je poté možné

[1] Portable Operating System Interface

spouštět jako běžné funkce programu.

Jelikož je tato knihovna původně navržena pro použití v jazyce C, je potřeba pro její správné fungování v jazyce C++ následovat určité postupy. Názorným a často referovaným návodem jak tuto knihovnu používat v jazyce C++ je internetová stránka *C++ dlopen mini HOWTO* napsaná Aaronem Isottonem [4].

Knihovna Asio

Asio je multiplatformní C++ knihovna určená pro realizaci síťových systémů a systémů využívajících nízkoúrovňové vstup–výstupní operace. Nabízí vývojářům konzistentní asynchronní model využívající moderní přístup C++ k implementaci serverových a klientských architektur.

Tato knihovna je od konce roku 2005 součástí sady C++ knihoven *Boost*. Pro účely této práce je však využita její osamostatněná varianta, která nabízí totožnou funkcionalitu bez nutnosti do rozšiřitelného systému přidávat závislosti na knihovny sady *Boost*.

Knihovna soci

Ke komunikaci a výměně dat se systémem pro řízení báze dat *PostgreSQL* byla zvolena C++ open-source knihovna soci. Ta si klade za cíl nabízet rychlé a intuitivní rozhraní pro používání příkazů jazyka SQL přímo v kódu jazyka C++. Při jejím správném používání tato knihovna také zajišťuje ochranu proti útoků na databázový systém skrze uživatelem zasílané požadavky, jako jsou například útoky typu *SQL injection*.

Jedním z hlavních rozhodujících faktorů ke zvolení této knihovny bylo její aktivní používání v rámci všech ostatních služeb serverové architektury systému *BeeeOn*. Je tak již dlouhodobě ověřeno, že je vhodná pro účely serverových služeb tohoto projektu.

Knihovna RapidJSON

RapidJSON je malá a rychlá C++ knihovna pro zpracování konstrukcí jazyka JSON. Její výkon je srovnatelný s funkcí standardní knihovny jazyka C `strlen()`. Pro akceleraci své činnosti také podporuje instrukční sady procesorů Intel SSE2 a SSE4.2, jež umožňují zpracování více dat v jedné instrukci. Instrukční sada SSE2 je také přímo dostupná na procesoru cílového *BeeeOn* serveru.

Tato knihovna je soběstačná a skládá se pouze z hlavičkových souborů. Neobsahuje závislosti na žádných jiných knihovnách, a to včetně standardních knihoven jazyků C a C++.

Knihovna PugiXML

PugiXML je rychlá[1] a jednoduchá open-source knihovna pro jazyk C++, sloužící ke zpracování konstrukcí jazyka XML. I přes svoji jednoduchost nabízí plnohodnotné rozhraní pro zpracování a tvorbu konstrukcí v jazyce XML.

Jedním z hlavních důvodů, proč byla pro účely této práce tato knihovna zvolena, je to, že je také již ověřena a využívána všemi ostatními službami *BeeeOn* serveru.

[1] PugiXML Benchmark - http://pugixml.org/benchmark.html

Knihovna unified_logger

K formátování a ukládání provozních záznamů (logů) rozšiřitelného systému byla použita knihovna *unified_logger*, vyvíjená jako bakalářská práce členem *BeeeOn* týmu Markem Beňom [2].

Tato knihovna umožňuje tvorbu provozních záznamů obsahujících kromě samotné zprávy popisující záznam také čas vytvoření záznamu s přesností na milisekundy, identifikátor určující hierarchii záznamů, číslo vlákna, ve kterém byl záznam vytvořen, název souboru a číslo řádku, na kterém se nachází funkce vytvářející záznam, a jednu ze sedmi úrovní určujících druh provozních dat.

Očekává se, že tato knihovna bude postupně také integrována do všech ostatních služeb *BeeeOn* serveru, aby tak došlo k finálnímu sjednocení formátu všech provozních záznamů.

Knihovna Notificator

Pro umožnění automatizačním úlohám zasílat uživatelům na jejich uživatelská zařízení notifikační zprávy byla použita knihovna *Notificator*. Ta byla v rámci projektu *BeeeOn* navržena a implementována členem Martinem Douděrou.

Ačkoliv je tato knihovna navržena tak, aby mohla v budoucnu zasílat notifikační zprávy na více druhů uživatelských zařízení, aktuálně dokáže zasílat zprávy pouze ve formě textu na zařízení s operačním systémem *Google Android* a to skrze službu *Google Message Cloud*.

Tato knihovna v rozšiřitelném systému slouží jako plnohodnotné rozhraní pro zasílání notifikačních zpráv a je jí tak rovnou implementováno *Rozhraní pro zasílání notifikací*. Je tudíž určena pro přímé užívání vývojáři automatizačních úloh.

5.2 Reprezentace automatizačních úloh

Tato sekce se zaměřuje na to, jakým způsobem jsou v rozšiřitelném systému implementovány třídy, jež slouží k realizaci automatizačních úloh.

Po vývojáři konkrétní automatizační úlohy je vyžadováno, aby implementoval dvě třídy potřebné pro chod úlohy v rozšiřitelném systému. První třída reprezentuje komponentu instance úlohy a druhá manažera úlohy. K umožnění implementace těchto komponent bylo vytvořeno několik podpůrných tříd, od kterých vývojář přímo může dědit a využívat jimi implementovanou funkcionalitu.

5.2.1 Instance úlohy

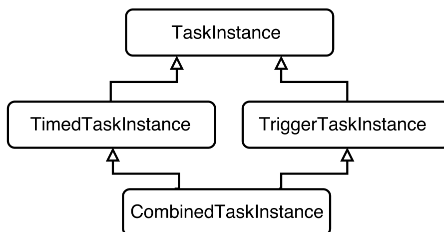
Pro správné fungování rozšiřitelného systému bylo sjednoceno rozhraní všech druhů instancí úloh třídou `TaskInstance`, jež slouží jako virtuální rozhraní, od kterého poté dědí třídy zastupující jednotlivé druhy instancí úloh.

Konstruktoru této třídy je nutné předat dva údaje. Prvním je identifikátor instance úlohy, jenž byl databází vygenerován ještě před konstrukcí objektu této třídy. Druhým je reference na manažera, který spravuje danou instanci úlohy. Hlavním důvodem pro předání reference na spravujícího manažera je umožnění fungování členské funkce `deleteItself()`, jíž je instanci úlohy umožněno smazat sebe samu z rozšiřitelného systému.

Hlavní funkcí této třídy je funkce `activate()`. Ta slouží jako vstupní bod spouštějící instance úlohy jejich řídící komponentou. Je přetížena tak, aby z řídící komponenty pro potřeby časovaných instancí bylo možné funkci předat čas spuštění instance a pro potřeby trigger instancí bylo možné předat senzorická data.

Ve funkci `activate()` je při spuštění instance zavolána čistě virtuální členská funkce `run()`. Tato funkce musí být implementována přímo instancí konkrétní úlohy a je v ní proveden výpočet specifický pro konkrétní úlohu. Hlavním důvodem, proč není řídicí komponentou rovnou spouštěna funkce `run()` je to, že ve funkci `activate()` dochází k uzamčení binárního semaforu, jenž zajistí, že nebude jedna instance úlohy najednou spuštěna z více vláken.

Tato třída také obsahuje virtuální funkci `deleteFromControlComponent()`, při jejímž zavolání je provedeno smazání všech záznamů sloužících ke spouštění dané instance úlohy z její řídící komponenty. Tato funkce je tudíž specifická pro každý druh instance, proto je implementována až přímo třídami implementujícími jednotlivé druhy instancí úloh.



Obrázek 5.1: Diagram dědičnosti tříd druhů instancí úloh

V diagramu s číslem 5.1 je vyobrazena hierarchie dědičnosti všech tříd druhů instancí úloh. Tyto třídy a jejich specifické funkce sloužící k plánování spouštění, jsou popsány v následujícím výčtu:

- **Časovaná instance úlohy** – Tento druh instance úlohy je implementován třídou `TimedTaskInstance`. V případě implementace konkrétní časované automatizační úlohy je potřeba, aby vývojář vytvořil třídu, která od této třídy přímo dědí.

  Tato třída nabízí několik funkcí, jež může vývojář použít k naplánování spouštění časované instance úlohy. První z nich je funkce `planActivationNow()`, která slouží k naplánování instance úlohy do *Kalendáře* na aktuální čas, což způsobí její okamžité spuštění. Další plánovací funkcí je `planActivationAfterSeconds()`, jejíž jediným parametrem je počet vteřin, po jejichž uplynutí bude provedeno spuštění dané instance úlohy. Poslední funkcí k naplánování spuštění instance je funkce `planActivationToDateAndTime()`, jíž jsou jedním řetězcem předána čísla oddělená mezerou značící den, měsíc, rok a čas ve formátu `HH:MM:SS`, kdy přesně má být instance úlohy spuštěna.

  Tato třída obsahuje také kolekci aktivačních časů, do které jsou při každém naplánování spuštění automaticky vloženy záznamy, kdy má být instance úlohy spuštěna. Při spuštění je z této kolekce příslušný záznam smazán. Tato kolekce je určena hlavně pro usnadnění mazání již existujících naplánování spuštění instance z *Kalendáře*.

  Ke smazání již naplánovaného spuštění instance úlohy slouží funkce `removePlannedActivation()`. Tato funkce vyžaduje, aby jí byl předán aktivační čas, jenž má být z *Kalendáře* odstraněn.

- **Trigger instance úlohy** – K realizaci tohoto druhu instance úlohy byla implementována třída `TriggerTaskInstance`. Podobně jako v předchozím případě musí vývojář, který má v úmyslu implementovat automatizační úlohu druhu *trigger*, vytvořit konkrétní instanci úlohy, jež od této třídy dědí.

V této třídě je určena k plánování spouštění pouze funkce s názvem `registerToReceiveDataFromDevice()`. Ta slouží pro uložení záznamu do komponenty *Registr požadovaných senzorických dat*, který danou instanci následně začne spouštět vždy s příjmem senzorických dat z vybraného koncového zařízení. Této funkci je tudíž potřeba předat jednoznačný identifikátor koncového zařízení, z něhož má instance senzorická data přijímat.

Součástí této třídy je také kolekce, do které jsou automaticky ukládány identifikátory všech koncových zařízení, ze kterých zařízení chce daná instance přijímat senzorická data. Díky tomu je zjednodušena správa těchto záznamů a jejich mazání z *Registru požadovaných senzorických dat.*

K mazání záznamů z řídící komponenty je určena funkce `removeEntryFromDataMessageRegiser()`. Tato funkce pouze vyžaduje, aby jí byl předán identifikátor koncového zařízení jehož senzorická data má *trigger* instance přestat požadovat.

- **Kombinovaná instance úlohy** – Jak již bylo v sekci *Druhy automatizačních úloh* 3.2.1 popsáno, kombinované automatizační úlohy jsou spojením časovaných a *trigger* úloh. Z tohoto důvodu by měla třída `CombinedTaskInstance` implementující tento druh instance úlohy mít analogicky k dispozici všechny funkce k plánování svého spuštění, jako mají třídy `TimedTaskInstance` a `TriggerTaskInstance`.

  Tato třída obsahuje dva vstupní body pro spuštění z každé řídící komponenty. Vývojář tohoto druhu instance úlohy tudíž musí implementovat dvě funkce `run()`. Jednu, která je spuštěna s přijetím požadovaných senzorických dat, a druhá, jež je spouštěná v naplánovaných časech.

  Jak je viditelné v diagramu 5.1, třída `CombinedTaskInstance` dědí zároveň od třídy `TimedTaskInstance` i třídy `TriggerTaskInstance`, jež zase obě dědí od třídy `TaskInstance`. Tímto druhem vícenásobné dědičnosti dochází k takzvanému diamantovému problému. Ten se vyznačuje tím, že bez příslušných opatření by byly objektu třídy `CombinedTaskInstance` zkonstruovány dva základní objekty třídy `TaskInstance`. Překladač by v tomto případě nevěděl, ze kterého z těchto dvou objektů má spouštět funkce a vznikla by tak kolize, jež by vyústila v chybu při překladu. Řešením tohoto problému je v jazyce C++ použití virtuální dědičnosti [7], díky které je třídě `CombinedTaskInstance` explicitně zkonstruován pouze jeden objekt základní třídy `TaskInstance`.

5.2.2 Manažer úlohy

Základní třídou implementující tuto komponentu je třída `BaseTaskManager`. Ta slouží jako čistě virtuálního rozhraní, jehož funkce implementují až třídy, které od něj dědí. Valná většina těchto funkcí sloužící ke správě instancí úloh a je spouštěna *Rozhraním pro příjem uživatelských požadavků.*

Tato třída obsahuje asociativní kolekci, v němž jsou uchovávány unikátní identifikátory a reference na objekty instancí úlohy, které byly daným *manažerem* vytvořeny. Součástí této třídy je také binární semafor, jenž slouží k ochraně popsané kolekce před souběžným přístupem z více vláken.

Od třídy `BaseTaskManager` dědí třída `TaskManager`. Ta přímo implementuje některé její virtuální funkce. Jednou z nich je `createInstance()`, jež je určena k vytvoření záznamu o instanci úlohy v databázi a následně k získání databází vygenerovaného identifikátoru

této instance. Funkce `deleteInstance()` slouží naopak k smazání záznamu o instanci úlohy z databáze a k smazání jejího objektu z *manažera*. Funkcí `getInstanceIds()` jsou na základě identifikátoru uživatele, který je této funkci předán, navráceny z databáze získané identifikátory všech instancí dané úlohy, jež uživatel vlastní.

Jak již bylo zmíněno výše, od vývojáře konkrétní automatizační úlohy je vyžadováno, aby pro její chod implementoval jednu třídu konkrétního manažera úlohy dědící od třídy `TaskManager`. V této třídě musí vývojář implementovat několik funkcí, které byly deklarovány ve třídě `BaseTaskManager`.

První z těchto funkcí `createConfiguration()` je určena k vytvoření objektu konkrétní instance úlohy přímo v manažeru dané úlohy a k uložení její konfigurace do databáze. Důvodem, proč není uložení konfigurace součástí výše popsané funkce `createInstance()`, je to, že tabulka databáze uchovávající konfigurace je známa pouze konkrétnímu manažeru úlohy a není proto možné ukládání konfigurace generalizovat. Funkce `changeConfiguration()` je zavolána v momentě, kdy je uživatelem vyžadována změna konfigurace již existující instance úlohy, a slouží k uložení nová konfigurace do databáze.

K získání konfigurace konkrétní instance úlohy z databáze, aby mohla být zaslána uživateli na jeho uživatelské zařízení, je určena funkce `getConfiguration()`. Speciální funkcí, kterou může volitelně vývojář implementovat, je funkce `getData()`. V případě, že úloha například počítá nějaká pro ni specifická data, jež by měla být na uživatelském zařízení zobrazena, tato funkce slouží právě k získání těchto dat.

Poslední funkcí, kterou musí vývojář konkrétní úlohy implementovat je funkce `reload Instances()`. Tato funkce je rozšiřitelným systémem volána vždy při jeho spuštění. Musí být implementována tak, aby získala z databáze uložené konfigurace již dříve vytvořených instancí dané automatizační úlohy a těmto instancím znovu vytvořit v *manažerovi* objekty.

Při ukončování běhu rozšiřitelného systému, je nutné před smazáním manažera úlohy nejdříve smazat všechny jím vytvořené objekty instancí ze systému. K tomuto účelu slouží funkce `deleteAllInstances()`. Funkcí `suicideInstance()` je nakonec umožněno instancím úlohy, jež této funkci předají svůj identifikátor, se za běhu systému smazat z rozšiřitelného systému.

5.3 Řídicí komponenty

Jak již bylo v návrhu v sekci 4.2.2 popsáno, jedním z důležitých předpokladů pro všechny řídící komponenty je, aby každá byla spuštěna v rozšiřitelném systému pouze jedenkrát. Pro zajištění tohoto požadavku byla při jejich implementaci použita obdoba návrhového vzoru *Singleton*. Informace o tomto návrhovém vzoru byly při realizaci čerpány převážně z knihy *Design Patterns* [3]. Každá řídící komponenta je implementována jednou hlavní třídou, jejíž jediná instance je vždy vytvořena při spuštění rozšiřitelného systému a je uchovávána v paměti až do jeho ukončení.

5.3.1 Zavaděč úloh

Tato řídící komponenta slouží k načtení a uchování automatizačních úloh v rozšiřitelném systému. Základní třídou, jež deklaruje virtuální rozhraní *Zavaděče úloh* je virtuální třída `BaseTaskLoader`.

Aby tato komponenta věděla, které automatizační úlohy má do systému načíst, je jí potřeba předat z konfiguračního souboru rozšiřitelného systému cestu ke konfiguračnímu

souboru popisujícímu automatizační úlohy. Ukázka tohoto souboru se nachází v příloze D.2.

Jak již bylo dříve v návrhu popsáno, každá automatizační úloha je v rozšiřitelném systému zastoupena jednou dynamickou knihovnou. Tato knihovna musí vždy obsahovat třídu implementující instanci, dědící od jedné ze tříd `TimedTaskInstance`, `TriggerTaskInstnace` nebo `CombinedInstance`. Také musí obsahovat třídu sloužící jako správce instancí dědící od třídy `TaskManager`. Cesty k těmto knihovnám musí být obsaženy v konfiguračním souboru automatizačních úloh.

Pro účely vytvoření a načtení objektu manažera úlohy je kriticky důležité, aby knihovna s konkrétní automatizační úlohou obsahovala funkci `createTaskManger()`. V té musí být vytvořen dynamicky objekt manažera konkrétní úlohy a musí být na něj navrácena reference. Aby bylo možné knihovnou `dlfcn.h` načíst za běhu systému z dynamické knihovny tuto funkci, je nutné, aby funkce `createTaskManager()` byla zahrnuta v bloku `extern "C"`. Tímto je zajištěno, že symbol popisující funkci je ve formátu jazyka C a funkce knihovny `dlfcn.h`, jež slouží k načítání symbolů, jej tak může bez problému načíst. Tento postup pro načtení objektu manažera byl inspirován dříve zmíněným návodem Aarona Isottona [4].

Aby bylo zpřehledněno uchování informací o úlohách v rozšiřitelném systému, byla pro tento účel implementována pomocná třída `Task`. Tato třída obsahuje referenci na dynamickou knihovnu otevřenou pomocí funkce `openTaskLibrary()` a referenci na vytvořeného manažera automatizační úlohy vytvořeného funkcí `createTaskManager()`. Objekty této třídy zastupující v systému načtené automatizační úlohy jsou v *Zavaděči úloh* uchovávány v kolekci, ve které jsou řazeny podle jejich identifikátorů.

Od virtuálního rozhraní `BaseTaskLoader` dědí třída `TaskLoader`, jež implementuje několik funkcí tohoto rozhraní. Po vytvoření objektu třídy `TaskLoader` je volána jeho členská funkce `createAllTasks()`. Té je předána z konfigurace cesta ke konfiguračnímu souboru popisujícímu automatizační úlohy a jsou v ní zavolány příslušné funkce ke zpracování tohoto souboru a načtení v něm obsažených automatizačních úloh.

Rozšiřitelný systém také po svém spuštění detekuje příjem signálu `SIGUSR1` od operačního systému. V případě, že správce systému tento signál rozšiřitelnému systému zašle, je spuštěna funkce `TaskLoader` objektu `createNewTasks()`. V té je znovu zpracován konfigurační soubor popisující automatizační úlohy a jsou do systému načteny automatizační úlohy, jejichž identifikátor ještě není přítomen v systému. Tímto způsobem není nutné pro načtení nové úlohy rozšiřitelný systém vypínat, nebo restartovat.

5.3.2 Kalendář

Komponenta *Kalendář* je hlavní řídící prvek sloužící ke spouštění a plánování spouštění časovaných a případně také kombinovaných, instancí úloh. Její implementace je realizována třídou `Calendar`.

Hlavním prvkem této třídy je asociativní kolekce třídy `std::multimap`, do níž je možné vkládat dvojce klíč-hodnota, kdy je však možné mít uchováno více hodnot pro jeden klíč. Do objektu této třídy poté dochází k vkládání dvojic, kdy klíčem je aktivační čas, při kterém má dojít ke spuštění instance úlohy, a hodnotou je reference na instanci, jež se má spustit.

K zajištění správného spouštění naplánovaných úloh byl implementován funkcí `run Calendar()` řídící algoritmus 5.1, který zajišťuje správné a přesné spouštění naplánovaných instancí úloh a zároveň je nenáročný na výpočetní výkon počítače, na němž je rozšiřitelný systém provozován. Spuštění této funkce je provedeno vždy pouze jednou, bezprostředně po vytvoření objektu třídy `Calendar` v samostatném vlákně. Tím je zajištěno, že je algo-

ritmus ve smyčce spuštěn paralelně ke všem ostatním částem rozšiřitelného systému.

```
1  while Program nebyl ukončen do
2      Čekej až bude kalendář neprázdný;
3      Získej aktuální čas;
4      while Naplánované spuštění s nejmenším aktivačním časem má menší, nebo
         stejný čas jako je získaný aktuální čas do
5          Ulož kopii naplánovaného spuštění s nejmenším aktivačním časem z kalendáře
             do kolekce instancí úloh ke spuštění;
6          Odstraň právě uložené naplánované spuštění z kalendáře;
7          if Kalendář je prázdný then
8              Ukonči cyklus.
9          end
10     end
11     if Kalendář je neprázdný then
12         Získej aktivační čas naplánovaného spuštění s nejmenším aktivačním časem a
             ulož jej jako čas k dalšímu probuzení algoritmu kalendáře;
13     else
14         Nastav čas na probuzení algoritmu na aktuální čas;
15     end
16     Spusť instance úloh uložené v kolekci instancí úloh spuštění (paralelně k
         hlavnímu algoritmu kalendáře v samostatném vlákně);
17     Uspi algoritmus do získaného času k probuzení, nebo dokud není naplánováno do
         kalendáře spuštění s nižším aktivačním časem, než je získaný čas k probuzení;
18 end
```

Kód 5.1: Hlavní řídící algoritmus kalendáře

K umožnění časovaným instancím úloh plánovat svá spouštění libovolně do kalendáře a také k umožnění jejich naplánování mazat bylo v kalendáři implementováno několik veřejných funkcí. Argumentem, jenž těmto funkcím vždy musí být předán, je reference na instanci úlohy, s jejíž plánováním má být v kalendáři manipulováno.

Funkce `planActivation()` slouží k naplánování spuštění instance úlohy do *Kalendáře*. Aby bylo možné více způsobů naplánování instance úlohy, je tato funkce přetížena tak, že jsou implementovány tři varianty, které reflektují možnosti plánování časovaných instancí popsané v sekci 5.2.1.

V případě, že této funkci není předán žádný jiný argument než ukazatel na instanci úlohy, je naplánována její aktivace na aktuální čas. Spuštění takto naplánované instance je tak provedeno okamžitě. Je však rovněž možné této funkci navíc předat číslo reprezentující počet vteřin, za kolik se má instance úlohy spustit, nebo přesné datum a čas spuštění ve formě řetězce.

Aby byla instance úlohy schopna odstranit již naplánované spuštění z kalendáře, byly implementovány funkce `removeActivation()` a `removeAllActivations()`. První je nutné předat kromě reference na instanci, jejíž spuštění má být z kalendáře odebráno, také aktivační čas určující, které spuštění se má přesně z kalendáře odstranit. Druhá funkce provede smazání všech naplánování referencí předané instance úlohy.

Ke správnému ukončení činnosti algoritmu kalendáře je implementována funkce `stopCalendar()`, jež je volána pouze jednou a to při vypínání rozšiřitelného systému.

5.3.3 Registr požadovaných senzorických dat

Tato komponenta je v rozšiřitelném systému implementována třídou `DataMessageRegister`, a je určena k řízení spouštění *trigger* a kombinovaných instancí úloh.

Hlavním členským objektem této třídy je stejně jako u *Kalendáře* asociativní kolekce `std::multimap` sloužící k uchovávání záznamů o tom, které instance úloh chtějí přijímat senzorická data z kterých koncových zařízení. V této kolekci je tak klíčem unikátní identifikátor koncového zařízení a hodnotou reference na instanci úlohy, jež má být spuštěna.

Funkce `activateInstances()` je určena ke spuštění instancí úloh. Jsou jí předaná přijatá senzorická data zpracovaná ve formě objektu třídy `DataMessage`. Na základě identifikátoru koncového zařízení, který je v datech obsažen, jsou poté vybrány aktivační záznamy, jež tento identifikátor obsahují, a instance úloh v těchto záznamech jsou následně spuštěny.

Tento registr obsahuje několik funkcí, jež umožňují *trigger* instancím úloh manipulovat se záznamy o tom, ze kterých koncových zařízení chtějí získávat data a na základě jejichž přijetí mají být spouštěny. Funkce `insertEntry()` umožňuje instanci úlohy vložení takového aktivačního záznamu a funkce `removeEntry()` má účel opačný a slouží k smazání aktivačního záznamu podle předaného identifikátoru koncového zařízení. Poslední funkce, jež slouží k manipulaci s aktivačními záznamy, je `removeAllEntries()`. Ta provede smazání všech aktivačních záznamů instance úlohy z *Registru požadovaných senzorických dat.*

5.4 Komunikační rozhraní

Nutnou součástí rozšiřitelného systému pro automatizaci domácnosti jsou rozhraní umožňující rozšiřitelnému systému komunikovat s ostatními částmi *BeeeOn* systému. Tato kapitola je věnována způsobům, jakými byla všechna tato rozhraní implementována.

5.4.1 Implementace rozhraní pro příjem

Jedněmi z hlavních komunikačních rozhraní rozšiřitelného systému je dvojce rozhraní sloužící k přijímání požadavků od uživatelů a k přijímání senzorických dat. Tato rozhraní jsou implementována jako konkurentní servery komunikující za pomoci *unixových soketů*. Díky tomu by měly být schopny obsloužit větší množství připojení najednou.

Pro účely obou serverů byly implementovány základní třídy `Server` a `Session`. Třída `Server` při své konstrukci vyžaduje z konfiguračního souboru číslo portu, na kterém má přijímat spojení a počet pracovních vláken, jež jsou určena ke zpracování všech těchto spojení. Vhodným zvolením počtu těchto vláken je tak možné dimenzovat počet spojení, které dokáže server najednou obsloužit a zpracovat. Objekt třídy `Session` je vytvořen při úspěšném navázání spojení s klientem a slouží ke zpracování přijaté zprávy a k zaslání případné odpovědi.

K implementaci těchto tříd byla použita dříve popsaná knihovna `Asio`. Hlavním zdrojem, z něhož byly čerpány informace o tom, jak správně tuto knihovnu používat, byla kniha *Boost.Asio C++ Network Programming* [9].

Jelikož je cílovým zařízením pro běh rozšiřitelného systému stejný počítač, na kterém běží také služby *UI Server* a *ADA Server Receiver*, od nichž mají serverová rozhraní přijímat zprávy, je pevně zvolena jako cílová adresa pro běh těchto rozhraní adresa lokálního stroje 127.0.0.1. Kdyby v některé serverové službě nastala chyba, a tak nebylo spojení úspěšně zpracováno do dvou vteřin, je na základě vypršení časového limitu ze strany rozšiřitelného systému ukončeno.

V rámci třídy implementující specifické rozhraní, jež dědí třídu `Server`, je nutné implementovat členskou funkci `startAccept()`. Tato funkce je zavolána vždy, když je navázáno nové spojení a mělo by tak v ní dojít k vytvoření objektu dědícího od třídy `Session`, ve kterém bude spojení zpracováno.

5.4.2 Rozhraní pro příjem senzorických dat

K realizaci tohoto rozhraní jsou implementovány třídy `GatewayServer` a `GatewaySession`, jež přímo dědí od tříd `Server` a `Session`. Je očekáváno, že na port tohoto rozhraní budou zasílány zprávy obsahující senzorická data z koncových zařízení ve formátu, jehož ukázka je zobrazena v příloze C.1.

Objekt `GatewaySession`, vytvořený při navázání spojení s tímto rozhraním obsahuje funkci `processMessage()`, jíž jsou přijatá senzorická data ve formě řetězce předána. V této funkci je poté vytvořen objekt třídy `DataMessageParser`, který obsahuje funkci `parseMessage()`, jíž je přijatá zpráva zpracována do objektu struktury `DataMessage`. Ta je nakonec předána k dalšímu zpracování funkci `activateInstances()` *Registru požadovaných senzorických dat*.

5.4.3 Rozhraní pro příjem uživatelských požadavků

Toto rozhraní je implementováno třídami `UserServer` a `UserSession`. Tomuto rozhraní jsou serverovou službou sloužící pro komunikaci s uživatelskými zařízeními přeposílány uživatelské požadavky v jazyce JSON určené pro ovládání rozšiřitelného systému z uživatelského zařízení. Důvodem, proč nebyl pro realizaci těchto požadavků použit jazyk XML, jenž je aktuálně užíván ke komunikaci uživatelských rozhraní se serverem, je jeho očekávané brzké nahrazení alternativou právě v jazyce JSON. Ukázky těchto požadavků jsou obsaženy v příloze E.

Stejně jako u *Rozhraní pro příjem senzorických dat* i toto rozhraní implementuje funkci `startAccept()`. Při navázání spojení je v této funkci vytvořen objekt třídy `UserSession`, ve kterém je následně spuštěna funkce `processMessage()`, jež slouží ke zpracování přijatého požadavku. V té je nejdříve identifikován druh požadavku a poté je zpracován za pomoci objektu třídy `UserMessageParser` do jedné z programových struktur sloužících k uchování informací o požadavku. Těmito strukturami jsou `CreateMessage`, `ChangeMessage`, `DeleteMessage`, `GetInstIdsMessage`, `GetConfMessage` a `GetDataMessage`.

Každý zpracovávaný požadavek vždy musí obsahovat identifikátor automatizační úlohy, jíž je směřován. Podle tohoto identifikátoru je z komponenty *Zavaděč úloh* získána reference na *manažera* dané automatizační úlohy. Poté jsou zavolány adekvátní funkce tohoto *manažera* určené k vykonání přijatého požadavku.

Pokud je požadavek úspěšně proveden, je uživateli zaslána odpověď v jazyce JSON značící, že byl požadavek úspěšně zpracován. V případě požadavků, jež mají uživateli vracet také nějaká konkrétní data, jsou tato data přidána na konec zprávy.

V momentě, kdy by jakýkoliv uživatelský požadavek nemohl být korektně zpracován, nebo obsahuje nevalidní nebo neplatné informace, je uživateli odeslána jako odpověď zpráva značící chybový stav.

5.4.4 Databázové rozhraní

Toto rozhraní je implementováno třídou `DatabaseInterface`. Je u něj stejně jako u řídících komponent využito návrhového vzoru *Singleton*. Hlavním důvodem použití tohoto vzoru je zajištění jednotného přístupu do databáze skrze celý rozšiřitelný systém.

Při spuštění rozšiřitelného systému je vytvořen jeden objekt této třídy, jemuž je předán z konfigurace počet spojení, které má trvale udržovat se systémem řízení báze dat. Během běhu rozšiřitelného systému tato spojení zůstávají neustále otevřená a jsou využívána všemi komponentami, jež s databází potřebují komunikovat.

Další předávanou informací při vytváření tohoto rozhraní jsou údaje potřebné pro navázání spojení s databází z konfiguračního souboru ve formě řetězce. Součástí těchto údajů je port, na kterém naslouchá systém řízení báze dat, název databáze, s níž má být operováno, jméno uživatele, skrze kterého bude s databází komunikováno, heslo tohoto uživatele a časový limit, v němž musí systém řízení báze dat stihnout odpovídat na dotazy.

Aby byla zjednodušena komunikace s databázovým systémem, byla implementována funkce `makeNewSession()`. Tato funkce vytvoří jedno sezení třídy `soci::session` nad již existujícími spojeními určenými ke komunikaci s databází a navrátí na něj referenci. Tato třída má přetížený operátor <<, kterým je možné rovnou zasílat databázi dotazy v jazyce SQL. Informace o použité knihovně *soci* byly čerpány převážně z oficiální dokumentace [5].

5.4.5 Rozhraní pro komunikaci s bránou

Ke komunikaci s bránou je určeno rozhraní implementováno třídou `GatewayInterface`. To je implementováno za pomoci knihovny *Asio* jako synchronní klient a umožňuje zasílat službě *ADA Server Sender* zprávy sloužící ke komunikaci s bránami. Je tudíž určeno převážně jako nástroj pro automatizační úlohy, kterým je tímto rozhraním umožněno měnit stavy aktuátorů nebo zjišťovat, jestli je nějaká konkrétní brána dostupná v síti *Internet*.

Jak již bylo v návrhu rozšiřitelného systému popsáno, toto rozhraní využívá již existujícího interního XML protokolu, jenž slouží ke komunikaci služeb *UI Server* a *ADA Server*. Ukázka používaných zpráv tohoto protokolu je uvedena v příloze C.2.

Pro komunikaci s bránou byly implementovány dvě funkce, po jejichž vytvoření jsou zkonstruovány příslušné zprávy v interním XML protokolu, vytvořeno spojení se službou *ADA Server Sender*, které je následně zpráva odeslána. Odpovědí, pokud byla zpráva úspěšně doručena bráně, je kód s číslem 0. Jakékoliv jiné číslo značí chybový stav.

Funkce sloužící k zaslání požadavku pro změnu stavu aktuátoru má název `sendSetState()`. Této funkci je potřeba předat identifikátor brány, identifikátor koncového zařízení, číslo modulu na zařízení, jež je aktuátorem, a hodnota, která má být na aktuátoru nastavena. Na základě těchto informací je následně vytvořena a odeslána zpráva interního protokolu `SWITCH`.

Funkce `pingGateway()` slouží k zjištění, jestli je brána, jejíž identifikátor je předán této funkci, dostupná v síti *Internet*. V případě, že brána na zprávu odpoví a je tudíž dostupná, vrací funkce `pingGateway()` hodnotu `true`, jinak vrací hodnotu `false`.

Kapitola 6

Implementace konkrétních automatizačních úloh

Pro ověření správné funkčnosti rozšiřitelného systému byly implementovány navržené konkrétní automatizační úlohy ze sekce 3.3. Při implementaci těchto úloh byl kladen důraz na to, aby jejich kód mohl případně sloužit jako reference pro vývoj dalších a případně složitějších automatizačních úloh.

6.1 Obecný hlídač (Watchdog)

Jak již bylo dříve popsáno, po svém spuštění obecný hlídač hlídá hodnoty přijímané z uživatelem zvoleného senzoru a poté na základě porovnávání s uživatelem vybranou hodnotou provede uživatelem zvolenou činnost.

Nejvhodnějším druhem pro tuto úlohu je tudíž *trigger* automatizační úloha. Z tohoto důvodu třída `WatchdogInstance` implementující instanci úlohy hlídače, dědí přímo od třídy `TriggerTaskInstance`, která jí nabízí funkcionalitu potřebnou pro její chod. Třída `WatchdogManager`, jež dědí od třídy `TaskManager` a implementuje virtuální funkce potřebné se správě instancí úloh hlídače.

Podle činnosti, jež má být provedena při splnění operátoru porovnání, je tato úloha rozdělena na tři druhy `NOTIF`, `SWITCH` a `BOTH`. První druh provede změnu stavu aktuátoru, druhý zašle uživateli notifikační zprávu a třetí provede obě předchozí činnosti.

Konfigurace této úlohy předávaná požadavkem pro vytvoření její nové instance vždy musí obsahovat následující položky:

- **type** – Tento řetězec určuje jakého druhu je obecný hlídač.
- **gateway_id, device_euid, module_id** – Číselné identifikátory brány, koncového zařízení a modulu (senzoru), ze kterého má být hlídačem přijímána hodnota k porovnání.
- **comp_operator** – Operátor ve formě řetězce, podle něhož bude provedeno porovnání. Jedná se buďto o hodnotu `LT` pro menší než nebo `GT` pro větší než.
- **value** – Číselná hodnota, se kterou se mají přijatá senzorická data porovnávat.

Pokud je zvoleným druhem hlídače `SWITCH` nebo `BOTH`, musí také navíc obsahovat tyto položky identifikující aktuátor, jemuž má být změněn stav:

- **a_gateway_id, a_device_euid, a_module_id** – Číselné identifikátory brány, koncového zařízení a modulu, jehož stav má být při splnění podmínky operátoru změněn.
- **a_value** – Číselná hodnota, na kterou má být nastaven aktuátor při splnění podmínky operátoru porovnání.

V případě, že je úloha druhu `NOTIF` nebo `BOTH`, musí také navíc konfigurace obsahovat:

- **notification** – Uživatelem zvolená zpráva ve formě řetězce, jež mu má být při splnění podmínky zaslána na jeho uživatelské zařízení.

Při předání konfigurace pro vytvoření nebo změnu instance této úlohy *manažerovi* je vždy provedeno několik kontrol. Je ověřeno, že uživatel vytvářející instanci úlohy opravdu vlastní zařízení, jejichž identifikátory jsou zaslány v konfiguraci. Je také kontrolováno, že v položkách **type** a **comp_operator** jsou obsaženy pouze úlohou specifikované řetězce.

K uložení konfigurace instance úlohy obecný hlídač je určena databázová tabulka *task_watchdog*. Ta obsahuje sloupce s názvy a datovými typy, které jsou totožné s názvy a datovými typy údajů v popsané konfiguraci. To, že budou do tabulky vkládány pouze identifikátory existujících zařízení, je zajištěno příslušnými cizími klíči do již existujících tabulek uchovávajících údaje o těchto zařízeních.

Hlavní algoritmus *Obecného hlídače* je implementován tak, že při prvním přijetí senzorických dat je uložena hodnota z požadovaného modulu a až při přijetí dalších dat z tohoto modulu je teprve provedeno první porovnání. Splnění podmínky nastane v případě `LT` operátoru, kdy poslední přijatá hodnota je větší než hlídaná hodnota a nově přijatá hodnota je menší než hlídaná hodnota. U operátoru `GT` je průběh opačný. Tímto je zajištěno, že bude uživatelem definovaná operace provedena pouze, když měřená hodnota projde přes hlídanou hodnotu uživatelem zvoleným směrem.

Aby bylo zabráněno rychle se opakujícímu vykonání uživatelem definované operace v momentě, kdy by měřená hodnota oscilovala kolem hlídané hodnoty, je do úlohy implementován časový limit, jenž zajistí, že operace bude provedena maximálně jednou za deset vteřin.

6.2 Kontrola živosti brány a koncových zařízení (AliveCheck)

Automatizační úloha sloužící k periodické kontrole, že brána a její koncová zařízení jsou dostupná v síti *Internet*, je implementována třídami `AliveCheckInstance` a `AliveCheckManager`. Jelikož byl jako nejvhodnější druh pro realizaci této úlohy zvolena časovaná automatizační úloha, dědí adekvátně třída `AliveCheckInstance` od třídy `TimedTaskInstance`.

Konfigurace nutná pro vytvoření této automatizační úlohy obsahuje následující položky:

- **gateway_id** – Číselný identifikátor brány, na níž má být prováděna kontrola živosti.
- **send_notif** – Boolovská hodnota značící, jestli uživatel chce, aby mu byly zasílány notifikační zprávy v případě nedostupnosti jeho zařízení.

Pro uložení konfigurace této automatizační úlohy je určena databázová tabulka *task_alive_check*, ta obsahuje sloupce se stejným pojmenováním a datovými typy jsou položky konfigurace. V této tabulce je také zaveden cizí klíč, který zajišťuje, že do sloupce *gateway_id* nebude vložen identifikátor neexistující brány.

Při vytvoření instance této úlohy dojde k naplánování jejího spuštění náhodně v rozmezí 0–60 vteřin. Při každém následném spuštění je provedeno další naplánování spuštění za 30 vteřin. Tímto způsobem je zajištěno periodické spouštění instancí této úlohy.

Důvodem, proč není instance úlohy spuštěna po vytvoření okamžitě, je hlavně případ, kdy by bylo nutné rozšiřitelný systém restartovat. Bez náhodného naplánování z databáze znovu načtených instancí úlohy by rozšiřitelný systém skokovitě prováděl spuštění všech instancí najednou a spotřebovával tak nárazově vyšší výkon počítače.

Ve schématu databáze B.1 se nachází tabulky reprezentující informace o bránách a koncových zařízení *gateway* a *device*. Tyto tabulky obsahují sloupec *status*, jenž slouží k označení dostupnosti konkrétního zařízení. *Google Android aplikace* na uživatelském zařízení má implementovanou funkcionalitu, kdy v případě, že je v tabulce v tomto sloupci hodnota `unavailable`, označí v uživatelském rozhraní zařízení jako nedostupné. Pokud je naopak tato hodnota `available`, je označeno jako dostupné.

Při každém naplánovaném spuštění je pomocí funkce rozhraní pro komunikaci s bránou `pingGateway()`, které je předán z konfigurace instance úlohy identifikátor hlídané brány, zjištěno jestli je uživatelem zvolená brána dostupná. V případě, že není, je v databázi brána označena jako nedostupná. V momentě, kdy se brána znovu připojí, je touto automatizační úlohou znovu označena jako dostupná.

Každé koncové zařízení připojené k této bráně je poté zkontrolováno tak, že je z řádku tabulky `device` reprezentujícího dané zařízení získán poslední čas, kdy zařízení zaslalo data, ze sloupce *measured_at*. Ten je poté sečtený s třikrát vynásobenou hodnotou sloupce *refresh* a pokud je vypočtený čas větší než aktuální čas, je zařízení označeno v databázi jako nedostupné. Pokud po nějaké době začne koncové zařízení znovu zasílat měřená senzorická data, tak je touto úlohou znovu označeno jako dostupné.

Kapitola 7

Testování a integrace

Tato kapitola je zaměřena na způsoby, jakými byl implementovaný rozšiřitelný systém pro automatizaci domácnosti testován a následně se věnuje tomu, jak byl systém nasazen, a integrován na server *BeeeOn* systému.

Rozšiřitelný systém byl vyvíjen a prvotně testován na osobním počítači, jenž se skládá z dvou jádrového procesoru Intel® i7® 3517U 1.90 GHz s 4 GB operační paměti a operačním systémem GNU/Linux Xubuntu 14.04 s jádrem 3.16. Každá komponenta systému a její funkce byla nejdříve ověřena v testovacím prostředí, jestli funguje tak, jak je od ní podle návrhu očekáváno, než byla finálně integrována do zbytku rozšiřitelného systému.

Celý rozšiřitelný systém byl v konečné fázi integračně testován a dynamicky analyzován metodou black box. Při tomto druhu testování je chování systému ověřováno tak, že na určitou množinu vstupních dat systém reaguje, jak se z jeho návrhu očekává.

Pro tento účel bylo vytvořeno množství testovacích zpráv, které byly zasílány rozhraní určených k jejich příjmu. Následně z odpovědí na tyto zprávy, či případně ze souboru obsahujícího provozní záznamy bylo ověřováno, že rozšiřitelný systém reaguje správně. Některé druhy uživatelských požadavků také ovlivňují data uložená v databázi, proto bylo u těchto případů ověřováno, že jsou data v databázi korektní.

K původním navrženým automatizačním úlohám byla navíc přidána úloha *FireHazard* určená převážně k ověření správného fungování kombinovaných automatizačních úloh. Byla implementována tak, že je z uživatelem vybraného senzoru hlídána jeho měřená hodnota a v případě, že tato hodnota přesáhne v konfiguraci zvolenou hranici, je spuštěna sekvence přepínající uživatelem zvolený aktuátor. Tato sekvence vždy po vteřině aktuátor sepne, po dvou vteřinách vypne a opakuje se, dokud z vybraného senzoru nepřijde hodnota menší, než je hlídaná hodnota. Touto testovací úlohou bylo úspěšně ověřeno, že je možné v rozšiřitelném systému kombinovat bez potíží jak okamžitou reakci na přijatá senzorická data, tak spuštění instance úlohy s časovým odstupem.

V rámci testování výkonnosti na osobním počítači autora této práce bylo například ověřeno, že systém zvládne naráz bez potíží provozovat 5000 instancí úlohy *AliveCheck* a 5000 instancí úlohy *Watchdog*. Několika hodinové spouštění tohoto výkonnostního testu na testovacím počítači s průběžným zasíláním senzorických dat vyžadovalo maximálně 22 MB operační paměti a najednou bylo využíváno nejvýše 26 programových vláken. I tak velké množství spuštěných instancí nemělo na testovacím počítači vliv na jejich činnost a na dobu odezvy systému na uživatelské požadavky.

Výsledný implementovaný rozšiřitelný systém byl nakonec integrován na vývojový *BeeeOn* server s názvem `ant-2`, jenž je fyzicky umístěn v budově Fakulty informačních technologií na VUT v Brně. Procesorem tohoto počítače je čtyřjádrový Intel® Xeon®

E5410 2.33 GHz. Tento počítač také disponuje 10 GB operační paměti a operačním systémem CentOS ve verzi 7.1 s GNU/Linuxovým jádrem verze 3.10.

Na tomto stroji byl rozšiřitelný systém spuštěn několik týdnů a průběžně bylo na implementovaných úlohách testováno, jestli celou dobu funguje korektně. Správné fungování automatizačních úloh měnících stavy aktuátorů bylo také ověřeno na zásuvce od firmy Jablotron, kterou je možné po připojení do systému *BeeeOn* zapínat a vypínat.

Jelikož měl být v rámci této práce proveden přechod z XML komunikačního protokolu mezi *UI Serverem* a uživatelskými zařízeními na jeho JSON variantu, byl protokol realizující uživatelské požadavky rozšiřitelného systému implementován tak, aby tuto skutečnost reflektoval. Z tohoto důvodu nebyla zatím v uživatelských rozhraních systému BeeeOn implementována grafická podpora pro spouštění a ovládání automatizačních úloh v rozšiřitelném systému.

Kapitola 8

Závěr

Výsledkem této bakalářské práce je rozšiřitelný systém, který do serverové části projektu inteligentní domácnosti *BeeeOn* přidává možnost automatizovaného zpracování z domácností získávaných dat a také na výsledky zpracování reagovat zasláním notifikační zprávy uživateli nebo přepnutím aktuátoru.

Tento systém umožňuje vývoj automatizačních úloh, jež slouží k realizaci konkrétních případů automatizace, a poskytuje prostředky pro jejich řízení a správu. Jeho rozšiřitelnost spočívá v poskytování prostředků pro vývoj různých forem úloh a v umožnění jejich následného nasazení do tohoto systému. Byly navrženy a implementovány konkrétní úlohy: Obecný hlídač, Kontrola živosti brány a koncových zařízení a testovací úloha *FireHazard*, kdy každá z nich realizuje jiný druh automatizační úlohy podle jejich způsobu spouštění.

Řešení této práce bylo započato seznámením se s existujícím systémem inteligentní domácnosti *BeeeOn* a s principem fungování všech jeho částí.

Po identifikaci několika příležitostí pro automatizaci domácnosti na základě systémem *BeeeOn* měřených fyzikálních veličin byly navrženy druhy automatizačních úloh, které umožňují tyto příležitosti realizovat, včetně několika konkrétních úloh.

Následně byl navržen rozšiřitelný systém a jeho komponenty umožňující realizaci automatizačních úloh, řídící komponenty umožňující spuštění těchto úloh a několik rozhraní umožňujících rozšiřitelnému systému komunikovat s ostatními částmi projektu *BeeeOn*.

Na základě tohoto návrhu byl rozšiřitelný systém a všechny jeho součásti implementovány. Část tohoto dokumentu řešící implementaci komponent reprezentujících automatizační úlohy také podrobněji popisuje, jakým způsobem musí vývojář konkrétní automatizační úlohy implementovat, aby je bylo možné do rozšiřitelném systému načíst.

Nakonec bylo implementováno několik navržených automatizačních úloh, jež byly následně využity při testování správného fungování rozšiřitelného systému a finálně byl tento systém integrován do serverové části projektu *BeeeOn*.

Kromě vývoje dalších automatizačních úloh jsou možnými budoucími rozšířeními systému, jenž byl vyvinut v rámci této bakalářské práce, například umožnění sdílení jedné spuštěné automatizační úlohy mezi více uživateli nebo poskytnutí možnosti pozastavení vykonávání již spuštěných automatizačních úloh. Tím by mohla být v systému uchována nastavená konfigurace, aby ji uživatel při znovu spuštění pozastavené úlohy nemusel znovu zadávat.

Literatura

[1] *Standard 62-1999, Ventilation for Acceptable Indoor Air Quality*. American Society of Heating, Refrigerating, and Air-Conditioning Engineers, Inc., Atlanta, Ga, 1999.

[2] Beňo, M. *Automatizované zpracování provozních záznamů v systému BeeeOn*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Vampola Pavel.

[3] Gamma, E.; Helm, R.; Johnson, R.; aj. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN 0-201-63361-2.

[4] Isotton, A. *C++ dlopen mini HOWTO* [online]. 2006. [cit. 10.4.2016]. Dostupné z: <http://tldp.org/HOWTO/C++-dlopen/>.

[5] Loskot, M.; Zeitlin, V.; Sobczak, M.; aj. *SOCI - The C++ Database Access Library: Documentation and tutorial* [online]. 2013. [cit. 2016-04-01]. Dostupné z: <http://soci.sourceforge.net/doc/>.

[6] Malý, M. *REST: architektura pro webové API* [online]. 2009. [cit. 14.4.2016]. Dostupné z: <https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/>.

[7] Milea, A. *Solving the Diamond Problem with Virtual Inheritance* [online]. 2011. [cit. 7.4.2016]. Dostupné z: <http://www.cprogramming.com/tutorial/virtual_inheritance.html>.

[8] Sterling, E.; Arundel, A.; Sterling, T. : *Criteria for human exposure to humidity in occupied buildings. ASHRAE transactions*, roč. 91, č. 1B, 1985: s 611–622. Dostupné z: <http://www.pro.net/sterlingiaq.com/html/photos/1044922973.pdf>

[9] Torjo, J. *Boost.Asio C++ Network Programming*. Packt Publishing Ltd., 2013. ISBN 978-1-78216-326-8.

[10] Valeš, R. *Problematika kondenzace vlhkosti na izolačním skle okna*. IZOLAS spol. s r.o. Dostupné z: <http://www.kplasty.cz/w/k-plasty/files/sklo_problematika_kondenzace_vlhkosti_na_izolacnim_skle_okna.pdf>.

[11] ŠetřímEnergii.cz. *Udržujte v interiéru správnou vlhkost vzduchu* [online]. 2016. [cit. 16.4.2016]. Dostupné z: <http://www.setrimenergii.cz/mikroklima/spravna-vlhkost-vzduchu-v-interieru>.

Přílohy

Seznam příloh

A Obsah CD **41**

B Schéma databáze **42**

C Ukázky komunikačních protokolů **43**

D Ukázky konfiguračních souborů **44**

E Ukázky uživatelských požadavků **45**

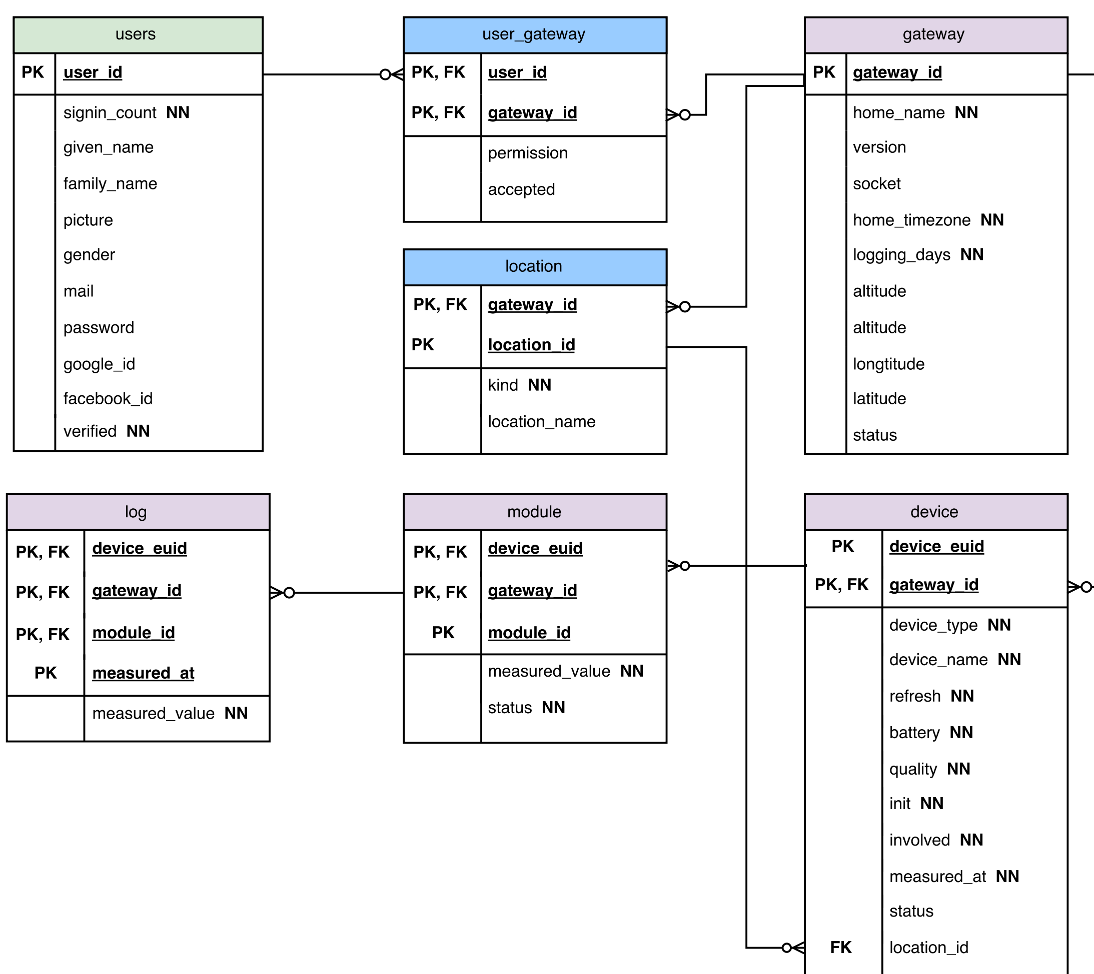
Příloha A

Obsah CD

Přílohou této bakalářské práce jsou také zdrojové soubory rozšiřitelného systému pro automatizaci domácnosti na standardním nepřepisovatelném médiu CD. Na tomto disku jsou ve složce `src` obsaženy všechny soubory implementující hlavní části systému. Ve složce `tasks` se nachází podsložky obsahující implementaci několika konkrétních automatizačních úloh. K snadnějšímu překladu a následnému ovládání rozšiřitelného systému jsou určeny skripty pro interpret příkazové řádky *BASH*. Zdrojové soubory tohoto dokumentu v jazyce LaTeX jsou obsaženy ve složce `latex`. Na médiu jsou také uloženy ukázky konfiguračních souborů, ukázky uživatelských požadavků ve složce `json`, soubor `README.md` s popisem překladu a spuštění rozšiřitelného systému, soubor `baf.sql` sloužící k vložení tabulek rozšiřitelného systému a automatizačních úloh do databáze a soubor s licencí `LICENCE.md`.

Příloha B

Schéma databáze



Obrázek B.1: Schéma základních tabulek databáze

Příloha C

Ukázky komunikačních protokolů

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter_server adapter_id="0x57d3e01695f35" fw_version="v1.2"
protocol_version="1.0" state="data" time="1448964949">
    <device device_id="0x00" euid="0xea000011">
        <values count="6">
            <value module_id="0x00">25.19</value>
            <value module_id="0x01">-0.01</value>
            <value module_id="0x02">35.61</value>
            <value module_id="0x03">68.00</value>
            <value module_id="0x05" status="unavailable">
            </value>
            <value module_id="0x04">100.00</value>
        </values>
    </device>
</adapter_server>
```

Kód C.1: Ukázka zprávy obsahující senzorická data

```
<request type="switch">
    <sensor id="1111" type="0" onAdapter="12345">
        <value>1</value>
    </sensor>
</request>

<request type="ping">
    <adapter id="12345"/>
</request>
```

Kód C.2: Ukázka zpráv interního XML protokolu určeného k operacím s bránami

Příloha D

Ukázky konfiguračních souborů

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <tasks_config path="./tasks_config_ant2.xml"/>
    <user_server threads="10" port="7084"/>
    <gateway_server threads="10" port="7083"/>
    <database sessions="10" connection_string="port = '5432'
       dbname = 'home7' user = '*****' password = '*****'
       connect_timeout = '5'"/>
    <logger log_folder_path="/var/log/baf/" log_level="INFO"/>
    <ada_server_sender port="7081"/>
</config>
```

Kód D.1: Ukázka konfiguračního souboru použitého na vývojovém BeeeOn serveru

```
<?xml version="1.0" encoding="UTF-8"?>
<tasks>
    <task id="1" version="1">
        <name>AliveCheck</name>
        <type>timed</type>
        <path>./tasks/AliveCheck/AliveCheck.so</path>
    </task>
    <task id="2" version="1">
        <name>Watchdog</name>
        <type>trigger</type>
        <path>./tasks/Watchdog/Watchdog.so</path>
    </task>
    <task id="3" version="1">
        <name>FireHazard</name>
        <type>combined</type>
        <path>./tasks/FireHazard/FireHazard.so</path>
    </task>
</tasks>
```

Kód D.2: Ukázka souboru popisujícího automatizační úlohy

Příloha E

Ukázky uživatelských požadavků

```
{
    "msg": "create",
    "user_id": 123,
    "task_id": 1,
    "config": {
        "gateway_id": "1",
        "send_notif": "1"
    }
}
```

Kód E.1: Požadavek CREATE

```
{
    "msg": "delete",
    "user_id": 123,
    "task_id": 1,
    "instance_id": 50
}
```

Kód E.2: Požadavek DELETE

```
{
    "msg": "change",
    "user_id": 123,
    "task_id": 1,
    "instance_id": 50,
    "config": {
        "send_notif": "0"
    }
}
```

Kód E.3: Požadavek CHANGE

```
{
    "msg": "get_inst_ids",
    "user_id": 123,
    "task_id": 1
}
```

Kód E.4: Požadavek GET INSTANCE IDS

```
{
    "msg": "get_conf",
    "user_id": 123,
    "task_id": 1,
    "instance_id": 50
}
```

Kód E.5: Požadavek GET CONFIGURATION

```
{
    "msg": "get_data",
    "user_id": 123,
    "task_id": 5,
    "instance_id": 35,
    "parameters": {
        "type": "graph",
        "range": "month"
    }
}
```

Kód E.6: Požadavek GET DATA