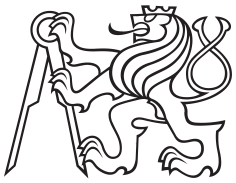
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra radioelektroniky

Systém pro automatizaci domácnosti

Samuel Trávníček

Vedoucí: doc. Ing. Stanislav Vítek, Ph.D.
Studijní program: Elektronika a komunikace
Duben 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Trávníček** Jméno: **Samuel** Osobní číslo: **478068**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro automatizaci domácnosti

Název bakalářské práce anglicky:

Home Automation System

Pokyny pro vypracování:

Cílem práce je návrh a implementace systému pro automatizaci domácnosti, který bude založen na architektuře server-klient.
Při vypracování se řiďte následujícími pokyny:
1) Sestavte přehled již dostupných řešení pro automatizaci domácnosti na českém (případně zahraničním) trhu.
2) Navrhněte a implementujte hardware pro centrální jednotku a zařízení, která budou s centrální jednotkou komunikovat pomocí vhodného rozhraní.
3) Pro celý systém implementujte ovládací software včetně uživatelského rozhraní.
4) Diskutujte spolehlivost a míru zabezpečení navrženého systému.

Seznam doporučené literatury:

[1] WAHER, Peter. Learning internet of things. Packt Publishing Ltd, 2015.
[2] LIN, Huichen; BERGMANN, Neil W. IoT privacy and security challenges for smart home environments. Information, 2016, 7.3: 44.
[3] MORRIS, Meg E., et al. Smart-home technologies to assist older people to live well at home. Journal of aging science, 2013, 1.1: 1-9.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Stanislav Vítek, Ph.D., katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **10.02.2020** Termín odevzdání bakalářské práce: **14.08.2020**

Platnost zadání bakalářské práce: **30.09.2021**

doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) práce	doc. Ing. Josef Dobeš, CSc. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Nejprve bych rád poděkoval rodině, která mě po celou dobu studia podporovala. Dále bych chtěl poděkovat firmě ALLCOMP a.s., ve které jsem měl během let možnost nabrat vědomosti o elektronice a zdokonalit svoje programátorské schopnosti vývojem zakázkových aplikací, a která mi poskytla některé součástky pro sestavení řídicí jednotky. Také bych rád poděkoval panu Václavu Vilímkovi za poskytnutí dat k sestavení orientačního rozpočtu pro zavedení systému pro automatizaci domácnosti. Zejména bych pak chtěl poděkovat panu Vladimíru Červinkovi, který mi po léta praxe ve firmě předával svoje osobní zkušenosti a rady. Nakonec chci poděkovat svému vedoucímu bakalářské práce panu doc. Ing. Stanislavu Vítkovi, Ph.D. za příjemnou spolupráci a bezproblémovou komunikaci.

Prohlášení

Prohlašuji, že jsem zadanou bakalářskou práci na téma „Systém pro automatizaci domácnosti“ vypracoval sám s přispěním vedoucího práce a používal jsem pouze literaturu uvedenou v práci. Zároveň souhlasím s tím, aby byla moje práce půjčována nebo zveřejňována pod podmínkou souhlasu katedry.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací systému pro automatizaci domácnosti v jazyce C++, včetně hardwaru v podobě řídicí jednotky. Zvláštní pozornost věnuje řešení pomocí zásuvných modulů v jazyce LUA. Dále se zabývá implementací administračního a uživatelského rozhraní pro ovládání tohoto systému. Rozebírá bezpečnost a spolehlivost uvedeného řešení a jeho možná vylepšení. Také uvádí již dostupná řešení na trhu.

Klíčová slova: Automatizace domácnosti, elektronika pro řízení, zásuvný modul, LUA, C++

Vedoucí: doc. Ing. Stanislav Vítek, Ph.D.

Abstract

This bachelor's thesis deals with the design and implementation of a system for home automation (including hardware in the form of a control unit) implemented in C++ language. Special attention is paid to the solution using plugin system in the LUA scripting language. The thesis also deals with the implementation of the administration and user interface, used to configure and control the system. It discusses system security, reliability and its possible improvements. It also lists solutions already available on the market.

Keywords: Home automation, control electronics, plugin, LUA, C++

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

V dnešní době zaznamenáváme obrovský růst v oblasti elektroniky. Je to nejrychleji vyvíjející se oblast, které se člověk věnuje. Dnes se nikdo nepodivuje nad tím, že se v autě při otevření dveří samo rozsvěcí světlo, že si auto kontroluje tlak v pneumatikách, že pomocí radaru a ultrazvukových senzorů může zabránit srážce, udržovat bezpečnou vzdálenost za ostatními vozidly nebo zaparkovat. Naopak domácnosti se od roku 1955, kdy byla na území České Republiky elektrifikována poslední obec Hrčava, příliš nezměnily. Stále zapínáme světlo stejným způsobem, tedy vypínačem (či dvěma) v sériovém zapojení se žárovkou. Na rozdíl od aut si v případě chytrých domácností lidé často kladou otázku bezpečnosti a jsou silně ovlivněni sci-fi literaturou a filmy. Přehnané obavy ale nejsou na místě. Lidé si často neuvědomují, že i ty nejjednodušší stroje jako pračka, lednička a mikrovlnná trouba, či složitější, jako výtah, obsahují mikrokontrolér či mikropočítač.

Trend chytrých domácností se objevil poměrně nedávno. Bohužel drtivá většina existujících systémů je založena na myšlence, že chytrý dům je takový, který lze ovládat na dálku, ať už chytrým telefonem, tabletem nebo dálkovým ovládáním. Pokročilejší systémy pak umožňují interakci s hlasovým asistentem a vlastní přizpůsobení chování domácnosti. V této práci se naopak zaměřím na systém, který bude sloužit nejen pro automatizaci běhu domácnosti, ale také umožní implementovat určitou chytrost. Zároveň poskytne vývojáři možnost snadno rozšířit jeho funkčnost formou modulů, čímž zajistí kompatibilitu s libovolným již existujícím elektronickým zařízením.

V kapitole 2 seznámím čtenáře s výběrem několika dostupných řešení pro automatizaci domácnosti. Dále se budu v kapitole 3 zabývat implementací hardwaru, řídicí jednotky, která ovládá elektronická zařízení a čte data ze senzorů. V kapitole 4 budu věnovat pozornost implementaci softwaru, který tvoří největší část této práce. V kapitole 5 rozeberu otázku bezpečnosti a spolehlivosti systému, jak z hlediska fyzického provedení, tak softwarového. Nakonec se v kapitole 7 pokusím odhadnout rozpočet na zavedení tohoto systému do novostavby.

Rád bych upozornil čtenáře na to, že architektura systému, ke které jsem v této práci došel, vznikala od začátku pouze na základě mých dosavadních zkušeností a znalostí bez inspirace od ostatních systémů pro chytrou domácnost nebo automatizaci. Na ostatní systémy jsem se po několika letech

vývoje, kdy jsem vystřídal několik architektur a programovacích jazyků, podíval až v této práci. Tento projekt je pokus o nahrazení staršího systému *Vejminek snů*, který jsem vyvíjel v předchozích letech pro firmu Allcomp a.s.

1.1 Krátce k chytrosti domácnosti

Osobně nemám pojmy „Chytrá domácnost“ nebo „Smart Home“ rád, protože jsou často zavádějící. Dnes se bohužel za chytrou domácnost často považuje taková, kterou lze ovládat dálkově, nejlépe přes internet na mobilním zařízení. Definovat chytrost je nelehký úkol, nicméně tyto dva body považuji za nutné:

- **Bezpečí.** Chytrá domácnost by měla umět reagovat na nečekanou situaci. *Příklad: zapomenutá zapnutá žehlička nebo sporák, prasklé potrubí, požár.*
- **Úspora.** Pokud uživatel zapne světlo, mělo by se při jeho nepřítomnosti automaticky vypnout. Stejně tak topení a ventilace.

Na třetím místě, a příjemnou skutečností, pak mohou být vlastnosti, které poskytují uživateli **komfort**. Tedy dálkové ovládání, časové spínání, hlasový asistent a jiné.

Dalším příkladem chytrosti může být zabezpečovací systém, který v případě výpadku proudu dále plní svou funkci, buď díky záložnímu zdroji nebo alespoň brání vstupu. *Elektronický zámek je bez napájení zamknut, lze jej odemknout pouze mechanicky klíčem, teprve při přiložení napájení se odblokuje.*

Jelikož zajistit chytrost není lehký úkol, rozhodl jsem se tuto práci věnovat systému, který sám o sobě chytrost neimplementuje, ale poskytuje vývojáři nástroje k tomu, aby ji mohl dle svých představ implementovat sám. Dále toto téma rozvinu v kapitole 4.

Kapitola 2
Dostupná řešení na trhu

V této kapitole krátce shrnu již dostupná komerční řešení pro automatizaci domácnosti a jejich klíčové vlastnosti.

2.1 Somfy

Společnost Somfy poskytuje celou řadu produktů pro chytrou domácnost. Venkovní a vnitřní žaluzie, garážová vrata a příjezdové brány, dveře s chytrým zámkem, osvětlení a vytápění, různé senzory, dálkové ovladače, kamery a alarmy. Zároveň také dodává software pro řízení domácnosti zvaný *TaHoma*®. Systém umožňuje kontrolu a dálkové ovládání přes chytrý telefon, simulaci přítomnosti v domě jako ochranu proti nezvaným hostům, časové spínání. Uživatel má možnost si nastavit scénáře například pro příchod a odchod z domu. Synchronizací ovládání topení a žaluzií může dosáhnout úspory energie. V neposlední řadě systém dbá na bezpečí uživatele. Při podezřelých aktivitách zašle upozornění formou notifikace.[1]

2.2 Jablotron

Firma Jablotron[2] není přímo zaměřena na automatizaci domácností. Zabývá se primárně vývojem a výrobou alarmů. Pomocí aplikace *MyJABLOTRON* umožňuje dálkově kontrolovat a ovládat bezpečnostní prvky domácnosti. Také slouží k notifikaci uživatele.

2.3 ABB

Společnost ABB se, mimo jiné, specializuje na domovní elektroinstalace, výrobu vypínačů a zásuvek, termostaty, osvětlení, snímače pohybu a jiné prvky. Poskytuje inteligentní elektroinstalaci *ABB-free@home*®, která uživateli umožňuje automatizovat osvětlení, řídit topení, klimatizaci a žaluzie,

[1] Více o systému TaHoma zde: https://www.somfy.cz/produkty/ovladace-cidla-a-chytra-domacnost/chytra-domacnost

[2] Webové stránky společnosti Jablotron: https://www.jablotron.com/cz/

časové ovládání a simulaci přítomnosti. Uživatel zařízení ovládá chytrým telefonem, tabletem nebo spínačem na zdi.[3]

2.4 Inels

Společnost Inels poskytuje komplexní řešení pro automatizaci budov. Vyrábí převážně bezdrátové produkty, které lze snadno integrovat do již hotových budov. Ovládání je realizováno dálkovými a nástěnnými ovladači, chytrým telefonem nebo tabletem. Zařízení jsou propojena systémem *Smart Home*. Systém zahrnuje monitoring spotřeby, vestavěný alarm, ovládání kamer a jiné.[4]

2.5 xComfort

Bezdrátový systém chytré elektroinstalace *xComfort* umožňuje bezdrátové ovládání osvětlení a spotřebičů, automatické stínění dle počasí a denní doby, vytápění a chlazení, zabezpečení a propojení všech technologií do centralizovaného řízení. Díky bezdrátovému řešení lze snadno implementovat do již hotových domácností. Dálkové ovládání probíhá pomocí chytrého telefonu, tabletu nebo počítače přes webové rozhraní.[5]

2.6 Loxone

Inteligentní elektroinstalace *Loxone* propojuje řídicí jednotky a senzory do hlavní jednotky *Miniserver* pro centralizované řízení. Systém lze rozšířit pomocí modulů například o sběrnici RS485, spínací relé, stmívače, bezdrátovou řídicí jednotku, analogové vstupy a další. Konfigurace probíhá pomocí specializovaného softwaru *Loxone Config* a ovládání pomocí softwaru *Loxone App*, která je dostupná pro známé operační systémy Windows, macOS, Linux, Android a iOS.[6]

2.7 Control4

Systém pro chytrou domácnost *Control4* nabízí ovládání odkudkoliv. Umožňuje ovládání spotřebičů a snímání senzory, regulaci teploty, ovládání audio zesilovačů a přepínačů, video přepínačů a kamer. Poskytuje funkci hlasového asistenta a zahrnuje jak klasické elektroinstalační moduly, tak bezdrátové moduly pro snadnou instalaci. Ovládání probíhá přes chytré telefony, tablety, vestavěné či přenosné dotykové panely a dálkové ovladače.[7]

[3] Více k systému ABB-free@home® zde: https://new.abb.com/low-voltage/cs/nizke-napeti/produkty/automatizace-bytu-a-budov/produktove-rady/abb-free@home

[4] Více k systému Smart Home zde: https://www.inels.cz/home

[5] Domovská stránka projektu xComfort: http://www.xcomfort.cz/

[6] Více informací k chytré domácnosti od společnosti Loxone zde: https://www.loxone.com/cscz/chytry-dum/

[7] Domovská stránka projektu Control4 zde: https://www.control4.cz/

2.8 FIBARO

FIBARO je bezdrátové řešení pro automatizaci domácnosti, které umožňuje snadnou instalaci bez zásahu do stávající elektroinstalace. Poskytuje možnost sledovat stav senzorů a zařízení po celém domě, časové spínání, reakci na počasí a denní dobu. Pomocí notifikací informuje uživatele o možných problémech. Díky standardu Z-Wave může interagovat s moduly různých výrobců.[8]

[8] Domovská stránka projektu FIBARO zde: https://www.fibaro.com/cz/

Kapitola 3

Hardware a firmware

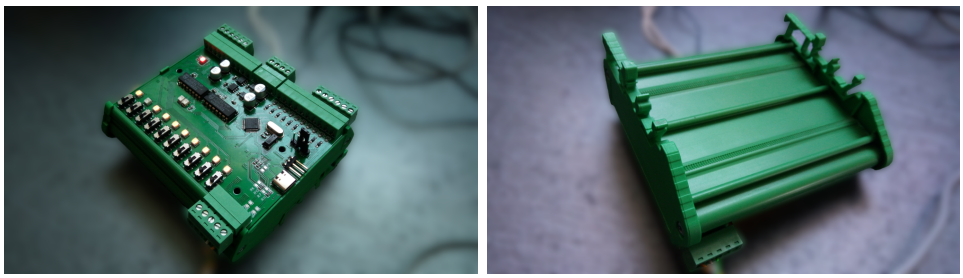
Základním stavebním kamenem systému pro automatizaci domácnosti je hardware. Bez něj by systém nemohl existovat a interagovat s reálným světem. Pro tento systém jsem navrhnul řídicí jednotku, která má následující vlastnosti:

- Deset výstupů, které pracují v režimech digitální a PWM.
- Deset vstupů, které pracují v režimech digitální, analogový, čítač na vzestupnou hranu, čítač na sestupnou hranu.
- Výstupy mají nadproudovou, přepěťovou a tepelnou ochranu.
- Vstupy mají proudové omezení a přepěťovou ochranu.
- Ladění přes USB rozhraní.
- Komunikace po sběrnici CAN 2.0B nebo UART (případně s fyzickou vrstvou protokolu CAN).
- Manuální sepnutí výstupu pomocí přepínače na desce mimo mikrokontrolér, který má ale zpětnou vazbu, pokud uživatel převzal manuální řízení.
- Pro každý výstup je určena kontrolka s jednou barvou pro sepnutý výstup mikrokontrolérem a jednou barvou pro manuálně sepnutý výstup.
- Jednotka je uzpůsobena rozměry a konstrukcí pro montáž na DIN lištu.

Řídicí jednotka dále obsahuje osazenou paměť FLASH, která by měla v budoucnu obsahovat celý souborový systém dostupný přes rozhraní USB. Ten by měl sloužit převážně ke konfiguraci jednotky, ale zároveň umožňoval spouštění skriptů v jazyce LUA s autonomní logikou pro případ ztráty spojení s počítačem. Bohužel jsem narazil na licenční nároky pro komerční použití souborového systému FAT32 (případně exFAT), které předkládá společnost Microsoft. Jedinou možností bylo vytvořit vlastní souborový systém a to je úkol nad rámec této práce.

3.1 Konstrukce a rozměry

Velikost desky plošného spoje na výšku jsem zvolil dle dostupného panelu pro montáž na DIN lištu, tedy 72 mm, a šířku 10 mm, vyhovující požadavkům na rozmístění přepínačů a kontrolek. Tloušťka desky nebyla příliš podstatná, zvolil jsem 1,6 mm. Deska je připevněna k plastovému panelu určenému pro montáž na DIN lištu, která je zachycena na obrázku 3.1b. V levém horním rohu na přední straně (obrázek 3.1a) je umístěn konektor pro výstupy, vedle něj je zdvojený konektor pro napájení a v pravém rohu konektor pro vstupy. V pravém dolním rohu je pak zdvojený konektor pro sběrnici a zem. Pro konektor jsem zvolil svorkovnici s volným protikusem, který umožní snadné odpojení vodičů. Pro ladící rozhraní na sběrnici USB jsem zvolil nejnovější typ konektoru USB-C, který usnadňuje manipulaci díky jeho symetrii a má lepší mechanické vlastnosti než jeho předchozí verze.



Obrázek 3.1: (a) Přední strana řídicí jednotky; (b) Zadní strana

3.2 Mikrokontrolér

Jako mikrokontrolér jsem zvolil STM32F303CC, který vyhovuje všem požadavkům řídicí jednotky a snadno se konfiguruje. Ke konfiguraci jsem použil specializovaný software *STM32CubeMX*. Většinu periferií jsem nakonfiguroval manuálně dle datasheetu [1] a uživatelské příručky [2]. Firmware jsem napsal v programovacím jazyce C jako vhodném kompromisu mezi složitostí jazyka Assembler a hardwarovými nároky jazyka C++. Pro konfiguraci distribuce hodinového signálu a USB rozhraní jsem použil knihovnu HAL, která je volně dostupná ke stažení na webových stránkách[1] společnosti STMicroelectronics. Mikrokontrolér pracuje na napětí 3,3 V.

3.3 Výstupy

Pro každý výstup jsem zajistil následující funkce:

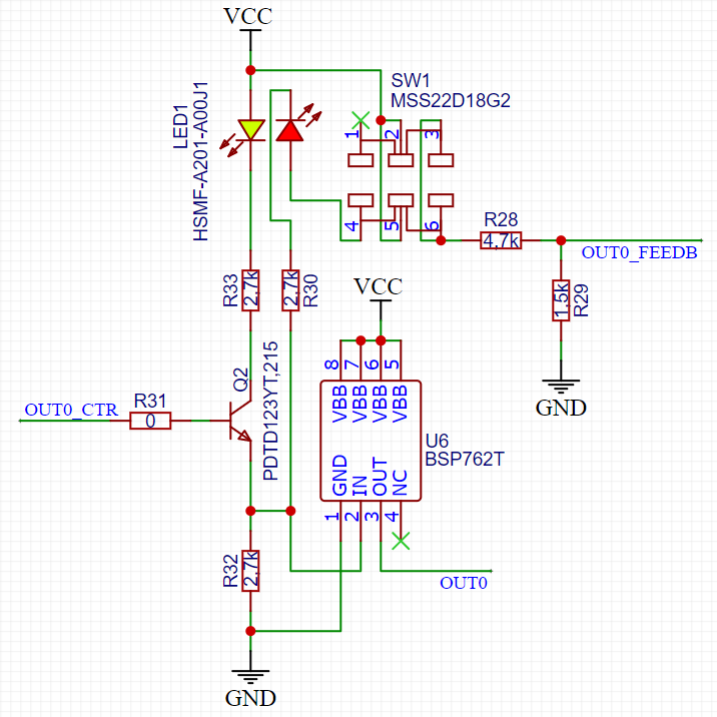
- Kontrolka signalizuje žlutým světlem sepnutí mikrokontrolérem, červeným světlem manuální sepnutí přepínačem.

[1] https://www.st.com/en/embedded-software/stm32cubef3.html

- Mikrokontrolér má zpětnou vazbu, zda je výstup sepnut manuálně.
- Výstup je schopen spínat proud o velikosti až 2 A při frekvenci až 1 kHz.

Díky vyššímu spínacímu proudu na výstupu je možné přímo spínat relé použitá ke spínání spotřebičů bez potřeby dalších obvodů.

Na obrázku 3.2 je schéma zapojení výstupu. Jako spínací prvek jsem použil integrovaný obvod BSP762T. Ten, dle datasheetu [3], obsahuje mosfet v high-side zapojení, nadproudové ochrany, přepěťové ochrany, ochrany proti ESD a teplotní ochranu. Pro sepnutí stačí na vstup přivést napětí o velikosti 2 až 16 V. Odpory R30, R32 a R33 jsem napočítal tak, aby se při sepnutí mikrokontrolérem, přepnutím přepínače SW1, nebo kombinací obou možností, na emitoru tranzistoru Q2, a tedy zároveň na vstupu BSP762T, objevilo napětí o velikosti alespoň 2 V. To kvůli všem požadavkům na kontrolky a manuální režim nebylo jednoduché a hodnota na emitoru při sepnutí obou zdrojů (přepínače a mikrokontroléru) se pohybuje těsně nad dvěma volty. Lepších výsledků bych dosáhl použitím kombinace dvou tranzistorů PNP a NPN, ale z důvodu složitosti návrhu DPS a nutnosti použití další nadbytečné součástky jsem nechal zapojení v původní podobě, která splňuje výše uvedené požadavky na výstup řídicí jednotky.



Obrázek 3.2: Schéma zapojení výstupu

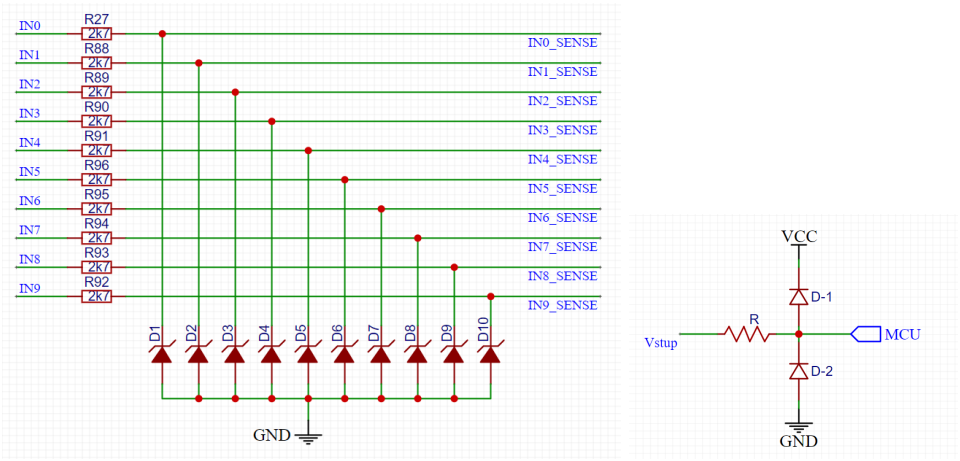
Z přepínače je dále vyvedená zpětná vazba přes napěťový dělič do integrovaného obvodu PCF8574AP. Jedná se o IO expander. Ten jsem použil z důvodu nedostatku pinů na samotném mikrokontroléru. Expander komunikuje přes sběrnici I^2C a je schopen každý pin individuálně nastavit jako

digitální vstup i výstup. Naneštěstí jsem v datasheetu přehlédl informaci o výstupním proudu, který je omezen na pouhých 100 μA. Vstupní proud ale může být až 100 mA. Z tohoto důvodu jsem musel udělat dvě manuální úpravy na plošném spoji (změna polarity indikačních LED). Podrobnosti o expanderu jsou v datasheetu [5].

3.4 Vstupy

Zapojení vstupů je podstatně jednodušší. Každý vstup má omezení proudu sériovým odporem a přepěťovou ochranu realizovanou zenerovou diodou (obrázek 3.3a). Tímto způsobem se zajistí, aby bylo možné na vstup přivést napětí o velikosti až 12 V (napájecí napětí). Toto zapojení velmi dobře funguje pro digitální vstupy. Ukázalo se však, že není vhodné pro analogové vstupy, které jsou kvůli malé strmosti charakteristiky za zenerovým napětím v závěrném směru silně zkreslené.

Budoucím řešením by mohlo být zapojení se sériovým odporem a dvěma diodami, jak ukazuje obrázek 3.3b. Toto zapojení je dokonce možné realizovat bez úpravy plošného spoje, protože mikrokontrolér už takové ochranné diody na svých portech obsahuje. Bohužel jsem v datasheetu nenalezl informaci o maximálním proudu, který tyto diody vydrží.



Obrázek 3.3: Schéma zapojení vstupů (a) použité; (b) náhradní

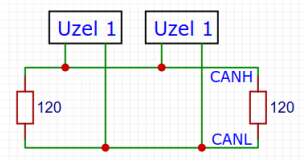
3.5 Komunikační sběrnice

Řídicí jednotka může komunikovat ve třech konfiguracích. Změnou osazení rezistorů R6 až R13, R25 a R26 na obrázku 3.5 lze vybírat mezi sériovou komunikací přes periferii UART, případně UART s napojením na transceiver pro sběrnici CAN, a periferií bxCAN.

Fyzická vrstva protokolu CAN je realizována řadičem zabudovaným v mikrokontroléru (periferie bxCAN) a transceiver TJA1055. Ten zajišťuje převod

logických úrovní 0 a 3.3 V na logické úrovně diferenciálního páru CANL a CANH, které jsou více popsány v datasheetu [4] na straně 16. Na tento pár je navíc připojen filtr,[2] který vyhladí případné rušení.[3] Mimo to umí, na rozdíl od běžných transceiverů, detekovat přerušení jednoho ze dvou vodičů diferenciálního páru a pokud mají komunikující zařízení propojené země, je schopen ustálit komunikaci po jednom vodiči. Takový transceiver nese označení *fault-tolerant*.

Tento transceiver se ještě liší v jedné věci od obyčejných CAN transceiverů. Na oba konce sběrnice CAN se běžně umísťují zakončovací odpory (obrázek 3.4). V tomto případě však jejich úlohu přebírají odpory R15 a R16 (obrázek 3.5 nahoře). Ty právě umožňují rozšířenou funkci transceiveru. Takové zapojení ale znamená, že připojováním více uzlů (komunikujících zařízení) na stejnou sběrnici dochází ke změně jejich celkové impedance (impedance se zmenšuje, protože jsou rezistory zapojené paralelně). Je tedy třeba již při osazování těchto rezistorů počítat přibližně s celkovým počtem uzlů na sběrnici. Naštěstí má transceiver TJA1055 velkou toleranci v rozsahu požadované impedance (500 až 16000 Ω).



Obrázek 3.4: Běžná sběrnice CAN

Takové řešení také omezuje maximální přenosovou rychlost na 125 kBd, místo standardní rychlosti 1000 kBd. S rostoucí délkou (zhruba nad 40 metrů) je třeba snižovat přenosovou rychlost. Podrobnosti o samotném protokolu jsou rozebrány v kapitole 4.

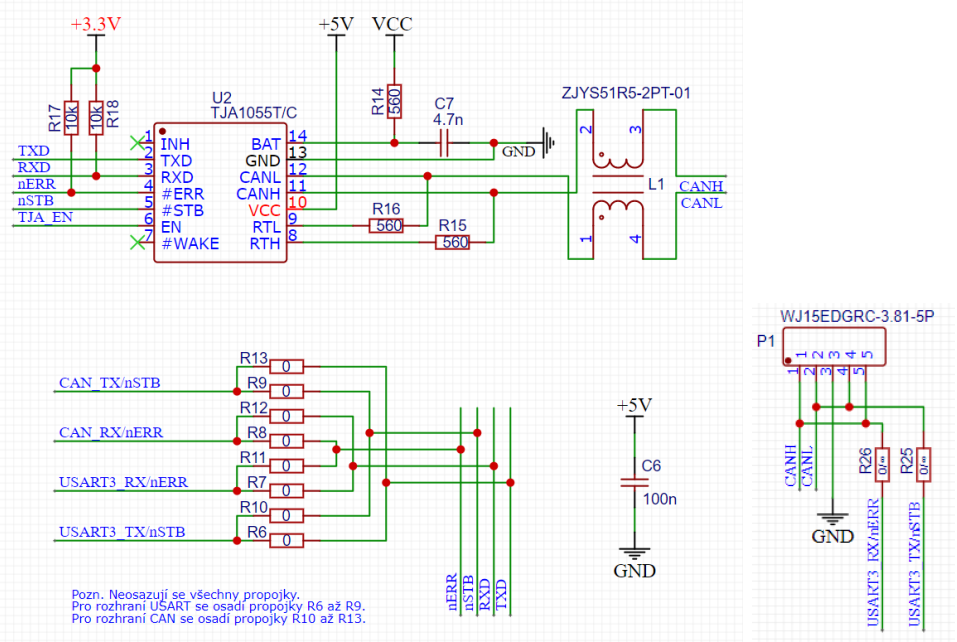
3.6 Napájecí obvod

Jednou z nejdůležitějších součástí řídicí jednotky je její napájecí sekce. Napětí 5 V, které je třeba pro chod sběrnice CAN, a napětí 3,3 V, které je využito zbylou částí řídicí jednotky, zajišťují napěťové stabilizátory (resp. napěťové regulátory s nízkým úbytkem) LE33 a LE50. Na obrázku 3.6 je celé schéma napájecího obvodu.

3.6.1 Kondenzátory v napájecí sekci

Většina integrovaných obvodů se skládá převážně z polovodičů. Výjimkou nejsou ani integrované obvody použité na řídicí jednotce. Takové obvody mají z hlediska napájení tu nepříjemnou vlastnost, že v čase mění svůj odběr. Obecně se dá říci, že s rostoucí frekvencí je toto chování kritičtější. Taková

[2] Na obrázku 3.5 označen jako L1.
[3] TJA1055 má filtr již zabudovaný, ale není na škodu použít ještě jeden externí.



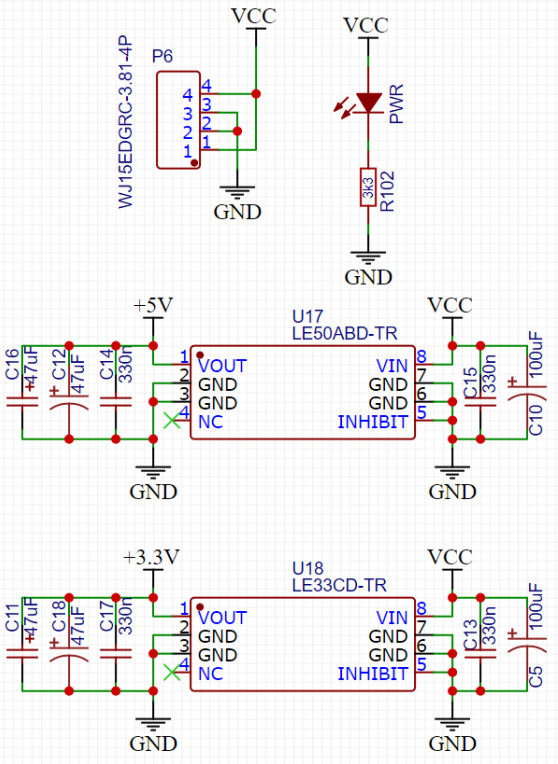
Obrázek 3.5: Schéma zapojení komunikační sběrnice

náhlá změna odběru proudu způsobí chvilkový pokles napětí na zdroji. Jelikož se takové změny dějí často (zvlášť u čipů pracujících na vyšších frekvencích - zejména u mikrokontrolérů, kde odběr závisí, mimo jiné, přímo na firmwaru), dojde k zvlnění napájecího napětí. To samozřejmě ovlivňuje také okolní periferie. U digitálních obvodů to není až takový problém, protože pracují pouze ve dvou diskrétních stavech. Pokud není zvlnění příliš velké, nemělo by dojít k chybným stavům. Daleko větší problém je s analogovými periferiemi, například AD převodníky, které vyžadují přesné napěťové reference. Při zvlnění pak přirozeně dochází k chybnému převodu na digitální hodnotu a tedy celkovému zvýšení standardní nejistoty měření. Z tohoto důvodu se kolem takových periferií navíc osazují blokovací kondenzátory. Ty mají za úkol, ve chvíli, kdy je odběr nízký, akumulovat energii. V případě, že dojde k náhlému vzrůstu odběru, přejde většina zátěže ze zdroje (v našem případě napěťového stabilizátoru) na blokovací kondenzátor. V případě ideálního kondenzátoru je zapojení bez problémů. Stačilo by vybrat libovolnou, dostatečně vysokou, hodnotu a kondenzátor by přesně plnil požadovanou funkci. Problém nastává, kdy je takový kondenzátor použit v reálném světě.

Reálný kondenzátor má, kromě kapacity, také parazitní sériový odpor, sériovou indukčnost a paralelní odpor. Na obrázku 3.7 je jednoduché náhradní schéma. Následující rovnice vyjadřuje jeho impedanci.

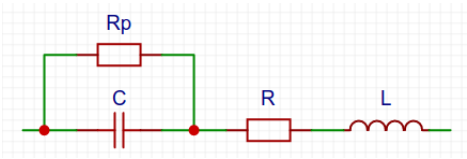
$$Z_{re} = \frac{R_p}{j\omega R_p C + 1} + j\omega L + R \tag{3.1}$$

Na obrázku 3.8 vlevo je vyobrazená frekvenční závislost impedance ideálního kondenzátoru v decibelech, vpravo pak charakteristika reálného kondenzá-



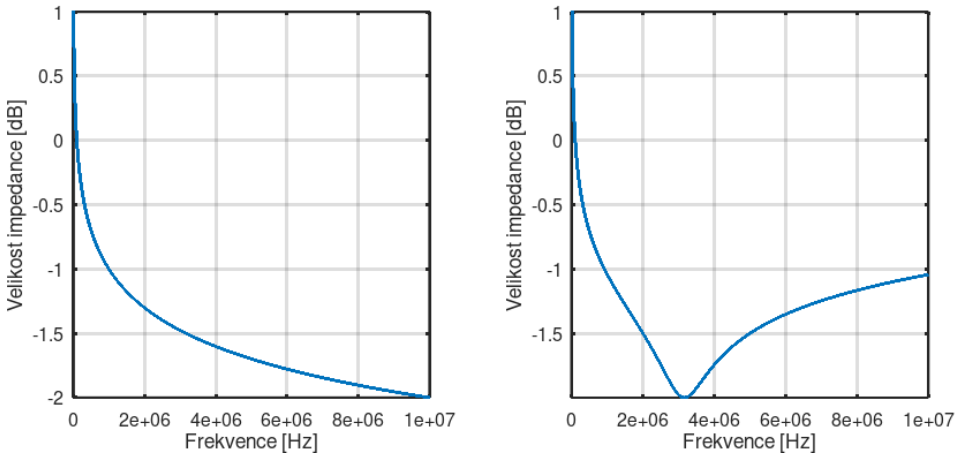
Obrázek 3.6: Schéma zapojení napájecího obvodu

toru.[4] Z grafu je patrné, že spolu indukčnost a kapacita vytváří rezonanční obvod, který má při rezonanci minimální velikost impedance. Takový kondenzátor dokáže na rezonanční frekvenci dodat obrovský proud, ale s rostoucí frekvencí a zároveň impedancí není schopen dodávat dostatečný proud. Pokud nastane špičkový odběr na nějakém z integrovaných obvodů, blokovací kondenzátor sice má dost energie na pokrytí odběru, ale přes vyšší impedanci není schopen v čase dodat dostatek energie. Proto jsem v napájecí sekci použil více druhů kondenzátorů. Obecně se dá říci, že s rostoucími geometrickými rozměry součástky roste její parazitní indukčnost a s klesající parazitní indukčností se rezonanční frekvence zvyšuje. Proto jsem, kromě elektrolytických kondenzátorů s vyšší kapacitou, použil v paralelním zapojení také keramické a tantalové kondenzátory v SMD pouzdře. V takové kombinaci pokryjí širší část spektra než jeden jediný. Navíc má každý integrovaný obvod na řídicí jednotce poblíž svých napájecích pinů umístěný další malý kondenzátor.



Obrázek 3.7: Náhradní schéma reálného kondenzátoru

[4] Zvolil jsem hodnoty $C = 10\ \mu\text{F}$, $L = 10\ \text{nH}$, $R = 10\ \text{m}\Omega$, $R_p = 1\ \text{M}\Omega$

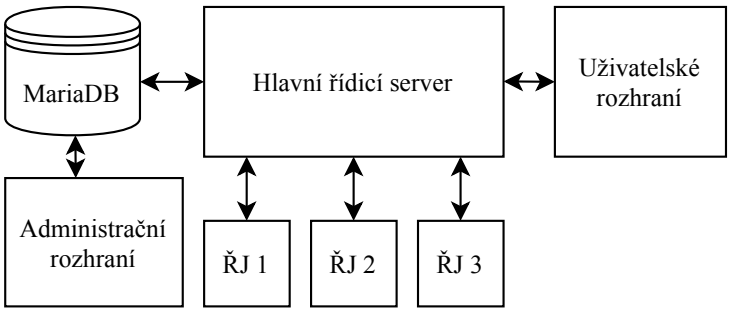


Obrázek 3.8: Frekvenční závislost impedance ideálního (vlevo) a přibližně reálného (vpravo) kondenzátoru v decibelech

Kapitola 4
Software

Celý systém pro automatizaci se skládá z několika nezávislých programů, které navzájem spolupracují. Srdcem systému je hlavní řídicí server, který zajišťuje samotnou logiku. Dále se jedná o „cloudový“ server, zprostředkující komunikaci s řídicími jednotkami. Tento server jsem během vývoje nakonec, implementací zásuvných modulů,[1] integroval do hlavního řídicího serveru. Tedy se ve výsledku nejedná o nezávislou aplikaci.

Pro interakci s uživatelem existují dvě webová rozhraní. Jedno z nich je administrační rozhraní, které nekomunikuje se serverem přímo, ale pouze přes databázi. Druhé je uživatelské rozhraní, které skrze server v reálném čase ovládá řídicí jednotky. Na obrázku 4.1 je velmi zjednodušený diagram celého systému.



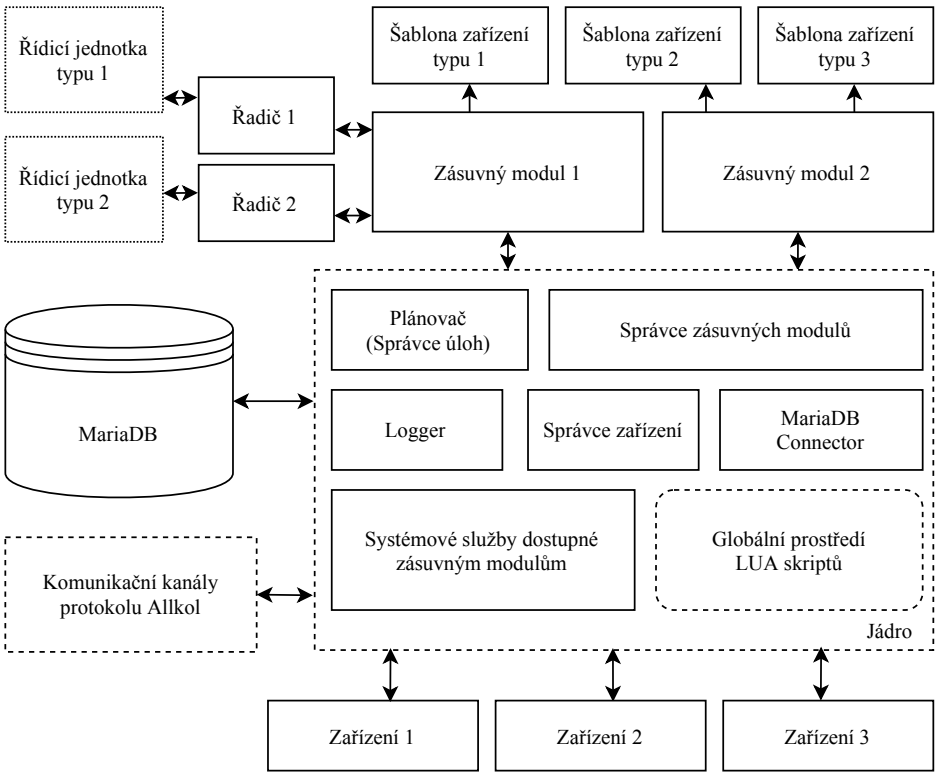
Obrázek 4.1: Diagram systému pro automatizaci

4.1 Řídicí server

Největší část softwaru tvoří hlavní řídicí server. Jako platformu jsem zvolil operační systém Linux, který může běžet bez grafického rozhraní a mít tak minimální požadavky na hardwarové vybavení hostujícího počítače. Zároveň nabízí mocné nástroje, jako jsou snadno použitelné systémové služby, crony a spoustu užitečných systémových volání (například *epoll*). Při použití jednodeskového počítače, jako je Raspberry PI, který bootuje přímo z SD karty,

[1] Více o zásuvných modulech v kapitole 4.1.2.

je první zprovoznění pro kohokoliv velmi snadné, protože stačí naklonovat SD kartu s předpřipraveným systémem. Server je programovaný v jazyce C++ standardu C++17. Pro systém zásuvných modulů jsem zabudoval podporu skriptovacího jazyka LUA verze 5.3, který umožňuje výrazně měnit chování a funkce serveru bez nutnosti opětovné kompilace. Na obrázku 4.2 je znázorněn diagram řídicího serveru.

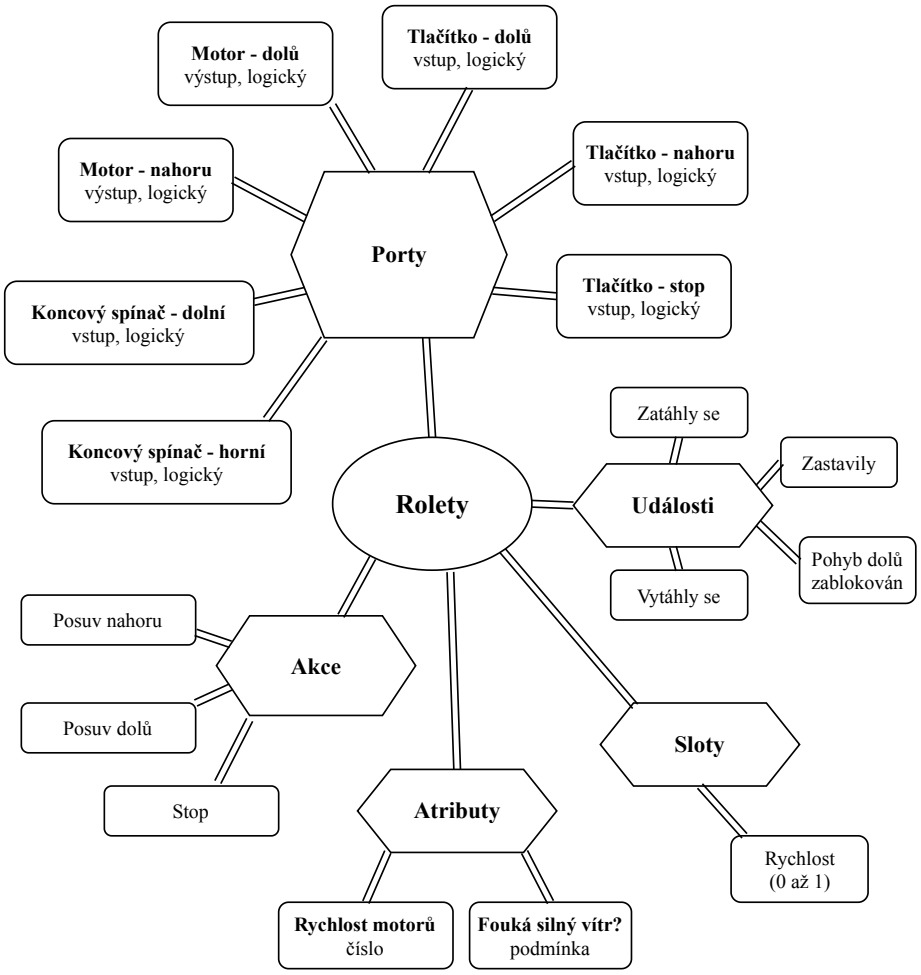


Obrázek 4.2: Diagram hlavního řídicího serveru

4.1.1 Zařízení

Zařízení je objekt, který zastupuje fyzické elektronické zařízení nebo soustavu elektronických zařízení. Může se jednat o vypínač, žárovku, žaluzie, ale i zařízení typu „stmívané osvětlení," které zahrnuje nějaký atenuátor a žárovku. Zařízení jsou načítána z databáze (MariaDB) a konstruována dle příslušných šablon nadefinovaných zásuvnými moduly. Jsou složena z portů, atributů, slotů, signálů, událostí a akcí. Příklad takového zařízení je na obrázku 4.3.

Porty jsou základní součástí každého zařízení a reprezentují fyzické porty na fyzickém zařízení. Mohou mít jednu ze dvou orientací - *vstup* nebo *výstup* - a jsou dvojího typu - *logické* a *diskrétní*. Logické mohou nabývat pouze stavů 0 (LOW) a 1 ($HIGH$). Diskrétní mohou nabývat jakékoliv reálné



Obrázek 4.3: Diagram ukázkového zařízení *Rolety*

hodnoty, která je vyjádřitelná datovým typem *double* v jazyce C++.[2] Původní návrh obsahoval typy *digitální*, *analogový* a *pwm*. Ty jsem však nahradil zmíněnými abstraktnějšími typy, protože jsou použitelné ve všech situacích a nejsou vázány přímo na hardware.

Atributy jsou vstupní parametry zařízení, načtené z databáze. Mohou nabývat typu *Boolean*, *Number*, *String*, *Time* mebo *Condition*. První tři ze jmenovaných typů jsou obyčejné datové typy, které se objevují ve staticky typovaných programovacích jazycích (typ *Number* implementován datovým typem *double*). Typ *Time* je objekt, obsahující textový řetězec času bez data v pevném formátu, nad nímž je možno provádět některé matematické operace jako sčítání a odčítání. Ten může být použit k časově podmíněnému chování zařízení. Poslední uvedený typ, *Condition*, je objekt, který umožňuje dynamicky vyhodnotit podmínku zapsanou ve

[2] Vyjádřitelnost závisí na implementaci, tedy kompilátoru, a cílové platformě.

skriptovacím jazyce LUA.

Sloty jsou parametry zařízení, proměnlivé za chodu. Mohou být typu *Number*, *Boolean* nebo *String*. Jejich účelem je výměna dat mezi řadičem a zařízením.

Signály podobně slouží pro vzájemnou signalizaci mezi řadiči a zařízeními.

Událostmi může zařízení informovat jiné části systému o změně svého stavu. Jiná zařízení na tyto události mohou reagovat svými akcemi.

Akce jsou parametrické úkony zařízení. Může se jednat například o stažení rolet nebo změnu intenzity osvětlení. Akce jsou přizpůsobeny tak, aby mohly být volány z uživatelského rozhraní.

4.1.2 Zásuvné moduly

Zásuvné moduly jsou psány ve skriptovacím jazyce LUA. Obsahují jeden deskriptivní soubor *plugin.lua*, ve kterém je popsán jejich identifikátor, název, autor, popis, verze a další.

```
1 identifier = "cz.travnicek.sample_plugin"
2 label = "Sample Plugin"
3 author = "Samuel Trávníček"
4 description = "This is a sample Plugin Description File for my
      Bachelor thesis."
5 version = "1.0"
6 entry_file = "main.lua"
7
8 -- using Language Codes ISO 639-1 standard
9 primary_language = "en"
```

Jak je vidět z ukázkového kódu výše, v systému je připraven prostor pro vícejazyčnou podporu, pro kterou bohužel v této práci nebyl prostor. Také je zde položka *entry_file*, která systém informuje, z jakého souboru se má zásuvný modul načíst. Protože je tento soubor také v jazyce LUA, může obsahovat i pokročilé skripty, generující parametry zásuvného modulu, s využitím všech výhod skriptovacího jazyka. Zásuvné moduly mají dvě hlavní funkce - definovat šablony zařízení a poskytnout řadiče k obsluze hardwaru.

Šablony zařízení

Šablony zařízení definují všechny parametry rozebrané v předchozí části. Navíc obsahují samotnou logiku zařízení v podobě zpětných volání na události jako změna hodnoty portu, slotu a spuštění signálu. Toto je zdrojový kód pro šablonu zařízení *Rolety* z diagramu na obrázku 4.3.

```
1 local dev = Device:new("blinds", "Rolety")
2
```

```
3  dev:port("motor_up", "Motor - nahoru", { direction = OUTPUT,
       mode = LOGICAL })
4  dev:port("motor_down", "Motor - dolů", { direction = OUTPUT,
       mode = LOGICAL })
5  dev:port("push_button_up", "Tlačítko - nahoru", { direction =
       INPUT, mode = LOGICAL })
6  dev:port("push_button_down", "Tlačítko - dolů", { direction =
       INPUT, mode = LOGICAL })
7  dev:port("push_button_stop", "Tlačítko - stop", { direction =
       INPUT, mode = LOGICAL })
8  dev:port("limit_switch_upper", "Koncový spínač - horní", {
       direction = INPUT, mode = LOGICAL })
9  dev:port("limit_switch_lower", "Koncový spínač - dolní", {
       direction = INPUT, mode = LOGICAL })
10
11 dev:slot("speed", "Rychlost", DRIVER_IN, NUMBER)
12
13 dev:emitted_event("down", "Zatáhly se")
14 dev:emitted_event("up", "Vytáhly se")
15 dev:emitted_event("stopped", "Zastavily")
16 dev:emitted_event("movement_down_inhibited", "Pohyb dolů
       zablokován")
17
18 dev:attribute("motors_speed", "Rychlost motorů", NUMBER)
19 dev:attribute("unsafe_wind", "Fouká silný vítr?", CONDITION)
20
21 dev:action("move_up", "Posuv nahoru", function(instance)
22     instance:port("motor_down"):write(LOW)
23     instance:port("motor_up"):write(HIGH)
24 end)
25
26 dev:action("move_down", "Posuv dolů", function(instance)
27     instance:port("motor_up"):write(LOW)
28     instance:port("motor_down"):write(HIGH)
29 end)
30
31 dev:action("stop", "Stop", function(instance)
32     instance:port("motor_up"):write(LOW)
33     instance:port("motor_down"):write(LOW)
34     instance:fire("stopped")
35 end)
36
37 dev:on("port_changed", function(instance, port, value,
       prev_value)
38     if value ~= HIGH then
39         return
40     end
41
```

```
42      local pid = port:identifier()
43
44      if pid == "limit_switch_upper" then
45          instance:port("motor_up"):write(LOW)
46          instance:fire("up")
47      elseif pid == "limit_switch_lower" then
48          instance:port("motor_down"):write(LOW)
49          instance:fire("down")
50      elseif pid == "push_button_up" then
51          instance:port("motor_up"):write(HIGH)
52          instance:port("motor_down"):write(LOW)
53      elseif pid == "push_button_down" then
54          if not instance:attribute("unsafe_wind") then
55              instance:port("motor_up"):write(LOW)
56              instance:port("motor_down"):write(HIGH)
57          else
58              instance:fire("movement_down_inhibited")
59          end
60      elseif pid == "push_button_stop" then
61          instance:port("motor_down"):write(LOW)
62          instance:port("motor_up"):write(LOW)
63          instance:fire("stopped")
64      end
65  end)
66
67  local scheduler = service('scheduler')
68
69  local function check_wind_condition(instance)
70      warning('Checking wind condition')
71
72      if instance:attribute("unsafe_wind") then
73          instance:port("motor_down"):write(LOW)
74          instance:port("motor_up"):write(HIGH)
75      end
76
77      scheduler:schedule(30000, function()
78          check_wind_condition(instance)
79      end)
80  end
81
82  dev:on("instantiation", function(instance)
83      instance:slot("speed"):write(instance:attribute("
        motors_speed"))
84      check_wind_condition(instance)
85  end)
```

Nejprve se na prvním řádku vytvoří nová šablona. Dále jsou nadefinovány porty, sloty, emitované události, atributy a akce tak, jak jsou zobrazeny na ob-

rázku 4.3. Při vytvoření instance je poté na jeden ze slotů připojeného řadiče zapsána informace o rychlosti motorů. Tato rychlost je načtena z atributu zařízení - tedy z databáze. Šablona dále reaguje na změny vstupních portů - tedy na tlačítka a koncové spínače - příslušným sepnutím motorů. Nutno podotknout, že vypínání koncovými spínači by mělo být ve skutečnosti realizováno na úrovni hardwaru. Software ale může zachovat tuto funkcionalitu pro možnost sledování stavů motorů. Také je zde implementována již určitá forma chytrosti. Zařízení každých 30 vteřin kontroluje, zda nefouká příliš silný vítr. V opačném případě vytáhne rolety nahoru, aby nedošlo k jejich poškození. Také v takové situaci uživateli zabraní v pokusu o stažení dolů.

Řadiče

Řadiče mají za úkol poskytnout systému univerzální komunikaci s externím hardwarem. Stejně jako u zařízení definují své porty, sloty a signály. Porty mohou být i ve více konfiguracích. Při spuštění serveru jsou dle databáze propojeny porty řadičů a zařízení. Pro každý port je vybrána jedna z konfigurací (automaticky nebo preferovaná, pokud je dostupná). Taktéž jsou propojeny signály a sloty. Ukázkovým příkladem použití signálů a slotů by mohl být řadič, který ovládá znakový displej. Ten by měl jeden slot *Text* a dva signály *Zobrazit* a *Vyčistit*. Pro ukázku jsem ale napsal minimalistický řadič, který přečte číslo na vstupním slotu, po přijetí signálu jej porovná s určitou prahovou hodnotou a výsledek zapíše na výstupní slot.

```
1  TestDriver = {}
2
3  function TestDriver:new()
4      local obj = Driver:new("test_driver", "Testovací řadič")
5
6      obj:slot("in", "Vstup", DRIVER_IN, NUMBER)
7      obj:slot("out", "Výstup", DRIVER_OUT, BOOLEAN)
8      obj:signal("trigger", "Spoušť", DRIVER_IN)
9
10     local TRESHOLD = 20
11     local slot_value = 0
12
13     obj:on("slot_written", function(identifier, value)
14         slot_value = value
15     end)
16
17     obj:on("signal_invoked", function(identifier)
18         obj:write_slot("out", slot_value < TRESHOLD)
19     end)
20
21     return obj
22 end
```

4.1.3 Řadič pro řídicí jednotku

Řadič využívá implementované API pro komunikaci po sériovém portu. Na sériovém portu naslouchá jednoduchý převodník, který převede přijatá data na zprávy protokolu CAN 2.0B. Každá zpráva obsahuje identifikátor, který určuje, zda-li má přednost na sběrnici či nikoliv. V tomto identifikátoru je zakódována adresa přijímacího a vysílacího zařízení, zda se jedná o požadavek nebo odpověď, dva prioritní bity a druh požadavku. Pokud je zpráva odeslána na adresu 0, přijmou ji všechna zařízení (broadcast), v opačném případě pouze zařízení s uvedenou adresou. Po identifikátoru následují příznaky protokolu CAN, které v tuto chvíli nejsou důležité. Všechny podrobnosti o protokolu jsou k dispozici ve specifikaci [6]. Dále zpráva obsahuje délku dat. Pokud je nenulová, následují data o maximální délce osm bajtů. Zpráva je ukončena kontrolním součtem CRC a dalšími příznakovými bity. Protože je zdrojový kód řadiče příliš dlouhý, nebudu jej zde uvádět, ale je obsažen v přiložených souborech.

4.1.4 Propojení s ostatními aplikacemi

Aby mohl server komunikovat i s dalšími aplikacemi, vzniknul protokol s pracovním názvem Allkol. Důvod pojmenování nechám na představivosti čtenáře. Protokol definuje podobu přenášených zpráv a způsob přenosu. Přenos může probíhat po protokolech TCP/IP, TLS, WS (WebSocket) a WSS (WebSocket po TLS).

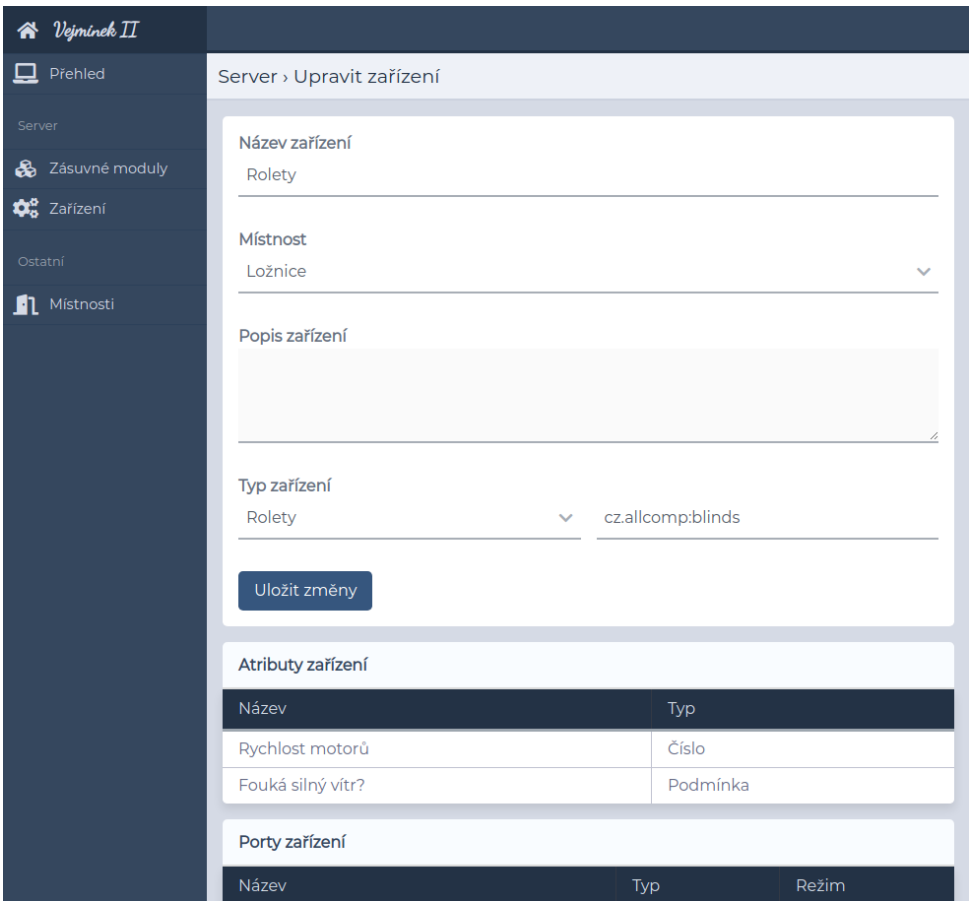
Každá zpráva obsahuje číselný identifikátor, druh požadavku (příkazu), informaci, zda se jedná odpověď na požadavek nebo ne, a samotná data. Odpověď pak navíc obsahuje informaci o tom, zda byl požadavek zpracován úspěšně a v případě neúspěchu je uveden také důvod. Zpráva je přenášena ve formátu JSON bez bílých znaků[3]. Ten umožňuje snadnou práci s daty, zejména ve skriptovacím jazyce JavaScript, pro který byl tento formát původně navržen, a který je použit v uživatelském rozhraní.

Každá externí aplikace musí mít přiřazený *Univerzální unikátní identifikátor* (UUID), kterým se vůči serveru autorizuje, a který zároveň slouží jako privátní klíč. Na straně serveru je pak možno otevřít libovolný počet komunikačních kanálů na zmíněných protokolech a pro každý z nich zvlášť nastavit port, na kterém komunikují, IP adresu, limit počtu klientů a kontrolu klíčů formou whitelistu (pouze uvedené klíče mají přístup), blacklistu (uvedené klíče nemají přístup, ostatní ano) nebo bez kontroly. Jestliže klíč projde kontrolou, může si aplikace zažádat o dostupná zařízení a jejich porty a akce. Samozřejmě může na tyto porty zapisovat, nebo z nich číst, a volat akce s jejich parametry.

[3] Tj. mezer, znaků odsazení, nových řádků a podobně.

4.2 Administrační rozhraní

Administrační rozhraní slouží pro přidávání nových zařízení a pro propojování jejich portů, slotů a signálů s jejich protějšky na řadičích. Veškeré informace o šablonách zařízení, řadičích a samotných zásuvných modulech jsou načteny z databáze. Tyto informace jsou vygenerovány při spuštění řídicího serveru. Rozhraní je implementováno jako webová aplikace v jazycích HTML, SASS, PHP a JavaScript. To zajišťuje, že je spustitelné na všech platformách s webovým prohlížečem s podporou JavaScriptu, což jsou dnes prakticky všechna mobilní zařízení i osobní počítače. Grafické rozhraní je responzivní, takže je snadno použitelné i na chytrých telefonech a tabletech. Na obrázku 4.4 je ukázka nastavení zařízení *Rolety* z předchozí části.



Obrázek 4.4: Ukázka administračního rozhraní

Aplikace využívá šablonovací systém *BladeOne*,[4] který umožňuje snadnou integraci výstupu PHP skriptů do HTML dokumentu. Navíc výsledný kód vypadá graficky lépe a snadněji se čte. Také zajišťuje, že žádná šablona nemá přístup ke všem proměnným systému, ale pouze k těm, které využívá, což snižuje celkovou chybovost při vývoji.

[4]Repozitář systému je zde: https://github.com/EFTEC/BladeOne

4.2.1 Nástroj pro zpracování URL

Aby bylo adresování v aplikaci čitelnější, rozhodl jsem se implementovat vlastní nástroj pro zpracování URL, který podporuje tzv. „Fancy URL.“ Pro návrh jsem se inspiroval u frameworku *Laravel*, ze kterého také původně pochází samotný *BladeOne*. Důvodem, proč jsem nepoužil framework Laravel je jeho robustnost. Po vyzkoušení jsem také narazil na potřeby, které tento systém neuspokojuje.

Tento nástroj jsem napsal podobným způsobem, jako se píší kompilátory. Skládá se z *Tokenizeru* a *Validátoru*. Tokenizer nejprve převede část URL dotazu (za doménou) na tokeny. To jsou v tomto případě lomítka a jednotlivé výrazy mezi nimi. Seznam tokenů je předán validátoru, který provede přiřazení ke správnému vzoru. Vzor slouží k definici validního URL dotazu a extrakci případných proměnných. V následujícím kódu je na prvním řádku ukázka vzoru a na druhém řádku příklad validního dotazu.

```
1 /server/edit-device/{device-id}/port/{port-id}
2 /server/edit-device/1/port/example_port
```

Validátor by v takovém případě vrátil asociativní pole, obsazující dva indexy *device-id* a *port-id*, a jejich hodnoty. Nástroj nezpracovává část URL dotazu za otazníkem, protože to už umí PHP samo o sobě. Takto vypadá gramatika tokenizeru a validátoru:[5]

```
1 word:    *('[a-zA-Z0-9_\-]')
2 param:   '{' word ['?'] '}'
3 pattern: ['/'] [(word | param) *('/' (word | param)) ['/']] EOF
4 sample:  ['/'] [word *('/' word) ['/']] EOF
```

4.3 Uživatelské rozhraní

Uživatelské rozhraní jsem napsal také jako webovou aplikaci v jazycích HTML, SASS a JavaScript. Díky tomu, že jde čistě o JavaScriptovou aplikaci, může být spuštěna ve webovém prohlížeči přímo (bez použití webového serveru jako je například Apache) nebo být zabalena do nějaké nativní aplikace. Pro tyto účely jsem napsal JavaScriptovou knihovnu pro protokol Allkol. Knihovna využívá API pro WebSocket, které jde dnes dostupné již ve všech majoritních webových prohlížečích.

Dále jsem napsal knihovnu pro komunikaci s hlavním řídicích serverem. Ta při připojení načte veškerá zařízení ze serveru, jejich šablony, akce a porty. Zároveň udržuje spojení se serverem. Pokud je ztraceno, pokouší se o opětovné připojení. Po připojení jsou všechna data načtena znovu. Programátor má pak k dispozici snadno použitelné nástroje pro iteraci těchto dat a pro ovládání zařízení. V následujícím kódu je ukázka použití knihovny.

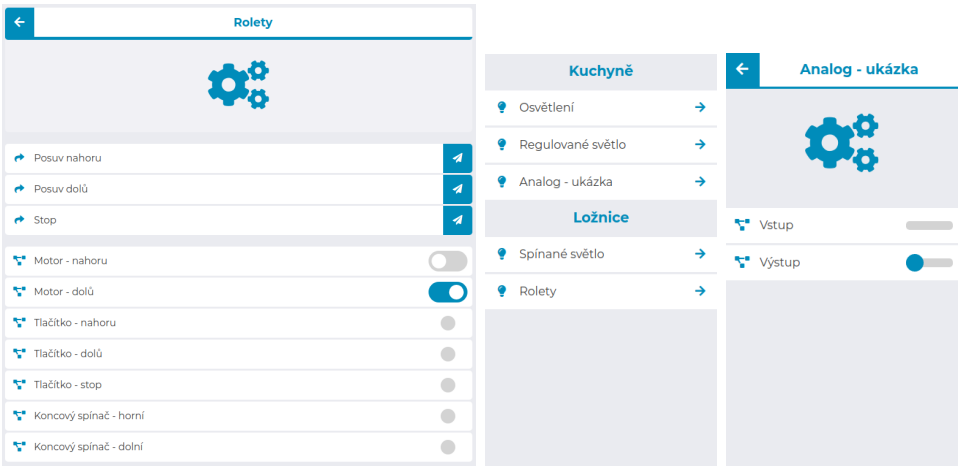
[5] Hvězdička znamená žádný nebo více prvků, znaky v apostrofech jsou konečné znaky nebo regulérní výrazy. Hranaté závorky značí nepovinný výraz.

```
(function() {
  const HOST_ADDR = "192.168.1.117";
  const HOST_PORT = 8083;
  const APP_UUID = "d85ee646-73fd-4aa0-b107-3f815c405ba7";

  let client = new VejminekClient(HOST_ADDR, HOST_PORT, APP_UUID
    , true);

  client.on("load", function() {
    client.eachRoom(function(floor, label, numDevices) {
      if(numDevices == 0)
        return;

      let roomLabel = label;
      if(label === null)
        roomLabel = "Nezařazeno";

      //... generate an HTML element for the room here

      client.eachDeviceByRoom(label, function(id, device) {
        // ... generate an HTML element for the device here
      });
    });

    setInterval(function() {
      client.eachPortValue(function(deviceId, portId, value) {
        //... update port values here
      });
    }, 100);

    // call an example action after 5 seconds
    setTimeout(function() {
      const deviceId = 1;
      client.callDeviceAction(
        deviceId, "impulse", {duration: 300});
    }, 5000);
  });

  client.connect();
})();
```

Ukázkové uživatelské rozhraní jsem pro jednoduchost napsal tak, aby ukázalo všechny možnosti, které lze na zařízení vykonat. Proto se u každého zařízení zobrazí všechny akce i porty (včetně vstupních). V případě potřeby by pak nemělo být problémem přizpůsobit zobrazení dle šablony zařízení.

Stejně jako administrační rozhraní, je i toto responzivní. Přesto je však určeno primárně pro mobilní zařízení, na kterých použití dává větší smysl. Na

obrázku 4.5 je ukázka na desktopu (vlevo) a na mobilním zařízení (vpravo). Díky použití WebSocketů má aplikace velmi rychlou, téměř nepostřehnutelnou, odezvu.



Obrázek 4.5: Ukázka uživatelského rozhraní

Kapitola 5

Bezpečnost a spolehlivost

Systém pro automatizaci je tedy hotov. Je na čase jej zprovoznit v nějaké domácnosti. Jednou z prvních otázek, které lidi napadne, když přijde řeč na chytré domácnosti, je, zda se dá takovým systémům věřit? Zda jsou spolehlivé? „Nestane se, že se dům obrátí proti mně?“ Nejprve bych rád zdůraznil, že žádný systém není dokonale spolehlivý a jeho zabezpečení může být prolomeno. Záleží pouze na tom, kolik úsilí a času útočník obětuje.

Na spolehlivost a úroveň bezpečnosti se lze dívat z několika úrovní. Začnu hardwarem. Řídící jednotka má tři možnosti komunikace s vnějším světem - UART, UART s fyzickou vrstvou CANu a CAN 2.0B. Standardní sériová komunikace přes UART je nejméně spolehlivá, protože přenos probíhá od vysílajícího zařízení k přijímajícímu pouze po jednom vodiči. Zároveň se neprovádí žádný kontrolní součet, bit stuffing, ani jiná protichybová opatření. Zařazením CAN transceiveru se vylepší odolnost proti souhlasnému rušení, protože pro přenos dat využívá diferenciální pár. Protokol CAN ovšem zahrnuje jak přenos po diferenciálním páru, tak bit stuffing a CRC součet. Pokud je přijatá zpráva poškozená, je navíc vyžádáno opětovné zaslání. To dělá z CANu velmi spolehlivý nástroj pro komunikaci. Použitý transceiver mimo jiné zvládne přenos i po jednom vodiči (pokud jsou propojené země komunikujících uzlů) v případě přerušení jednoho vodiče diferenciálního páru. Zabezpečení komunikace v tomto případě nemá význam, protože útočník nemá možnost se na sběrnici napojit (za předpokladu, že je kabeláž uzavřená v domácnosti). Pokud by ovšem nastalo neočekávané přerušení komunikace s řídicím serverem (například selhání paměťového média v počítači nebo poškození datového vedení), je řídicí jednotka vybavena přepínači pro manuální ovládání výstupů. Hardwarové řešení tedy lze prohlásit za dostatečně spolehlivé a bezpečné.

Dalším citlivým bodem je stroj, na kterém běží řídicí server. To může být libovolný počítač s operačním systémem Linux. Použití samotného operačního systému může způsobit komplikace, protože kromě řídicího serveru běží na pozadí i jiné programy, které musí operační systém obsluhovat. Pokud dojde na časově kritickou situaci, může se stát, že řídicí server nezareaguje okamžitě. V případě automatizované domácnosti k takovým situacím obvykle nedochází. Ty jsou běžné spíše v průmyslových aplikacích. Jestliže je potřeba zpracovat časově kritické situace, je nutné toto chování implementovat již na úrovni řídicí

jednotky. Pokud je potřeba operační systém, lze použít nějaký RTOS.[1] Řídicí server je provozován jako systémová služba. V případě, že by došlo k chybě za běhu, která by vedla k pádu programu, je služba nastavena tak, aby se okamžitě spustila znovu. K zjištění zdroje chyby je pak možné použít nástroje operačního systému (například *journalctl*) a logové soubory, produkované samotným serverem. Co se týče bezpečnosti, představuje bezpečnostní riziko otevřený SSH a SFTP server, skrze který lze stroj ovládat po síti. Pokud požadujeme silné zabezpečení, je lepší tento server neotevírat a stroj ovládat pouze přímo fyzicky. Možným řešením je také stroj provozovat pouze na lokální síti bez přístupu k internetu.

Řídicí server v libovolné konfiguraci vždy navazuje spojení s databázovým serverem MariaDB. Ten je přímo určen k tomu, aby poskytnul spolehlivé a rychlé úložiště dat. Může ale v nesprávném nastavení představovat bezpečnostní riziko. Obvykle tento server běží na stejném stroji, jako řídicí server. Pokud by chtěl útočník odposlouchávat nebo ovlivnit tuto komunikaci, musel by mít přístup na samotný stroj, na kterém tyto servery běží. Nejlepším řešením je, aby měl k databázi přístup pouze jediný uživatel se silným heslem, který je navíc omezen přístupem pouze z lokální IP adresy (127.0.0.1).

Nejslabším článkem jsou administrace a uživatelské rozhraní. Obě rozhraní používají k přenosu dat protokol TCP/IP, který zajišťuje spolehlivý přenos. Představují ale bezpečností riziko. Protože se k serveru obvykle přistupuje přes IP adresu, není možné použít zabezpečený HTTPS protokol, který vyžaduje pro platný certifikát doménu. Pokud vynecháme použití veřejné IP adresy (která také představuje bezpečnostní riziko) v kombinaci s doménou, nezbývá nic jiného, než použít nezabezpečený HTTP protokol. Administrace je chráněna autorizačním klíčem, ale ten lze na nešifrovaném protokolu snadno odposlechnout. Co se týče dalších rizik, je rozhraní chráněno proti *SQL Injection* a podobným útokům. Uživatelské rozhraní komunikuje přes WebSockety, které mají bohužel stejný problém jako HTTPS. Jinak se musí, stejně jako každá jiná aplikace navazující spojení s řídicím serverem, pro komunikaci nejprve autorizovat svým UUID.

[1] RTOS = Real Time Operating System

Kapitola 6

Vylepšení systému nad rámec této práce

I přesto, že systém, který jsem v rámci této práce vytvořil, je již poměrně robustní a univerzální, nabízí se celý seznam vylepšení, na které nebyl jednoduše dostatek času.

6.1 Hardware

V prvních dnech testování řídicí jednotky jsem zaznamenal hned několik nedostatků, které bych v této části rád uvedl. Předně je to napájení řídicí desky, které je realizováno dvěma napěťovými stabilizátory. Při vyšším odběru se tyto stabilizátory poměrně značně zahřívají kvůli vysokému úbytku napětí (při napájení z 12V zdroje je to 7 V a 9,7 V). Proto by bylo vhodné použít pro stabilizaci na napětí 5 V spínaný zdroj. Ze spínaných 5 V by pak stačilo na 3,3 V použít už jen jeden lineární stabilizátor. Také by bylo dobré výstup 5 V a 3,3 V vyvést na konektor, aby je bylo možné použít pro analogové aplikace. To mě přivedlo na myšlenku expanderů. Na řídící desce by mohl být dvouřadý konektor s vyvedenými sběrnicemi USART, SPI a I^2C a napájením. Takovýto konektor by zároveň sloužil (spolu s otvory pro uchycení řídicí jednotky, které nakonec nebyly použity) pro fyzické upevnění expanderu na řídicí jednotku. Expandery by mohly zajišťovat aplikačně specifické požadavky, jako RFID nebo NFC čtečky, přesnější analogová měření, zpracování signálů z pokročilých senzorů, jako je Grid Eye a podobně. Samozřejmě pro tyto potřeby lze navrhnout jiné řídicí jednotky, ale možnost rozšíření řídicí jednotky může přijít vhod. Nakonec bych ještě oddělil zvlášť napájení pro řídicí jednotku a zvlášť napájení pro spínání, která jsou na současné řídicí jednotce propojena. Spínání totiž může značně zarušit napájení řídicí jednotky, což může mít za následek nežádoucí reset mikrokontroléru. Mikrokontrolér sám je jinak chráněn proti podpětí interními obvody (Brown Out Reset). Toto chování jsem při testování nepozoroval, na druhou stranu jsem obvod netestoval spínáním velké indukční zátěže.

6.2 Firmware

Jak jsem již zmínil v kapitole 3, řídicí jednotka má také USB konektor typu C. Kromě toho, že by bylo možné tento konektor použít pro napájení (není to

ale potřeba), jeho hlavní funkcí mělo být umožnění snadné konfigurace ze strany počítače formou konfiguračních souborů. Původní návrh dokonce místo USB zahrnoval slot pro SD kartu. Použití SD karty selhalo na požadavcích organizace SD Association, která má vysoké finanční požadavky na licenci pro komerční použití. Ze stejného důvodu selhalo použití USB jako *Mass Storage Device*, ke kterému je potřeba nějaký souborový systém. Jedinými souborovými systémy, které jsou podporované napříč operačními systémy, jsou (alespoň dle výsledků mého hledání) FAT32 a exFAT, oba vydané společností Microsoft. Jediným současným řešením, jak obejít licenční požadavky, je implementovat vlastní jednoduchý souborový systém, což sebou nese řadu problémů. Toto téma ale nechám stranou.

USB rozhraní může posloužit ještě jedné věci. Jelikož je knihovna pro jazyk LUA, kterou jsem použil pro implementaci zásuvných modulů na řídicím serveru, prakticky platformně nezávislá, a je psaná v jazyce C, mělo by být možné ji použít i přímo na mikrokontroléru. I přesto, že je LUA interpretovaný jazyk, není zdrojový kód interpretován přímo. Nejprve je přeložen do strojového kódu. To z něj dělá velmi výkonný nástroj. Jedinou překážkou může být dynamická alokace paměti, která je obecně náročná na hardwarové prostředky, a která na mikrokontroléru může být pomalá a neefektivní. Jazyk LUA by ale mohl velice dobře posloužit pro uskutečnění autonomní logiky řídicí jednotky. Navíc by celé API mohlo být identické s API pro zásuvné moduly na řídicím serveru. Pro nahrání zdrojového kódu by se použilo právě USB rozhraní, které by ani nemuselo být typu *Mass Storage Device*.

6.3 Řídicí server

Zde je největší prostor pro možná vylepšení. Server ve své podobě již má nástroj pro časování úloh. Chybí zde však uživatelsky přívětivý mechanismus plánování podle reálného času. Tedy stejně jako je implementována systém služba *Plánovač* (*Scheduler*), bylo by na místě implementovat *Real Time Scheduler*, který by v daném okamžiku mohl buď spustit skript v jazyce LUA, nebo změnit hodnoty portů, a nebo zavolat akci na zařízení (tedy něco jako *Cron* v operačním systému). Tato služba by se nastavila skrze administrační rozhraní.

Dále chybí zabezpečovací systém. Ten lze implementovat pomocí zásuvného modulu, ale dost krkolomnou cestou. Zabezpečovací systém by měl být implementován již na straně serveru jako systémová služba. Tato myšlenka mě ale přivedla na celkové přepracování systémových služeb a doplnění API pro zásuvné moduly. Každý zásuvný modul by mohl ve svém deskriptivním souboru uvést závislosti na jiných modulech. Jiné moduly by mohly ostatním poskytnout nové systémové služby, jako je právě zabezpečovací systém. Tím by jádro zůstalo jednoduché a snadno by se jeho funkcionalita dala rozšířit v případě potřeby.

Co se týče API pro zásuvné moduly, chybí zde podpora několika věcí, které by v budoucnu mohly být potřeba.

- Protokol Allkol
- Samotné protokoly TCP/IP, TLS a WebSocket (resp. celé HTTP(S)/1.1), případně i UDP/IP.
- Konektor pro databáze MySQL/MariaDB, případně PostgreSQL, MongoDB a další
- Zařízení USB
- Nástroje pro hashování a šifrování
- Enkodér a dekodér pro formáty JSON a XML
- Neuronové sítě, strojové učení? Nebo alespoň nástroje pro lineární algebru a statistiku

6.4 Administrační rozhraní

Jak jsem již nastínil v předchozí části, systém by měl uživateli poskytnou plánovat úkony v reálném čase. Tyto úkony by se nastavovaly v administračním rozhraní. Každá položka by byla jednorázová nebo opakovaná. V případě jednorázové by byl nastavitelný čas a datum vykonání. V případě opakované pak dny v týdnu a čas. Dále by uživatel zvolil typ úkonu - tedy skript, zápis na port nebo volání akce na zařízení.

Dalším vylepšením, které se nabízí, je automatické načtení uživatelských LUA skriptů z databáze při spuštění serveru. Webové rozhraní by obsahovalo také editor a nějaký, alespoň jednoduchý, validátor. Skripty by sloužily převážně k propojení zařízení pomocí událostí a akcí. To mě přivedlo na myšlenku implementace vizuálního programovacího jazyka. Zařízení by se propojovala formou „puzzlí.“ Takovýto způsob umožňuje upravit chování serveru nejen programátorovi, ale i pokročilému uživateli, který nemusí nutně umět programovat. V opačném případě je uživatel odkázán pouze na dostupná zařízení, které poskytují zásuvné moduly.

Nakonec by měl mít uživatel možnost nainstalovat nebo odinstalovat zásuvné moduly přímo v administračním rozhraní. V případě potřeby pak zablokovat jejich spouštění. To také otevírá otázku nějakého oficiálního repozitáře, odkud by měl uživatel zaručeno stahování pouze funkčních a bezpečných zásuvných modulů.

6.5 Uživatelské rozhraní

Uživatelské rozhraní jsem napsal tak, aby demonstrovalo možnosti komunikačního rozhraní se serverem. První věc, která mě napadla, je systém zásuvných modulů, stejně jako u řídicího serveru. Každý zásuvný modul by mohl definovat widgety a ovládací panel pro konkrétní šablony zařízení. Pokud by pro zařízení byl k dispozici speciální ovládací panel, byl by načten ten, v opačném případě by se načetlo univerzální rozhraní, jako je tomu nyní.

Pro uživatelovo pohodlí by pak měla být aplikace nastavitelná individuálně. Uživatel by na své domovské obrazovce mohl mít widgety a zařízení, která používá nejčastěji. Také by se zde mohla nacházet makra, která provedou více úkonů najednou. Zároveň by dostával notifikace o dění na serveru. Pokud by například došlo k vypnutí nějakého spotřebiče kvůli úspoře energie, uživatel by byl o tomto úkonu informován. Tuto funkcionalitu by bylo samozřejmě nutné implementovat i na straně řídicího serveru.

Nakonec se nabízí možnost zabalit celé rozhraní do nějaké nativní aplikace (v jazyce Java pro systém Android a v jazyce Swift/Objective-C pro iOS), aby mohla být načtena i v mobilních zařízeních bez použití webového serveru. Tím by odpadla i nutnost zadávat URL adresu rozhraní do prohlížeče.

Kapitola 7

Odhad rozpočtu

Nyní, když jsem rozebral jednotlivé části systému, mohu odhadnout, na kolik by vyšla jeho integrace při stavbě nového bydlení. Následující tabulka zahrnuje uvažované položky a jejich orientační ceny. Vypínače jsem vybral z kategorie levnějších výrobků.[1]

Položka	Cena
Řízení	
Řídicí počítač (Raspberry Pi)	2 000,-
Software a jeho vývoj	16 000,-
Řídicí jednotka	4 500,-
Elektroinstalace	
Relé na DIN lištu	400,-
Pohybový senzor PIR	400,-
Kabel UTP (1 metr)	10,-
Spínač - jednoduchý	80,-
Spínač - dvojitý	120,-

Tabulka 7.1: Položky uvažované při návrhu rozpočtu.

Pro odhad použiji rodinný dům 4+KK. Ten zahrnuje obývací pokoj s kuchyňským koutem, dva dětské pokoje, jednu ložnici, koupelnu, toaletu, chodbu, technickou místnost s rozvaděčem a předsíň. Do odhadu nezahrnuji rozvod nízkého napětí (230 V) a zásuvky. Zároveň nepočítám samotná světla, žaluzie a další prvky, jejichž cena se může výrazně měnit dle výběru. K vypínačům vede po kabelu UTP malé napětí (12 V) a od nich přímo do řídicích jednotek. V obývacím pokoji, kuchyňském koutu, ložnici a dětských pokojích jsou umístěny žaluzie. V každé místnosti je pak dále osvětlení (jednoduché nebo dvojité). V obývacím pokoji je umístěno stmívané RGBW světlo pro odpočinek. V chodbě jsou nízko nad zemí umístěna noční navigační světla. Na toaletě je také odvětrávání. Následující tabulka zahrnuje výdaje rozdělené podle místností.

[1] Orientační ceny vypínačů odečteny z webových stránek společnosti ABB uvedené v kapitole 2.

Položka	Množství	Cena
Obývací pokoj		
Vypínač - hlavní světlo	1 ks	120,-
Vypínač - žaluzie	1 ks	120,-
Relé - hlavní světlo	2 ks	800,-
Relé - žaluzie	2 ks	800,-
RGBW zesilovač	1 ks	500,-
Zdroj 12 V, 120 W	1 ks	400,-
Pohybový senzor	1 ks	400,-
KK, ložnice, dětské pokoje		
Vypínač - světlo	1 ks	120,-
Vypínač - žaluzie	1 ks	120,-
Relé - žaluzie	2 ks	800,-
Relé - světlo	2 ks	800,-
Pohybový senzor	1 ks	400,-
Koupelna		
Vypínač - hlavní světlo	1 ks	80,-
Ultrazvukový senzor u umyvadla	1 ks	1000,-
Relé - hlavní světlo	1 ks	400,-
Relé - osvětlení nad umyvadlem	1 ks	400,-
Pohybový senzor	1 ks	400,-
Toaleta		
Vypínač - světlo	1 ks	80,-
Vypínač - ventilace	1 ks	80,-
Relé - světlo	1 ks	400,-
Relé - ventilace	1 ks	400,-
Pohybový senzor	1 ks	400,-
Chodba		
Vypínač - hlavní světlo	2 ks	160,-
Relé - hlavní světlo	1 ks	400,-
Relé - noční navigační světlo	1 ks	400,-
Pohybový senzor	2 ks	800,-
Předsíň, technická místnost		
Vypínač - světlo	1 ks	80,-
Relé - světlo	1 ks	400,-
Pohybový senzor	1 ks	400,-
Nezařazeno		
Kabeláž	400 m	4 000,-

Tabulka 7.2: Tabulka rozpočtu na rodinný dům 4+KK.

Do místností, jako je obývací pokoj, jsem také zahrnul pohybové senzory, protože mohou být použity pro případné zabezpečení, nejen pro rozsvěcení světel. Po sečtení položek (včetně řízení) celková suma vychází na 53760 Kč.

Nutno dodat, že se započítáním lidské práce, ceny za žaluzie, lepších vypínačů, zásuvek a dalších prvků celková suma značně vzroste.

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a implementovat systém, který by programátorovi usnadnil automatizaci domácnosti, a to včetně hardwaru. Systém sám o sobě neměl implementovat chytrost, ale měl vývojáři poskytnout nástroje k tomu, aby bylo možné chytrost pro danou situaci snadno implementovat.

Ná základě tohoto zadání jsem navrhnul řídicí jednotku v programu EasyEDA a nechal ji vyrobit zahraniční společností JLCPCB. Pro řídicí jednotku jsem zvolil mikrokontrolér STM32F303CC, který obsahuje periferie pro UART, CAN, I^2C, SPI a USB. Dále obsahuje dostačný počet portů a vhodné periferie pro realizaci deseti digitálních a analogových vstupů a deseti digitálních a PWM modulovaných výstupů. Jednotka má tedy více možností pro komunikaci s okolím a zároveň je vybavena všemi potřebnými prostředky pro řízení automatické domácnosti.

Softwarovou část jsem rozdělil do tří programů - hlavního řídicího serveru, administračního rozhraní a uživatelského rozhraní. Jako platformu jsem zvolil operační systém Linux (libovolné distribuce), který poskytuje vhodné nástroje pro provozování řídicího serveru (například systemd) a webového serveru Apache, na kterém může běžet jak administrátorské rozhraní, tak uživatelské rozhraní.

Jádro řídicího serveru jsem napsal v jazyce C++, který je kompilován do strojového kódu a paměť programu spravuje přímo programátor. To umožňuje optimalizovaný běh programu. Navíc jsou v jazycích C a C++ dostupné knihovny pro integraci a spouštění skriptů jazyka LUA. Na jazyku LUA jsem postavil systém zásuvných modulů, které umožňují snadno rozšířit funkcionalitu řídicího serveru o podporu nového hardwaru i řídicí logiku.

Administrační a uživatelské rozhraní jsem pro jednoduchost napsal jako webové aplikace. To nejen velmi usnadnilo vývoj, ale zároveň jsou aplikace maximálně multiplatformní. Uživatelské rozhraní je navíc napsáno pouze v JavaScriptu a proto může být zabaleno do nativní aplikace pro mobilní telefony nebo tablety. Jako problémové se ukázalo zabezpečení těchto aplikací pomocí šifrovaných protokolů HTTPS a WSS postavených na běžně užívaném protokolu TLS. Ten pro bezpečnou komunikaci vyžaduje certifikát spjatý s doménou. Jestliže aplikace běží na lokální síti, musí být použity protokoly HTTP a WS. To v případě, že by se útočník připojil k lokální síti, způsobuje bezpečnostní riziko.

Systém jsem zprovoznil na jednodeskovém počítači Raspberry PI s operačním systémem *Raspberry Pi OS*, postaveném na Linuxové distribuci *Debian*. Ukázky zdrojového kódu, které se nachází v této práci, jsem na něm odzkoušel a shledal funkčními.

Literatura

[1] STMicroelectronics. *STM32F303xB STM32F303xC.* ©2018. Dostupné online: https://www.st.com/resource/en/datasheet/stm32f303cc.pdf

[2] STMicroelectronics. *RM0316, Rev 7.* ©2017. Dostupné online: https://www.st.com/resource/en/reference_manual/dm00043574-stm32f303xb-c-d-e-stm32f303x6-8-stm32f328x8-stm32f358xc-stm32f398xe-advanced-arm-based-mcus-stmicroelectronics.pdf

[3] Infineon Technologies AG. *BSP762T, V1.1, 2007-05-29.* Dostupné online: https://www.infineon.com/dgdl/Infineon-BSP762T-DS-v01_01-EN.pdf?fileId=5546d4625a888733015aadf8e1b54c45

[4] NXP Semiconductors. *TJA1055, Rev. 5 — 6 December 2013.* Dostupné online: https://www.nxp.com/docs/en/data-sheet/TJA1055.pdf

[5] NXP Semiconductors. *PCF8574; PCF8574A, Rev. 5 — 27 May 2013.* Dostupné online: https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf

[6] Freescale Semiconductor, Inc. *Bosch Controller Area Network (CAN), Version 2.0, Rev. 3.* ©1998. Dostupné online: https://www.nxp.com/docs/en/reference-manual/BCANPSV2.pdf