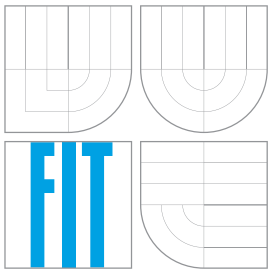
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



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

SYSTÉM PRO MONITOROVÁNÍ DOMÁCNOSTI A AUTOMATIZACI CENTRÁLNÍHO VYTÁPĚNÍ

SYSTEM FOR HOUSEHOLD MONITORING AND AUTOMATION OF CENTRAL HEATING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ SVOBODA
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÁCLAV ŠIMEK
SUPERVISOR

BRNO 2016

Zadání bakalářské práce/18895/2015/xsvobo0l

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Svoboda Tomáš**
Obor: Informační technologie
Téma: **Systém pro monitorování domácnosti a automatizaci centrálního vytápění**
System for Household Monitoring and Automation of Central Heating
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s již existujícími systémy pro automatizaci domácnosti, především pak s těmi, které jsou určeny k řízení vytápění.
2. Podrobně se zabývejte existujícími průmyslovými komunikačními sběrnicemi, přičemž hlavní pozornost věnujte sběrnici 1-wire.
3. Vytvořte návrh architektury systému, který bude plnit úkoly monitorování a automatizace domácnosti. Systém bude obsahovat senzorové moduly, akční členy a centrální řídící jednotku na bázi platformy Raspberry Pi.
4. V návrhovém prostředí připravte realizaci desek plošných spojů pro jednotlivé části navrhovaného systému.
5. Implementujte obslužný firmware pro jednotlivé komponenty systému (senzorové moduly, řídicí jednotka) a webovou aplikaci pro plánování událostí a ovládání automatizačního systému.
6. Vhodným způsobem demonstrujte funkčnost navrženého řešení a pokuste se navrhnout případná vylepšení či rozšíření.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

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: **Šimek Václav, 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
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Tato práce se řadí do kategorie vestavěných systémů. V rámci práce bylo navrženo a vyrobeno několik hardwarových modulů určených k automatizačním úkonům s využitím sběrnice 1-Wire a počítače Raspberry Pi. Byla navrhnuta a implementována rozšiřitelná aplikace pro řízení automatizace v jazyce C++, která je spuštěna po celou dobu běhu počítače. V rámci rozšíření byly implementovány ovladače pro ovládání vytápění v jednotlivých místnostech, řízení kotle a ovládání relé, kterým lze spínat běžné spotřebiče. Dále vznikla webová aplikace, která uživateli umožňuje přehlednou cestou monitorovat stav automatizačních činností a ovlivňovat jejich průběhy pomocí nastavení a plánování programů. Vytvořený systém lze nainstalovat do cílového objektu a s jeho pomocí ovládat centrální vytápění nebo spínat spotřebiče. Hlavní výhodou systému je jeho rozšiřitelnost, lze přidávat nové automatizační činnosti, zařízení nebo používat více zařízení připojených k počítači přes různé technologie.

Abstract

In this work, which belongs to the field of embedded systems, several hardware modules and central control unit for home automation purposes were designed and subsequently implemented using 1-Wire bus and tiny computer Raspberry Pi. Furthermore, an extensive application for management of automation activities in a given system was created in C++ programming language. This piece of software is running continuously as a service from computer boot time until its shut down. There were implemented several functionalities that enable controlling the general purpose relay to switch home appliances and to operate central heating automation features, which include operating of electronic actuators on the radiator and boiler. In addition, web-based application allows direct monitoring of automation activities and provides necessary means to plan the desired actions of the available units within the automation system. Resulting system could be installed on a target site and monitor temperature, automate central heating and switch basic home appliances using relay. Main advantage of the created system can be recognized in scalability where new automation tasks or devices could be added.

Klíčová slova

Domácí automatizace, monitorování domácnosti, řízení centrálního vytápění, Raspberry Pi, 1-Wire, C++

Keywords

Home Automation, Household Monitoring, Automation of Central Heating, Raspberry Pi, 1-Wire, C++

Citace

SVOBODA, Tomáš. *Systém pro monitorování domácnosti a automatizaci centrálního vytápění*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Šimek Václav.

Systém pro monitorování domácnosti a automatizaci centrálního vytápění

Prohlášení

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

.....
Tomáš Svoboda
18. 5. 2016

Poděkování

Rád bych poděkoval vedoucímu práce, panu Ing. Václavu Šimkovi, za odbornou pomoc při návrhu a výrobě hardwarových modulů, užitečné rady a připomínky k tvorbě této práce a její vedení. Dále také za zprostředkování zapůjčení servopohonů pro ovládání radiátorů, které byly použity v rámci předvedení práce. Tímto bych také rád poděkoval firmě Honeywell, která servopohony ochotně zapůjčila, konkrétně pak panu Matyášovi, který zapůjčení zprostředkoval. V neposlední řadě patří dík mé rodině za podporu při tvorbě díla a pomoc při jeho závěrečné korekci.

Obsah

Kapitola 1

Úvod

Automatizace je trendem současné doby a z velkých průmyslových aplikací se postupně rozšiřuje do domácností. Právě v domácnostech můžeme nalézt spoustu činností, které lze automatizovat, čímž je možné uživateli ušetřit náklady, čas a zvýšit jeho komfort. Domácí automatizační systémy vytváří tzv. inteligentní dům, ve kterém jsou určité činnosti automaticky ovládány na základě naplánovaných hodnot a údajů ze senzorů. Příkladem těchto činností může být ovládání vytápění, osvětlení, předokenních rolet nebo ovládání domácích spotřebičů.

Cílem práce je vytvořit funkční systém, který bude monitorovat domácnost (např. teplotu) a automatizovat centrální vytápění. Automatizací vytápění je myšleno ovládání radiátorů nezávisle v jednotlivých místnostech podle přednastavených programů (tzv. zónová regulace), ovládání kotle nebo oběhového čerpadla na základě požadavků na vytápění a možnost jednoduše měnit parametry vytápění. Systém bude rozšiřitelný i pro další automatizační činnosti, jednou z těchto činností může být spínání spotřebičů, které bude v práci také předvedeno.

Tato práce se nejprve v kapitole 2 zabývá studií již existujících řešení, která bude sloužit jako odrazový bod při návrhu systému. V navazující kapitole 3 je rozebrána sběrnice 1-Wire, která bude použita pro komunikaci řídicí jednotky se zařízeními. Kapitola 4 rozebírá jednotlivé části systému a poskytuje informace o počítači, který bude použit jako řídicí jednotka. Následující kapitola 5 se zabývá návrhem hardwarových modulů a jejich propojením s řídicí jednotkou. Návrh a implementace řídicí služby, která bude vykonávat automatizační úkony, je rozebrán v kapitole 6. Třetí částí systému je webová ovládací aplikace. Návrh a implementace této části jsou rozebrány v kapitole 7. Poslední kapitola si klade za cíl shrnout požadavky, které musí být dodrženy, aby mohl být systém rozšířen. V přílohách lze nalézt fotografie vyrobených hardwarových modulů, ER diagramy návrhu databáze, diagramy tříd řídicí služby, snímky obrazovek z ovládací aplikace a manuál pro instalaci projektu na použitý počítač.

Kapitola 2

Studium existujících řešení

Domácí automatizace je dnes velmi žádaná a s příchodem IoT (Internet of Things), který přináší koncept propojení vestavných systémů se zařízeními pomocí existující sítě, tento trend nabývá na objemu. V této kapitole bude rozebráno několik existujících řešení, dostupných na tuzemském trhu, které poskytnou odrazový bod pro návrh systému, který je náplní této práce.

2.1 INELS – smart home solutions

V České republice existuje mnoho firem, které se zabývají automatizací domácnosti či její vybrané části. Společnost Elko EP s.r.o. nabízí řešení iNELS smart home solutions [1]. Jedná se o ucelený systém pro řízení domácnosti postavený na bezdrátové a sběrnicové elektroinstalaci. Dle výrobce lze oba přístupy navíc kombinovat, což přináší vyšší flexibilitu a použitelnost řešení. Systémem lze ovládat regulaci vytápění, osvětlení, rolety, dveře garáže či spínat spotřebiče. Většina modulů je v provedení pro instalaci do rozvaděče na DIN lištu EN60175, ostatní jsou určeny do instalačních krabic přímo v místě použití – jedná se hlavně o aktuátory a ovládací prvky, které se umísťují na stěnu.

Na sběrnici je pravděpodobně použit standard RS485, jedná se o sériovou dvouvodičovou polo-duplexní sběrnici, která přenáší data i napájení. Napájecí napětí systému je 27 V DC. Maximální proud odebíraný jednou větví sběrnice je 1000 mA, což je součet proudového odběru všech zařízení na větvi. Zvýšení počtu větví lze provést jednotkou externí master MI3-02M. Počet zařízení na větvi je omezen použitým standardem sběrnice na 32 zařízení. Sestava iNELS se skládá ze systémových jednotek, kde se jedná především o zdroj, oddělovače sběrnice či externí master. Dále pak z řídicích jednotek, aktorů, vstupních jednotek, převodníků a nástěnných ovladačů.

Centrální jednotka řídí celý systém a zároveň poskytuje webové rozhraní pro jeho správu. Právě za tímto účelem jednotka nabízí ethernetový port pro připojení do existující LAN sítě. Další částí systému jsou vstupní jednotky. Jedná se hlavně o jednotky binárních vstupů, na které lze připojit až 14 zařízení s bezpotenciálovým kontaktem, dále pak jednotky teplotních vstupů. Na jednu jednotku teplotních vstupů lze připojit až 6 externích senzorů – dvou nebo tří vodičových termistorů. Rozsah měřené teploty je dle použitého senzoru, přesnost převodu je 15 bitů.

Aktory lze rozdělit na spínací a stmívací. Spínací aktory jsou umístěny v modulech pro spínání 2, 4, 6 nebo 12 kanálů. Maximální zatížitelnost se liší dle použitého modulu, jedná se o hodnoty 16 A/230 V AC nebo 8 A/230 V AC. Spínacím prvkem je relé. Spe-

ciálním spínacím modulem je roletový aktor, který umožňuje spínat pohony rolet, žaluzií, garážových vrat či jiných elektrických pohonů, které jsou řízeny ve dvou směrech a mají zabudovaný koncový spínač. Stmívací aktory slouží k ovládání intenzity jasu osvětlení. Typ světelného zdroje lze nastavit pomocí potenciometru. V nabídce jsou moduly obsahující 2, 3 a 6 kanálů.

Další skupinou zařízení jsou převodníky. Ty lze rozdělit do dvou podskupin na A/D a D/A. Druhá podskupina převodníků je určena pro generování analogových signálů, generované napětí se pohybuje v rozsahu 0 – 10 V. Těmito převodníky lze ovládat termostatické hlavice, servopohony, stmívací aktory pro LED a jiné. A/D převodník je vybaven šesti vstupy, které jsou převáděny s rozlišením 14 bitů. Měřící rozsahy jsou 0 – 10 V a 0 – 2 V. Vstupy je možno nakonfigurovat jako proudové, napěťové nebo teplotní.

Poslední částí systému jsou nástěnné ovladače, které umožňují přímou interakci se systémem. Jedná se o dotykové obrazovky, skleněné dotykové ovladače, dvou a čtyř tlačítkové ovladače, digitální pokojové termoregulátory či čtečky karet.

Představený systém lze hodnotit jako velmi vyspělý, nabízí spoustu modulů, které lze vhodným zapojením použít pro automatizaci mnoha prvků v domácnosti. Systém je navíc rozšiřitelný a poskytuje převodníky pro komunikaci s jinými sběrnicemi. V nabídce jsou aplikace pro jeho ovládání z chytrých telefonů, tabletů nebo počítače přes domácí LAN síť. V rámci mobilních platforem jsou podporovány operační systémy Android a iOS.

2.2 Jablotron 100

Další významnou společností na českém trhu, která se věnuje automatizaci domácnosti je společnost Jablotron, která nabízí systém Jablotron 100 [4]. Automatizační řešení je opět založeno na kombinaci sběrnicového a bezdrátového provedení. Společnost nabízí širokou paletu modulů, které lze propojit. Jedná se hlavně o ovládací prvky, detektory a vstupní moduly. Tento systém je zaměřen více na monitorování objektu a spínání zařízení než předchozí systém, který se snaží o komplexní automatizaci včetně vytápění. Většina poskytovaných modulů je určena k montáži do instalačních krabic, ostatní na DIN lištu.

Srdcem celého systému je ústředna, která nabízí bezdrátový vysílač a budič sběrnice. Použitá sběrnice je pravděpodobně proprietární, ke svému provozu vyžaduje čtyři vodiče, z nichž jeden slouží pro napájení a druhý vodič je zemnící. Další dva vodiče budou pravděpodobně datové. Z toho vyplývá, že by se mohlo jednat o plně duplexní přenos dat v případě využití každého vodiče pro jeden směr přenosu nebo o poloduplexní přenos dat s využitím diferenciálního signálu. Ústředna dále nabízí vestavěné GSM/GPRS, díky kterým je možné po připojení rádiového modulu JA-110R komunikovat přes síť operátora s koncovými uživateli nebo střediskem. Výrobce udává možnost rozesílání SMS a hlasových reportů až osmi uživatelům. Ústředna je dále připojena do sítě LAN, skrz kterou s ní lze komunikovat. Nastavení ústředny se provádí prostřednictvím softwaru F-Link.

Systém nabízí několik druhů detektorů. Modul JA-118M umožňuje připojení až osmi magnetických detektorů, které slouží ke snímání stavu otevření/zavření oken či dveří. Magnetické detektory pro snímání polohy oken a dveří jsou k dispozici i v bezdrátovém provedení a jsou napájeny jednou baterií AA 1,5 V. Udávaná typická životnost baterie v zařízení je zhruba dva roky při maximálně 20 aktivacích denně. Dalším senzorem v systému je detektor teploty, který je opět dostupný ve sběrnicovém i bezdrátovém provedení. Posledním detektorem je záplavový senzor, který je k dispozici pouze ve sběrnicovém provedení. Cílem detektoru je indikovat zaplavení prostoru vodou.

Většina výstupních modulů se soustředí na spínání silových výstupů, které mohou sloužit například ke spínání osvětlení, ventilátorů či jiných spotřebičů. Tyto moduly poskytují vždy jeden kanál pro spínání a jsou dostupné ve sběrnicovém i bezdrátovém provedení. Dalším výstupním modulem je JA-118N, který poskytuje 8 programovatelných výstupů. Tyto výstupy lze v systému použít i k různým signalizačním účelům. Vývody jsou izolovány od sběrnice, izolace je podle obrázku modulu pravděpodobně provedena pomocí optočlenů. Speciálním výstupním modulem je sběrnicový modul pro obsluhu elektrického zámku JA-120N, který slouží k napájení a ovládání zámků a propouštěcích systémů. Pro zajištění počátečního proudového impulzu, potřebného k otevření elektromagnetického zámku, obsahuje modul tři dobíjecí akumulátory.

Součástí systému Jablotron 100 jsou i ovládací prvky, které umožňují bezdrátově nebo drátově pomocí sběrnice ovládat různá zařízení. V nabídce jsou nástěnná tlačítka bezdrátová i sběrnicová, která mohou sloužit k vyvolání tísňového poplachu nebo k ovládání jiných zařízení, podle nastavení systému. Specialitou jsou bezdrátové dálkové ovladače, kterými lze ovládat zajištění/odjištění systému, aktivaci poplachů či další zařízení, například garážové dveře. Dalším speciálním ovládacím prvkem je vysílač do vozidla JA-185J, který lze použít pro dálkové otevírání garážových dveří nebo k přenosu poplachu z auta do domácího zabezpečovacího systému.

Systém lze ovládat pomocí webového rozhraní My JABLOTRON. Dále pak výrobce nabízí monitorovací a ovládací aplikace pro chytré telefony a tablety, konkrétně pro platformy iOS, Android a Windows Phone.

Popsaný systém se zaměřuje na lehce odlišnou část automatizace než první popisovaný systém, přičemž se daleko více zaměřuje na bezpečnost. Rozdíl obou systémů je i v montáži, druhý popisovaný systém více využívá instalaci modulů do instalačních krabic, zatímco první popisovaný se zaměřuje na montáž do DIN lišty rozvaděče.

2.3 Etatherm

Poslední případovou studií je systém Etatherm od stejnojmenné společnosti zaměřený výhradně na automatizaci vytápění, na rozdíl od předchozích dvou popisovaných systémů. Regulační souprava se skládá z řídicí jednotky, elektronických hlavic, čidel a koncových modulů. Systémem lze řídit teplovodní i přímotopné elektrické otopné soustavy. Ovládaný objekt je rozdělen do zón. Pro zjednodušení může každá zóna představovat jednu místnost.

Ke komunikaci mezi zařízeními používá systém vlastní sběrnici Etatherm, která je tvořena dvěma vodiči, mezi nimiž je napětí zhruba 10 V a proud nepřesáhne 500 mA. Sběrnice je odolná proti zkratu. Adresace zařízení na sběrnici je řešena pomocí propojek na každém zařízení. Srdcem systému je řídicí jednotka, která v pravidelném čtyřminutovém intervalu vyhodnocuje data z čidel a podle nich řídí aktuátory. Tato jednotka vystupuje na sběrnici v roli master. Jedna jednotka může poskytnout šestnáct adres, přičemž na jednu adresu lze připojit maximálně tři hlavice nebo koncové moduly. Fyzicky z jednotky vychází dvě sběrnice (větve), každá po osmi adresách. Čidla teploty nemají svoji adresu, jsou připojena na modul hlavice nebo koncový modul. Pokud je potřeba adresovat více koncových zařízení lze propojit více řídicích jednotek s využitím sběrnice RS485. K počítači lze jednotku připojit pomocí RS232 sériového portu.

K ovládání radiátorových ventilů slouží elektronická hlavice. Ta se skládá z bezkomutátorového motorku a desky elektroniky s mikroprocesorem. Každá hlavice musí mít připojeno své čidlo teploty, které může být interní (přímo v hlavici) nebo externí v krabičce. Pro snímání teploty je použit termistor, převod odporu na teplotu probíhá vždy v mikropro-

cesoru hlavice, ke které je termistor připojen. K hlavici lze dále připojit spínací kontakty pro snímání polohy oken – otevřeno/zavřeno. Výhodou těchto hlavic je plynulé ovládání radiátorového ventilu, které oproti dvoupolohovým termohlavicím nabízí jemnější regulaci. Další výhodou je rychlost přestavení mezi polohami otevřeno/zavřeno, která trvá zhruba 5 vteřin na rozdíl od termoelektrického pohonu, kde přestavení trvá okolo čtyř minut. Drobnou nevýhodou rychlého přestavení je hluk vydávaný motorkem.

Pro ovládání přímotopné soustavy nebo kotle je použit koncový modul, který plní podobnou funkci jako elektronická hlavice, ovšem místo pohybu motorkem spíná relé. Opět se tedy jedná o desku s mikroprocesorem, kterou lze adresovat a ke které je nutné připojit čidlo teploty. Řešení pro spínání kotle TUV je založeno právě na tomto modulu, ovšem místo termistoru je připojen pevný odpor. Spínání kotle se pak provádí změnou plánované teploty uvnitř řídicí jednotky v závislosti na požadavku na natápění. [2]

Systém lze ovládat přímo z řídicí jednotky čtyřmi tlačítky, což je značně nepohodlné. Ke grafickému ovládání je dostupná aplikace Etatherm, ovšem pouze pro operační systém Microsoft Windows a přes sběrnici RS232 lze s řídicí jednotkou komunikovat pouze z jednoho počítače. Pro správu systému přes síť lze zakoupit k jednotce webový server, který síťovou komunikaci zprostředkuje. S webovým serverem lze systém ovládat z jakékoliv platformy přes webový prohlížeč. Grafická uživatelská prostředí obou ovládacích aplikací se zaměřují především na funkčnost na úkor vzhledu, na druhou stranu všechna požadovaná funkčnost je dostupná.

2.4 Motivace

Uvedené společnosti rozhodně nepokrývají celý dostupný trh, jedná se pouze o tři krátké případové studie domácích automatizačních systémů. Srdcem každého automatizačního systému je řídicí jednotka, která zajišťuje komunikaci se vstupními/výstupními moduly, řídí jejich činnost podle nastaveného programu a zprostředkovává rozhraní pro komunikaci s uživatelem, ve kterém nabízí podporu pro sledování monitorovaných hodnot a plánování budoucích akcí.

Ze sledovaných systémů je patrná závislost na použité technologii, ať už se jedná o bezdrátovou nebo sběrnicovou. V navrhovaném systému, který je předmětem této práce, by měla vzniknout taková abstrakce, aby bylo možné použít a zkombinovat více technologií dohromady a současně zachovat jednotný přístup k zařízením stejného účelu připojeným přes různé technologie.

Jedním z hlavních kritérií, které rozhoduje při pořizování automatizačního systému, je jeho cena. Navrhovaný systém by měl cílit spíše na nadšence v oblasti automatizace, kteří jsou schopni (s případnou pomocí odborníka) systém zapojit, případně rozšířit o požadovanou funkcionalitu. Řídicí jednotkou by měl být levný počítač, který bude nabízet dostatečný výkon pro účely domácí automatizace.

Dalším rozhodujícím kritériem je rozšiřitelnost systému jak po stránce funkční, tak i po stránce hardwarových modulů. Rozšíření o nové hardwarové moduly by mělo být dostupné díky nezávislosti na použité technologii. Rozšíření funkčnosti se týká řídicího softwaru hlavní jednotky systému. Ten by měl obsahovat mechanizmy pro jednoduché rozšíření funkcionality, bez zásahů do ostatních částí softwaru.

Důležitým faktorem každého automatizačního systému je jeho ovládací část, se kterou koncový uživatel přímo interaguje. Zde je sympatická myšlenka využití www služby pro ovládací aplikaci, jelikož při jejím vhodném zabezpečení a nastavení ji bude možné provozovat i v síti internet a mít k ní tak přístup i mimo domácí prostředí.

Kapitola 3

Sběrnice 1-Wire

Ačkoliv by výsledné řešení mělo být nezávislé na použité technologii, pro zprovoznění a ověření funkčnosti je třeba jednu technologii zvolit a implementovat její obsluhu. Jako vhodná možnost se jeví použití sběrnice 1-Wire z důvodu jednoduchosti jejího zapojení, množství zařízení připravených pro tuto sběrnici a její univerzálnosti. Důležitým aspektem je i možnost rozšíření operačního systému Linux o podporu této sběrnice a jejího protokolu.

3.1 Popis sběrnice

1-Wire je sériová sběrnice navržená firmou Dallas Semiconductor (dnes Maxim Integrated), které ke svému provozu stačí pouze dva vodiče – datový a zemnící. Zařízení na sběrnici pak získává napájecí napětí z datového vodiče v případě, že se zrovna nekomunikuje – v klidovém stavu je sběrnice ve stavu logické 1. Pokud na sběrnici probíhá komunikace, zařízení je napájeno z vnitřního kondenzátoru, do kterého byl náboj uložen v klidovém stavu. Pro spolehlivou komunikaci na delší vzdálenost je vhodné použít třetí vodič pro napájení.

Na sběrnici je možné připojit mnoho zařízení, jejich počet je omezen topologií sběrnice, elektrickými vlastnostmi a délkou vodiče. Každé zařízení má z výroby unikátní 64bitové identifikační číslo, kterým jej lze adresovat. První byte z tohoto čísla identifikuje typ zařízení (rodinu), poslední byte je kontrolní součet (CRC).

Z elektrického hlediska se jedná o sběrnici s otevřeným kolektorem. Ke své činnosti potřebuje jeden pull-up rezistor, který zdvihá napětí na sběrnici na vysokou úroveň a může tak poskytovat napětí zařízením, která jsou připojena pouze dvěma vodiči. Právě zapojení s otevřeným kolektorem umožňuje, aby mohl být datový signál sběrnice stažen k logické 0 zařízením typu master i zařízením typu slave. Zařízení na sběrnici podporují 3 V i 5 V logiku, záleží tedy na aplikaci, ve které je sběrnice použita a na její délce. Rychlost komunikace po sběrnici je přibližně 16,3 kbit/s. Díky této rychlosti lze provozovat spolehlivou komunikaci na vzdálenost několika desítek metrů. Většina zařízení podporuje i tzv. overdrive mód, ve kterém lze komunikovat až 10krát rychleji.

3.2 Využitelná zařízení pro automatizaci

Pro sběrnici 1-Wire je připraveno velké množství zařízení. Množina použitelných zařízení se rozšiřuje, přidáme-li do ní zařízení z rodiny iButton, která také využívají tuto sběrnici. Pro účely této práce bude použito pár základních zařízení, která budou blíže popsána.

DS18B20

DS18B20 je digitální senzor teploty s programovatelným rozlišením 9 – 12 bitů. Poskytovaná teplota je ve stupních Celsia, s přesností ±0.5°C v intervalu -10 °C – +85 °C, což je pro případ domácí automatizace plně postačující. Celkový rozsah senzoru je pak -55 °C – +125 °C. Rychlost převodu teploty se odvíjí od použitého rozlišení. Při použití nejvyššího rozlišení trvá převod teploty na číslicovou hodnotu 750 ms. Integrovaný obvod je dostupný jak v tranzistorovém pouzdře TO-92, tak v SMD pouzdrech SO8 a μSOP8. [6]

DS2408

DS2408 je zařízení, které poskytuje 8 nezávislých programovatelných vstupně-výstupních pinů s otevřeným kolektorem. Logika zařízení zaručuje, že všechny změny výstupů proběhnou bez chyby. Dohromady lze z výstupních pinů odebrat až 20 mA. Jedná se o velmi univerzální integrovaný obvod, který může být využit v mnoha aplikacích domácí automatizace (např. spínání relé, ovládání displeje s řadičem, ovládání klávesnice, ovládání jiných senzorů/aktuátorů). Integrovaný obvod je dostupný pouze v SMD pouzdře SO16. [7]

DS2406

DS2406 poskytuje podobně jako předchozí zařízení programovatelné vstupně-výstupní piny, v tomto případě ovšem pouze dva. Z prvního (PIO-A) lze odebírat až 50 mA, z druhého (PIO-B) pouze 8 mA. Dále nabízí 1 Kb programovatelné paměti, do které lze zapsat uživatelská data. Integrovaný obvod je dostupný v pouzdrech TO-92 a TSOC6. Nevýhodou prvního pouzdra je absence druhého výstupního pinu a absence napájecího vstupu (využívá je pouze parazitní napájení). [5]

3.3 Připojení sběrnice k Raspberry Pi

Popisovaná sběrnice spadá do kategorie sběrnic typu master – slave. Master zařízení se může ve sběrnici vyskytovat pouze jednou. Jeho hlavním úkolem je zahajování a řízení provozu na sběrnici. Z toho vyplývá, že řídicí jednotka systému by se měla chovat vůči zbytku sběrnice jako prvek typu master. Existuje více způsobů, jak připojit sběrnici k počítači Raspberry Pi, v této sekci se budeme zabývat výběrem vhodné varianty pro konkrétní aplikaci. Možnosti připojení jsou následující:

1. Použití usb adaptéru DS9490R
2. Využití GPIO pinu č. 4 počítače Raspberry Pi
3. Připojení přes sběrnici RS232 (DS2480B)
4. Připojení přes sběrnici I^2C (DS2482-100, DS2482-800)

Použití usb adaptéru je finančně nejnáročnější varianta a vzhledem k tomu, že Raspberry Pi nabízí spoustu jiných možností, jak připojovat zařízení, byla tato varianta zavržena hned na začátku.

Nejjednodušším způsobem, jak připojit 1-Wire zařízení k Raspberry Pi, je připojení datového vodiče na pin GPIO4. K práci se zařízením je potřeba doinstalovat kernel patch w1, ke komunikaci se využívá tzv. bit-bangingu, což je softwarové řízení stavu pinu. Hlavní

nevýhodou tohoto řešení je, že datový vodič je připojen přímo k pinu mikroprocesoru a při indukci nečekaného napětí hrozí jeho nenávratné poškození. Nevýhodou je též komunikace na hladině 3,3 V, která je na delší vzdálenost nedostatečná. Pro účely serióznější automatizace je tedy tato varianta nevhodná.

Připojení přes sběrnici RS232 je často využívaná varianta, ovšem Raspberry Pi má pouze jeden výstup UART a ten bude v této aplikaci využit jako sběrnice RS232 pro komunikaci s počítačem za účelem komunikačního rozhraní bez využití sítě.

Možnost připojení 1-Wire sběrnice skrz sběrnici I^2C se jeví jako výhodná hned z několika hledisek. Prvním je cena mostu přes sběrnice, která je oproti variantě s usb adaptérem zanedbatelná. K tomuto účelu lze použít dva typy obvodů v závislosti na počtu nezávislých 1-Wire sběrnic, které zpřístupňují. Integrovaný obvod DS2482-100 zpřístupňuje jednu 1-Wire sběrnici, DS2482-800 zpřístupňuje 8 sběrnic. Druhé hledisko je použití hardwarového časování v mostu, čímž se docílí odstínění procesoru počítače od časování na sběrnici. Další výhodou je jednoduchá výměna v případě poškození mostu.

3.4 Komunikace se zařízeními

Pro komunikaci na sběrnici skrz operační systém opět existuje několik variant. Tou nejpropracovanější je použití virtuálního souborového systému OWFS (One Wire File System), který podporuje všechna zařízení použitá v této práci. K zařízení se přistupuje skrz adresář s jeho unikátním identifikačním číslem. Obsah adresáře je vázán na typ zařízení. Soubory uvnitř adresáře umožňují se zařízením pracovat pomocí operací čtení/zápisu. Virtuální souborový systém zpřístupňující zařízení se navíc stará i o kešování hodnot a umí tak zamezit opakovaným požadavkům ve velmi krátkém čase. Výhodou tohoto řešení je jeho elegance za cenu mírné režie virtuálního souborového systému a také skutečnost, že nezáleží na způsobu připojení 1-Wire sběrnice k systému. Z toho plyne použitelnost všech výše zmiňovaných způsobů.

Kapitola 4

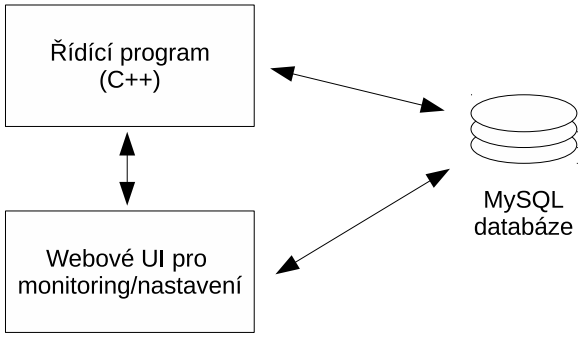
Architektura systému

Navrhovaný systém svou strukturou a účelem patří do skupiny vestavěných systémů. Dle zadání bude spustitelný na minipočítači Raspberry Pi pod operačním systémem Raspbian Lite, ovšem neměl by být problém přenést jej na jakoukoli platformu s Linuxovým operačním systémem.

Systém se skládá ze tří částí: perzistentního úložiště dat, řídicí služby a webové aplikace, sloužící pro ovládání systému. Schéma představených částí je zachyceno na obrázku 4.1. Perzistentním úložištěm dat byla pro tento projekt vybrána MySQL databáze, jelikož poskytuje dobrou podporu pro komunikaci s webovým ovládacím systémem (rozšíření mysqli v PHP) i s řídicí aplikací (Connector/C++[1]).

Další částí je řídicí aplikace. Jedná se o proces, který by měl běžet po celou dobu chodu počítače a provádět automatizační úkony. Do jisté míry se jedná o dynamicky rekonfigurovatelnou aplikaci, která mění svůj charakter podle stavu perzistentního úložiště. Právě v databázi je uložena veškerá konfigurace, která určuje, co se bude v aplikaci provádět.

Poslední částí je informační systém, který ke svému běhu vyžaduje webový server. Jedná se o jediné rozhraní, skrz které bude mít uživatel přístup k systému. Jeho hlavní náplní je poskytnout uživateli možnost vytvářet vlastní konfigurace, monitorovat aktuální stav systému a možnost zasahovat do vykonávání automatizačních činností. Komunikace s řídicí aplikací je realizována pomocí Unixových socketů.



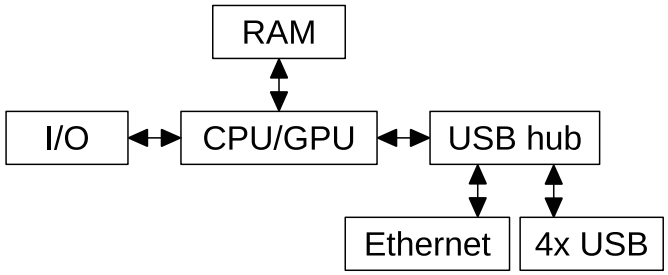
Obrázek 4.1: Návrh architektury celé aplikace

[1]Download MySQL Connector/C++ https://dev.mysql.com/downloads/connector/cpp/

4.1 Raspberry Pi

Raspberry Pi je jednočipový počítač velikosti kreditní karty, který nabízí výkon dnešních chytrých telefonů a mnoho periferních vstupů/výstupů. Počítač je od roku 2012 vyvíjen nadací Raspberry Pi Foundation. Hlavní výhodou počítače je jeho cena, která se pohybuje okolo 35$ za hlavní model. Od jeho vzniku vyšly tři verze a mnoho modelů počítače, přičemž pro tento projekt bude použita verze 2, model B. Raspberry Pi používá metodu SoC (System on Chip), což znamená, že všechny komponenty počítače, jako například procesor, paměť, grafické jádro, se nachází v jednom pouzdře. Od druhé verze počítače bylo z výkonnostních důvodů grafické jádro přesunuto do samostatného pouzdra, které se nachází na spodní straně desky.

Architektura počítače je oproti běžným počítačům značně zjednodušená a nenabízí vysokorychlostní rozhraní pro připojení pevného disku, jako primární úložiště dat je použita micro SD karta. Zjednodušený nákres architektury počítače je zobrazen na obrázku 4.2. Síťová karta je ve skutečnosti připojena přes rozhraní USB 2.0, od čehož se odvíjí i jeho rychlost. Na počítači je možné provozovat operační systém Linux, existuje mnoho distribucí, které jsou portovány pro použití na této platformě a liší se především svým zaměřením. Nejstarším a nejuniverzálnějším operačním systémem pro tento počítač je distribuce Raspbian. Jedná se o systém Debian upravený speciálně pro Raspberry Pi. [16]



Obrázek 4.2: Architektura počítače Raspberry Pi

Srdcem počítače, který bude použit pro tuto práci, je SoC Broadcom BCM 2836 z rodiny ARM Cortex-A7, který obsahuje čtyři procesorová jádra s taktem 900 MHz a 1 GB operační paměti. Dále obsahuje čtyři USB porty, z nichž bude jeden využit pro wifi adaptér, přes který bude připojen do LAN sítě. Jinou možností připojení počítače do místní sítě je pomocí kabelu s využitím síťové karty. Z nabízených periferií bude využita sběrnice I^2C pro připojení senzorů a aktuátorů skrz most přes sběrnice. Port UART bude využit pro komunikaci přes protokol RS232. Dále budou využity dva GPIO piny pro signalizaci stavu řídicí služby. Pro uložení dat bude využita paměťová karta o kapacitě 16 GB, která by měla být dostatečně naddimenzována pro uložení operačního systému i aplikačních dat. Vhodným operačním systémem pro účely automatizace je Raspbian ve verzi Lite. Tato verze obsahuje pouze jádro operačního systému bez grafického uživatelského prostředí a předinstalovaných aplikací.

Pro tento počítač bylo vytvořeno mnoho zajímavých výukových, ale i praktických aplikací, viz [9]. Jednou z nich je aplikace Open Z-Wave, která je určena k jednoduché domácí automatizaci na síti Z-Wave. Například pomocí serveru `lightscontrol` lze ovládat osvětlení v místnosti.

4.2 Použité abstrakce

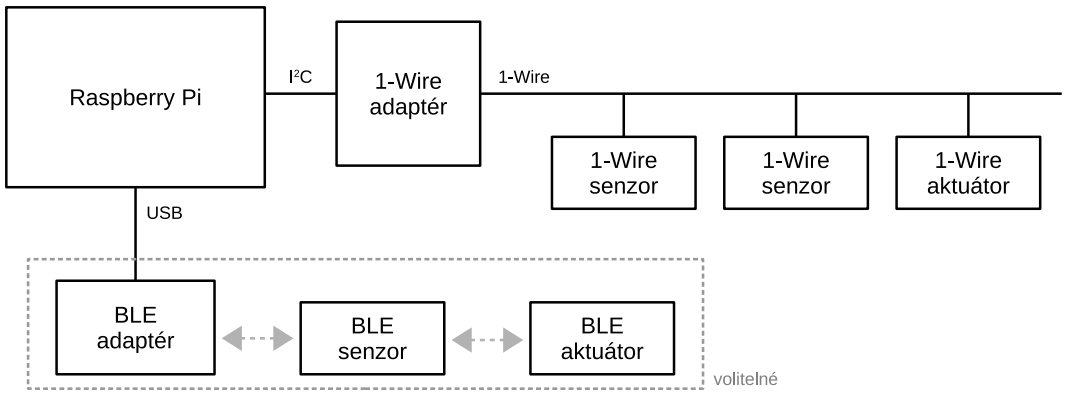
Pro úplnost a lepší pochopení jsou zde uvedeny abstrakce, které byly v projektu použity. Tyto abstrakce jsou zohledňovány jak v řídicí službě, tak v ovládací aplikaci.

- **Místnost** – abstrahuje fyzicky existující místnost, do které mohou být umístěny jednotlivé automatizační celky (okruhy). U místnosti je důležitý zejména její název a patro.
- **Zařízení** – cokoliv, co je připojeno k automatizačnímu systému. U zařízení je důležitý jeho typ, zde rozlišujeme dva – senzor a aktuátor. Ze senzoru je data možné pouze číst, z aktuátoru lze číst jeho aktuální stav a zápisem tento stav změnit. Pokud se jedná o aktuátor, musí být dodržena podmínka, že jej lze řídit pouze z jednoho místa, aby nedocházelo ke kolizím. Zařízení je pouze obálka pro jednotný přístup části řídicí služby a uživatele k zařízením se stejným účelem (např. senzor teploty). Každé zařízení má přiřazenou technologii, skrz kterou s ním bude manipulováno.
- **Technologie** – zapouzdřuje implementaci pro obsluhu zařízení. Přidává tedy možnost, jak připojit k systému více zařízení se stejným účelem skrz různá rozhraní. Příkladem může být připojení jednoho senzoru přes sběrnici 1-Wire a druhého přes Bluetooth Low Energy. Z uživatelova pohledu se jedná o dvě podobná zařízení určená ke stejnému účelu, z technologického pohledu je třeba každé obsloužit jiným způsobem.
- **Soket** – jedná se o odlišný termín od klasického socketu z operačních systémů. Zde je soketem myšlena zásuvka, do které lze přiřadit jedno či více zařízení stejného účelu (virtuálního typu hodnoty, který vrací). Smyslem soketu je možnost seskupit zařízení stejného účelu a přistupovat k nim jako k celku. Díky tomuto kroku bude systém schopen v rámci okruhu získávat data z více senzorů stejného typu, resp. ovládat více aktuátorů, jako by se jednalo o jedno zařízení. Každý soket má své unikátní jméno v rámci ovladače, díky kterému k němu lze přistupovat.
- **Ovladač** – určuje funkcionalitu, která bude vykonávána. Jedná se o implementaci algoritmu pro řízení konkrétní automatizační činnosti. Každý ovladač vyváží alespoň jeden soket, do kterého je možné připojit zařízení určitého účelu (podle virtuálního typu hodnoty). Počet možných soketů není shora omezen. Ovladač je jednoznačně rozpoznatelný podle svého unikátního jména. Jiný název pro ovladač v rámci této práce je modul, jelikož se jedná o rozšiřitelnou záležitost.
- **Okruh** – celek, který spojuje předchozí pojmy. Jedná se o nejmenší celek, který lze ovládat. Každý okruh má přiřazen právě jeden ovladač, který určuje jeho účel. Díky definici ovladače mohou v okruhu vzniknout sokety, do kterých lze přiřadit jednotlivá zařazení požadovaného typu. Okruhy lze vytvářet v místnostech, přičemž je možnost vytvoření více okruhů stejného ovladače v jedné místnosti. Ke každému okruhu je možné přiřadit program v závislosti na dnu v týdnu, který bude určovat plánovanou hodnotu v čase. Dále je možné okruhu plánovat operativní změny – rychlé změny, které zastíní naplánovaný program. Operativních změn může být naplánováno více, podmínkou je, že se nesmí překrývat v čase.

Kapitola 5

Návrh hardwarové platformy

Srdcem celého projektu je počítač Raspberry Pi, na kterém běží ovládací služba, jejímž úkolem je sbírat data ze senzorů a řídit aktuátory podle algoritmu implementovaného v ovladači (modulu). Algoritmus pro komunikaci se zařízeními je implementován v objektu konkrétní technologie. K počítači tedy může být současně připojeno více zařízení se stejným účelem, ovšem na bázi jiné technologie.



Obrázek 5.1: Schéma propojení Raspberry Pi se zařízeními

V této práci bylo použito sběrnice 1-Wire, která se k počítači připojí přes externí adaptér. Z adaptéru je pak možné propojit zařízení na sběrnici do zamýšlené topologie. Všechna zařízení vybrané sběrnice je možné propojit pouze vodičem, což by mělo být dostačující, vzhledem k povaze projektu, ve kterém budou zařízení pevně instalována a minimálně přenášena. Pokud ovšem vznikne potřeba přidat bezdrátová zařízení, rozšíření by se mělo skládat pouze z nainstalování nového adaptéru do počítače a naprogramování objektu technologie, který bude skrze nově nainstalovaný adaptér komunikovat se zařízeními. Příkladem, který je uveden na obrázku 5.1, může být technologie Bluetooth Low Energy, jejíž adaptér se připojí do USB portu a skrze něj se bude bezdrátově komunikovat s BLE zařízeními. Tato varianta je zde uvedena pouze jako příklad možného rozšíření o bezdrátová zařízení a nebude v této práci dále rozebírána.

5.1 Konektor a kabeláž

Z důvodu drátového propojení zařízení na sběrnici 1-Wire je vhodné zavést jednotné konektory a jejich zapojení. Na všech navrhovaných deskách byly použity konektory RJ45, které jsou oblíbené pro jednoduchou montáž na kabel. S použitím konektorů RJ45 se přímo nabízí využít UTP kabeláže k propojení, z důvodu velké rozšířenosti a především nízké ceny. Jelikož přes kabel nejsou přenášena velká pásma, postačí kabel Cat4 nebo vyšší. Na deskách jsou vždy dva konektory, aby bylo možné sběrnici propojit s dalším zařízením. Všechny použité signály jsou na konektorech propojené, až na piny 7 a 8. Z toho vyplývá, že pro účely propojení zařízení je nutné použít rovný kabel (na obou koncích zapojený buď podle standardu T-568A nebo T-568B).

Zapojení	Pin 1	Pin 2	Pin 3	Pin 4	Pin 5	Pin 6	Pin 7	Pin 8	RJ45
		Pin 1	Pin 2	Pin 3	Pin 4	Pin 5	Pin 6		RJ12
			Pin 1	Pin 2	Pin 3	Pin 4			RJ11
Dallas Semiconductor RJ11 standard			NC	DQ	GND	NC			RJ11
Dallas Semiconductor RJ12 standard		+5 V	GND	DQ	GND	NC	+		RJ12
Použité zapojení	NC	+5 V	NC	DQ	GND	NC	RJGP	RJGP	RJ45

Tabulka 5.1: Porovnání zapojení různých druhů konektorů

Pro 1-Wire sběrnici neexistuje standard od výrobce k zapojení konektoru RJ45. Různí výrobci zařízení s konektorem RJ45 pro tuto sběrnici navíc zapojení signálů interpretují mírně odlišně. Existují však standardy pro zapojení konektorů RJ11 (konektor se 4mi piny) a RJ12 (konektor se 6ti piny), které lze do použitého konektoru zastrčit. U většiny hotových zařízení s konektorem RJ45 lze nalézt signál +5 V na pinu 2, data na pinu 3 a zem na pinu 4. Zbylé signály se u různých výrobců liší. Právě kompatibilita se zapojením komerčně dostupných zařízení a dodržení standardů pro kompatibilní konektory s menším počtem pinů byl výchozí bod návrhu zapojení. V použitém zapojení je zapojení signálů téměř shodné se zapojením konektoru RJ12, přibyly však dva signály `RJGP`, které jsou na navrhovaných deskách vyvedené zvlášť vedle každého konektoru pro případné budoucí rozšíření. Takto je možné poslat přes volný pár vodičů v kabelu např. vyšší napětí, než je +5 V, případně jiný signál. Srovnání zapojení jednotlivých konektorů je znázorněno v tabulce 5.1.

5.2 Modul pro buzení sběrnice 1-Wire

Pro připojení sběrnice 1-Wire k počítači Raspberry Pi byl vybrán poslední ze způsobů uvedených v kapitole o sběrnici – připojení přes sběrnici I^2C. Ta se na počítači vyskytuje dvakrát, konkrétně bude použit vývod I^2C-1, který je vyveden na pinech 3 (SDA) a 5 (SCL) GPIO konektoru. Procesor Raspberry Pi pracuje na 3,3 V logice, která je pro použití na sběrnici při delších vzdálenostech nevhodná, a proto bude potřeba použít dvoukanálový převodník napěťových úrovní. Na straně počítače se tak bude pracovat s 3,3 V logikou a na straně budiče s požadovanou 5 V logikou signálů SDA/SCL. Pro buzení sběrnice 1-Wire bude využit most DS2482-100 [8], který nabízí jeden kanál sběrnice.

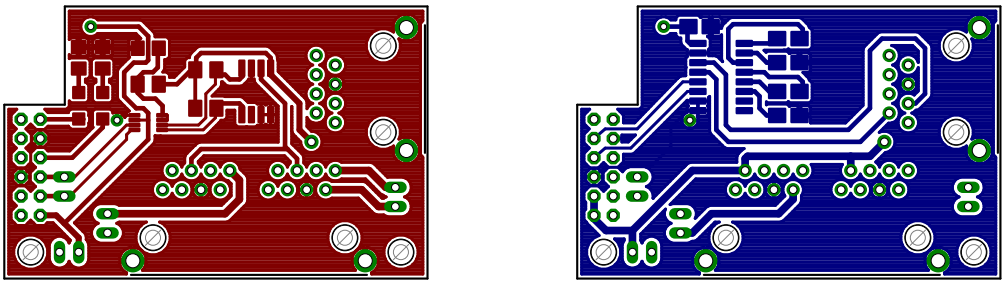
Cílem návrhu je vytvořit modul, který se připojí na část GPIO portu počítače a bude poskytovat výstup požadované sběrnice. Dalšími kritérii jsou velikost desky, která by měla

být co nejmenší, a ponechání nevyužitých pinů GPIO portu volných pro další aplikace. Pro snazší konfiguraci počítače bez podpory síťového prostředí se na desce bude nacházet výstup sběrnice RS232. Zde opět nastává problém s použitými napěťovými úrovněmi v počítači. Tento problém lze vyřešit integrovaným obvodem MAX3232, který se s pomocí nábojové pumpy postará o převod 3,3 V logiky počítače na hodnoty ± 12 V, které jsou použity ve sběrnici RS232. Pro zmenšení rozměrů desky bylo přistoupeno k použití konektoru RJ45 místo pro sběrnici RS232 standardního konektoru DB9. Zapojení signálů konektoru je znázorněno v tabulce 5.2.

Poslední částí modulu jsou dvě barevné LED diody, které budou indikovat stav běžící aplikace. Červená LED dioda je připojena k pinu GPIO4, zelená k pinu GPIO17. Obě LED diody je možné rozsvítit přivedením logické 1 na uvedené piny.

Pin	1	2	3	4	5	6	7	8
Signál	NC	RxD	TxD	NC	GND	NC	NC	NC

Tabulka 5.2: Zapojení konektoru RJ45 použitého pro sběrnici RS232



(a) Vrchní strana (b) Spodní strana

Obrázek 5.2: Otisk desky navrženého modulu *bus-driver*

Vyrobený modul poskytuje dva vývody pro sběrnici 1-Wire a jeden vývod pro sběrnici RS232. Velikost rozšiřující desky je zhruba třetinová oproti velikosti počítače Raspberry Pi a lze ji nasadit na všechny verze a modely počítače, jelikož vrchní část konektoru je ve všech verzích kompatibilní. Na desce jsou umístěny dva otvory, kterými ji lze při použití distančních sloupků délky 11 mm pevně spojit s deskou počítače. Toto je možné pouze u verzí počítače, které mají 40 pinový GPIO konektor (verze 1+ a výše). Na desce lze dále nalézt výstup použité sběrnice I^2C-1 pro připojení dalších zařízení a výstup napájecích napětí. Na horní straně desky lze nalézt výstupní konektory, převodník napěťových úrovní, budič 1-Wire sběrnice a kontrolní LED diody. Na spodní straně je umístěn integrovaný obvod pro převod napěťových úrovní RS232 s potřebnými kondenzátory pro nábojovou pumpu. Otisky desky vyrobeného modulu lze nalézt na obrázku 5.2. Fotografii vyrobeného modulu nasazeného na počítači lze nalézt v příloze A.

5.3 Modul s osmi relé

Pro automatizační účely je vhodné mít aktuátor, který poslouží jako spínač větší zátěže. V této práci se jedná o modul s osmi relé, z nichž každé dokáže spínat až 10 A při 250 V AC nebo 15 A při 24 V DC. Tímto modulem lze ovládat mnoho zařízení, k jejichž ovládání stačí

dva stavy (zapnuto/vypnuto), ale vyžadují spínání vyššího napětí nebo proudu. V rámci této práce bude hlavní využití modulu jako spínače dvoupolohových elektrických termopohonů, které ovládají topení. Ty běžně fungují na principu, že se při přivedení napětí otevřou a při odpojení zavřou. Druhů termopohonů je mnoho, vybraná varianta je výhodná pro tuto práci i z důvodu, že při případném výpadku napájecího napětí se hlavice samy zavřou. Modul bude dále možné v kombinaci s příslušným ovladačem použít ke spínání spotřebičů v domácnosti.

Srdcem modulu je integrovaný obvod DS2408, který nabízí 8 nezávislých výstupních kanálů, kterými ovšem může protékat jen velmi malý proud. Jelikož proud potřebný pro sepnutí relé se pohybuje okolo 70 mA a proud kontrolní LED diodou okolo 15 mA, bude nutné okruhy spínat přes tranzistor. K tomuto účelu byl použit integrovaný obvod ULN2803A [15]. Jedná se o pole Darlingtonových tranzistorů přímo určené ke spínání požadovaných zátěží. Jelikož výstup z integrovaného obvodu DS2408 je v sepnutém stavu připnut tranzistorem k zemi (0 V), bude nutné použít invertory, pokud chceme mít v tomto stavu relé sepnuté. Posledním integrovaným obvodem je tedy pole invertorů SN74HC240 [13], které je umístěno mezi obvodem poskytujícím osm výstupních pinů a tranzistorovým polem pro spínání zátěže. Jelikož výstupy obvodu DS2408 jsou představovány otevřeným kolektorem, je nutné připojit k nim napětí přes omezující rezistor, které bude zajišťovat vysokou logickou hodnotu, když nebude sepnut vnitřní tranzistor k zemi (na výstupu nízká log. hodnota). Toto je ošetřeno odporovou sítí umístěnou za 1-Wire integrovaným obvodem.

Použitá relé pracují na napětí 5 V, které je přítomné v konektoru, ovšem při sepnutí všech relé současně se odběr proudu pohybuje okolo 700 mA. Takto velký proud není možné odebírat z napájecího vodiče datového kabelu. Řešením je přivedení vlastního napájecího napětí pro spínací část desky. Přivedené napětí je regulováno regulátorem LM2940 [14] na požadovaných +5 V. Do konektoru přídavného napájecího napětí je tedy možné přivést 6 – 26 V DC, což odpovídá vstupnímu rozsahu použitého regulátoru. Vhodným zdrojem pro napájení spínací části tohoto modulu může být například zdroj 6 V DC s 1000 mA výstupem.

Ke každému relé patří jeden konektor se třemi vývody – přepínacími kontakty relé. Sepnutí je signalizováno rozsvícením LED diody příslušného relé. Pro kontrolu přítomnosti obou vstupních napětí (napájení z 1-Wire konektoru i přídavného napájení pro spínací část) jsou na desce umístěny další dvě LED diody. Navržená deska plošných spojů byla uzpůsobena rozměrům krabičky WM052C [1] s tím, že bude nutné do vrchní části krabičky vyříznout otvory pro konektory. Fotografie vyrobeného modulu a otisky jeho desky lze nalézt v příloze A.

5.4 Modul pro ovládání LCD displeje

Pro rychlý přehled o stavu senzorů a aktuátorů je vhodné mít v systému displej. V této práci bylo použito běžného LCD displeje s vlastním řadičem se 4mi řádky a 16ti sloupci – WINSTAR WH1604A. Cílem navrhované desky je konverze dat z 1-Wire sběrnice na signály rozhraní řadiče displeje s využitím integrovaného obvodu DS2408, který poskytuje 8 vstupně-výstupních pinů.

Sběrnice 1-Wire podporu pro displej bohužel nemá, ovšem OWFS tuto podporu pro displeje s řadičem HD44780 nebo kompatibilním s využitím obvodu DS2408 přidává [10].

[1] Plastová krabička Serpac WM052C: http://www.gme.cz/img/cache/doc/627/057/krabicka-plastova-wm052-c-abs-ip66-datasheet-1.pdf

V manuálu lze nalézt několik zapojení, jako funkční se ukázalo zapojení `Hobby Board`, které používá displej ve 4-bitovém módu. Díky tomu navíc vzniknou 3 volné piny pro všeobecné použití (např. tlačítka). Na navrhovaném modulu jsou tyto piny společně s napájecím napětím vyvedeny do konektoru `PIO_P0-P2`. Jelikož byl použit LCD displej s podsvícením, které vyžaduje proud okolo 440 mA, bylo přistoupeno k přidání externího napájení pro podsvícení. Vstupní napětí je opět regulováno regulátorem LM2940 [14], z toho vyplývá, že na vstup je možné přivést 6 – 26 V DC, stejně jako v případě modulu s 8mi relé. Použitý LCD displej je ovšem na světle dobře čitelný i bez nutnosti podsvícení. Externí napájení pro podsvícení displeje je proto označené jako volitelné. S použitím jednoho volného pinu by bylo možné ovládat podsvícení v závislosti na čase či intenzitě světla v místě displeje.

Navrhovaná deska svými rozměry, použitými dírami a umístěním výstupního portu odpovídá desce, na které je umístěn použitý LCD displej. Cílem je možnost umístit navrženou desku přímo pod displej s využitím distančních sloupků délky 5 mm. Na desce modulu se dále nachází trimr `R2`, kterým lze řídit jas displeje.

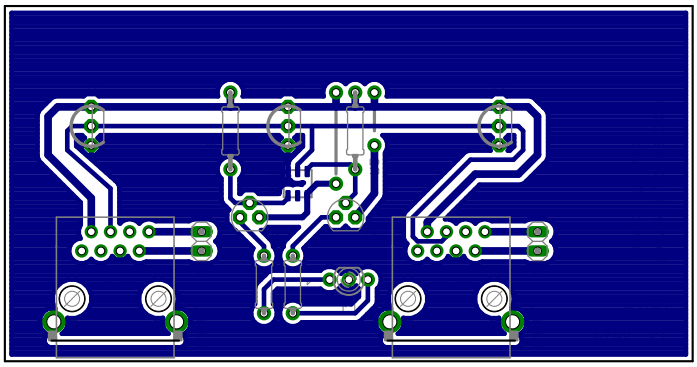
Výslednou desku s displejem lze připojit do 1-Wire sítě a s příslušnou softwarovou podporou zapisovat na displej. Komunikace s displejem je jednosměrná a tudíž na straně softwaru není možné rozpoznat velikost připojeného displeje a je nutné toto ošetřit v nastavení softwaru, který bude na displej vypisovat informace. Otisk obou stran desky a fotografii hotového modulu lze nalézt v příloze A.

5.5 Senzorový modul

Cílem senzorového modulu je poskytnout základnu pro monitorování teploty. Na desce se nachází tři místa pro digitální teplotní senzory DS18B20 v pouzdře TO-92 nebo pro upravené senzory vlhkosti. Modul poskytuje navržení rozhraní pro připojení 1-Wire sběrnice pomocí dvou RJ45 konektorů. Bonusem na desce je dvoubarevná LED dioda, která je spínána integrovaným obvodem DS2406, který lze na sběrnici adresovat a se změnou stavu jeho dvou výstupních pinů ovládat nezávisle dvě barevné složky kontrolky. Jelikož maximální proud pinem tohoto spínače je nedostačující pro rozsvícení LED diody, spínání mají na starost dva tranzistory BS170. Kontrolní LED diodu lze ovládat nezávisle na zbytku desky, což ji umožňuje použít k signalizování různých akcí, které nemusí souviset s vytápěním.

Výsledkem je jednoduchá jednostranná deska, která je svou velikostí koncipována do krabičky U-ICAS103[2], která se jevila pro použití se senzorem teploty jako vhodná, především díky množství větracích otvorů. Jako rozšíření by bylo vhodné desku zmenšit a vložit ji do menší krabičky podobného ražení i za cenu použití svorek namísto RJ45 konektorů pro připojení kabelu se sběrnicí. Fotografii vyrobeného modulu lze nalézt v příloze A, otisk desky na obrázku 5.3.

[2]Plastová krabička s větracími otvory: http://www.gme.cz/u-icas103-p622-834



Obrázek 5.3: Otisk desky senzorového modulu se součástkami

Všechny moduly byly navrženy v programu Eagle Professional 7.0.0 [11] a bylo použito jak klasických průchozích součástek, tak moderních SMD součástek pro povrchovou montáž [17]. Při návrhu hardwarových modulů bylo navrženo více modulů, než bylo vyrobeno, z důvodu nákladnosti výroby desek plošných spojů. Zdrojové soubory nevyrobených modulů `main_board` a `1W_bus_driver` jsou umístěny na přiloženém CD. První jmenovaný modul měl sloužit jako vývojová deska pro Raspberry Pi, nabízející 8 nezávislých sběrnic 1-Wire, RS232 a vývod všech GPIO pinů. Druhý modul byl navržen jako zásuvný do prvního a obsahuje integrovaný obvod, který řídí 8 kanálů sběrnic 1-Wire. Zásuvný z důvodu jednoduché výměny při poškození. Oba tyto moduly byly nahrazeny jednodušším a mnohem menším modulem `bus-driver`. Veškeré zdrojové soubory z vyrobených modulů jsou umístěny na přiloženém CD včetně diagramů zapojení, souborů se součástkami a osazovacími diagramy.

Kapitola 6

Návrh a implementace řídicí služby

Tato kapitola se zabývá popisem služby, resp. programu, jehož cílem je řídit automatizační jednotky zvané okruhy. Bude podrobně vysvětlena architektura jádra programu i všech jeho rozšiřitelných složek, tedy především zařízení, ovladačů a technologií. Tento program bude aktivní na počítači Raspberry Pi od spuštění až do okamžiku vypnutí. Aby mohl být spuštěn operačním systémem po startu jako služba, byl vytvořen System V inicializační skript, který je součástí instalace projektu.

Program se skládá ze dvou částí – jádra a rozšiřitelných částí. Jádro zajišťuje správný běh aplikace, její interakci s okolím, správné ukončování a restartování. Neméně důležitým úkolem je také vytvoření instancí tříd včetně jejich propojení a rozdělení do vláken, jelikož se jedná o dynamické vytváření objektů podle stavu perzistentního úložiště, který se mění. Každou změnu v perzistentním úložišti je nutné promítnout do aplikace. Nejjednodušší možností je její restart, což ovšem znamená vytvořit všechny objekty znovu podle aktuálního stavu. Tato možnost se hodí pouze při větších změnách, pro menší změny je vhodné mít způsob, jak měnit strukturu aplikace bez úplného zastavení jejího výpočtu. Jádro dále zastřešuje řadu funkcí, výběrem lze uvést například správu chybových hlášek, zpřístupnění perzistentního úložiště, reakci na příchozí signály od operačního systému, komunikaci mezi vlákny a okolním světem či plánovač hodnot. Vše, co se týká jádra, bude implementováno ve jmenném prostoru `core`.

V rozšiřitelné části jde o rozšíření aplikace o konkrétní automatizační funkčnost. Celkem jsou tři podčásti, které lze rozšířit – technologie, zařízení a ovladač. Pro dosažení funkčnosti spolu musí všechny tři části spolupracovat. Důvodem rozdělení je do určité míry jejich nezávislost a rozdělení tak napomáhá ke znovupoužitelnosti tříd. Každá z podčástí má vlastní jmenný prostor, postupně jde o prostory `technology`, `device` a `module`.

Kompletní diagram tříd řídicí aplikace lze nalézt na přiloženém CD, rozdělený diagram pak v příloze D.

6.1 Volba implementačního jazyka

Pro implementaci řídicího programu byl vybrán jazyk C++ z několika důvodů. Prvním a hlavním důvodem je podpora objektové orientace, která je klíčová pro implementaci rozšiřitelné části – zařízení, ovladačů a technologií. Všechny tyto 3 entity jsou objekty, které musí dědit od svého předka, aby mohly být použity. Druhým důvodem je blízkost vybraného jazyka k hardwaru. To umožní efektivně implementovat algoritmy, obsluhující zařízení připojená k počítači. Ty se budou nacházet v objektech technologie. Jedná se o překládaný

jazyk, což povaze této aplikace plně vyhovuje, jelikož překlad proběhne jednou a poté bude již pouze spouštěn do doby, než bude dostupná novější verze nebo jej bude třeba rozšířit o novou automatizační funkčnost. Výhodou překládaného programu je rychlost běhu oproti interpretovanému za cenu času stráveného překladem. Při překladu je také kontrolována typová kompatibilita a tak by nemělo dojít k pádu programu způsobeného právě tímto problémem. [12]

Dalším důležitým faktorem je podpora vybraného jazyka na cílové platformě. Na použitém počítači Raspberry Pi běží operační systém Raspbian, který obsahuje překladač `g++` pro vybraný jazyk. Pokud by překladač nebyl v operačním systému nainstalován, lze jej nainstalovat z repozitáře.

6.2 Správce instancí

Správce instancí – třída `core::InstanceManager` má zodpovědnost za vytvoření a zrušení většiny objektů potřebných pro chod aplikace. Jedná se hlavně o objekty, které je potřeba vytvořit a propojit podle aktuálního stavu perzistentního úložiště dat – relační databáze. Reflektuje tedy konfiguraci, kterou uživatel provedl ve webovém informačním systému. Jedná se především o objekty zařízení, technologií, ovladačů, soketů a objektů obálek pro běh modulu ve vlákně.

Další skupinou objektů, jejichž instance jsou vytvářeny, jsou objekty jádra, které budou později využívány rozšiřitelnými částmi systému. Jedná se především o instanci třídy `core::MySQLPool`, která zpřístupňuje konektivitu k databázi, instanci třídy implementující rozhraní `core::Logger`, které má za cíl ukládat veškeré provozní a chybová hlášení, instanci třídy pro posílání zpráv či instanci třídy pro získání naplánovaných hodnot z databáze.

Jelikož se jedná o aplikaci, jejíž výpočet je možné provádět paralelně, je její provádění rozděleno do několika vláken, přičemž je využíváno vláken nabízených jazykem C++11. V rámci sjednocení ovládání a kontrolování výpočtu ve vláknech je každý výkonný kód zabalen do třídy, která dědí od abstraktní třídy `core::ThreadWrapper`. Tato abstraktní třída obsahuje nástroje pro diagnostiku stavu vlákna a dvě čistě virtuální metody, které je nutné implementovat. Jedná se o metodu `run`, která je invokována při spuštění výpočtu ve vlákně. Jelikož se jedná o aplikaci, která běží v nekonečné smyčce po celou dobu běhu systému, z této metody by se výpočet ve vlákně při správné funkci neměl vrátit. Pokud výpočet metody skončí a nejedná se o konec aplikace, jde o chybu, kterou je nutno řešit. Pro ukončení běhu výpočtu vlákna je asynchronně volána metoda `stop`, která má za úkol pouze nastavit příznak k ukončení, aby bylo možné se vrátit z provádění metody `run`. Jelikož se jedná o asynchronní volání metody vzhledem k výpočtu ve vlákně, je vyžadováno jej v konkrétní implementaci obálky vlákna ošetřit vhodnými synchronizačními prostředky. Správce vytváří celkem 4 druhy vláken:

- `core::SignalThread` – úkolem vlákna je odchytávat signály od operačního systému a iniciovat vhodnou reakci na ně. Jedná se především o signály `SIGTERM` a `SIGINT`, které způsobují ukončení aplikace a dále signál `SIGUSR2`, který způsobí její restart.
- `core::CommunicatorThread` – smyslem vlákna je zprostředkovat meziprocesovou komunikaci a konvertovat ji na komunikaci, která probíhá uvnitř procesu aplikace mezi vlákny. Vlákno čeká blokujícím způsobem na nové připojení k unixovému soketu, který vytvoří při spuštění. Po navázání spojení přijme, deserializuje, odešle zprávu skrz rozhraní pro vnitřní komunikaci konkrétnímu příjemci a odpoví odesílateli zprávou, zda se přijetí zprávy povedlo bez chyb. Pokud se jedná o zprávu, jejímž adresátem

je *SYSTEM*, není zpráva odesílána k příjemci, ale je na místě zpracovávána. Jedná se především o zprávy, které zjišťují stav systému či iniciují restart.

- `core::LedStatusThread` – vlákno ovládá dvě LED diody na modulu *bus driver* připojeného k počítači Raspberry Pi. Informace o stavu systému získává z chybových hlášení, jelikož zároveň implementuje rozhraní `core::Logger`.
- `core::ModuleThread` – poslední druh vlákna má na starost zabalit provádění jednoho ovladače a implementovat jeho životní cyklus. Každý ovladač běží v samostatném vlákně, jelikož jeho výpočet je nezávislý na jiném ovladači.

Na začátku běhu aplikace je potřeba vytvořit objekt správce instancí a metodou `prepare` spustit přípravu všech instancí objektů. Pokud dojde k chybě, je vyhozena výjimka dědící od `std::exception` s popisem chyby. Zotavení z chyby vzniklé v tomto kroku není možné a tak vznik chyby vede k ukončení celé aplikace. V tomto kroku není dostupný ani objekt pro ukládání chybových hlášení a z toho důvodu je nutné postarat se o uložení chybové hlášky při přípravě jiným způsobem. Při vzniku chyby v tomto kroku je chybová hláška vypsána na standardní chybový výstup a program je ukončen. Po úspěšné přípravě lze použít metody pro získání instancí objektů ze správce, zejména pak pole s předpřipravenými obálkami, které je nutné spustit ve vláknech. Pátým typem vlákna je pak to, které se spustí s aplikací, provede přípravu a pak blokujícím způsobem čeká na ukončení všech ostatních vláken. Veškeré instance objektů vytvořených správcem jsou dealokovány při volání destruktoru objektu správce.

6.3 Zařízení

Objekt konkrétního zařízení je pouze jakousi obálkou, která představuje jeho fyzický protějšek. Důvodem tohoto kroku je oddělení typů zařízení od konkrétní technologie. Typem zařízení může být např. senzor teploty, jehož obsluha se bude lišit v závislosti na použitém fyzickém zařízení – získání dat z termistoru je nutné provést jiným algoritmem, než získání dat z digitálního senzoru. Z tohoto důvodu je obsluha fyzického zařízení svěřena objektu konkrétní technologie, jejíž ukazatel je součástí objektu předka. Všechna zařízení sdílí jmenný prostor `device`, jejich třídy jsou umístěny v adresáři `src/device` a dědí od abstraktní třídy `device::Device`, ve které jsou umístěny atributy zařízení z databáze a nadefinovány čistě virtuální metody, které je nutné ve třídě konkrétního zařízení implementovat:

- `void init()`
- `double getValue()`
- `void updateValue()`
- `void setValue(double)`
- `std::string getFormattedValue()`

První metoda slouží k inicializaci zařízení, kterou nelze provést v konstruktoru – inicializace, která je závislá na atributech rodičovského objektu `device::Device`. Jeho atributy jsou nastaveny těsně po vytvoření objektu a ihned poté je volána tato funkce. V praxi ji lze použít pro zjištění aktuální hodnoty fyzického zařízení či zjištění funkčnosti komunikace s fyzickým zařízením.

Metodou `getValue` lze získat aktuální hodnotu zařízení. Hodnota je datového typu double, avšak záleží na povaze zařízení, jak hodnotu interpretovat. Je nutné podotknout, že zařízení má možnost hodnotu kešovat, jak uzná za vhodné s cílem zmenšit počet (mnohdy i zbytečných) přenosů aktuální hodnoty z/do fyzického zařízení. K vynucení načtení skutečné hodnoty z fyzického zařízení lze použít metodu `updateValue`, která aktualizuje hodnotu uvnitř objektu, ale nic nevrací.

Pro nastavení hodnoty zařízení lze použít metodu `setValue`. V případě volání této metody mohou nastat tři stavy. Prvním stavem je úspěšná operace, kdy je metoda opuštěna bez vyvolání výjimky. Jelikož hodnotu lze nastavovat pouze aktuátorům, druhým stavem je nepodporovaná operace, která nastane v případě volání metody nad zařízením typu senzor. Chyba je signalizována vyhozením výjimky `device_invalid_operation`. Třetím stavem je chyba, která může nastat při volání nastavení hodnoty v příslušné technologii a je signalizována výjimkou `technology::value_error`.

Poslední metoda z uvedených, kterou je nutné implementovat, je `getFormattedValue`. Cílem je vracet formátovanou hodnotu i s jejím významem. Například pro použití s relé se nabízí texty „On“, resp. „Off“, pro použití se senzorem teploty může být vrácen řetězec s teplotou, která má fixní počet desetinných míst a značku stupnice.

Důležitým atributem, který zdědí každý objekt zařízení od svého předka `device::Device`, je ukazatel na konkrétní technologii, ke které zařízení patří. Právě přes tento ukazatel může zařízení získávat aktuální hodnoty ze svého fyzického protějšku. Z toho vyplývá myšlenka kešování aktuální hodnoty a její ukládání do databáze pouze když je to skutečně potřeba, jelikož právě konkrétní zařízení ví nejlépe, zda je potřeba do databáze ukládat pouze změny hodnot nebo vzorkovat aktuální stav při každém dotazu na hodnotu. Neméně důležitým atributem je řetězec označovaný `tId`, jež má význam unikátního identifikátoru zařízení v rámci technologie.

6.4 Okruh

Jak již bylo předesláno v předchozích kapitolách, okruh spojuje senzory a aktuátory s ovladačem a vytváří tak celek, který lze řídit a lze mu naplánovat program. V tomto řídicím programu jeden okruh představuje instance třídy `core::Circuit`. Okruh sám o sobě žádný výkonný kód ovladače neobsahuje, ale je podle pravidel vložen do kontejneru okruhů příslušného ovladače a tím je pak řízen.

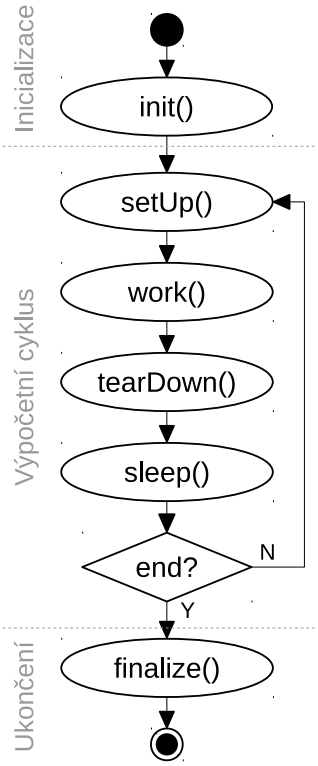
Okruh obsahuje atributy převzaté z databáze, konfigurátor pro získání své aktuální konfigurace z databáze a plánovač, díky kterému může získat aktuálně naplánovanou hodnotu z databáze. Nejdůležitějším atributem je však asociativní kontejner, který na základě unikátního jména soketu v rámci ovladače zpřístupní příslušný objekt `core::Socket`, který představuje rozhraní komunikace s přiřazenými zařízeními. Pomocí soketu je pak možné hromadně získávat hodnoty ze senzorů a nastavovat hodnoty aktuátorům.

6.5 Ovladač

Ovladačem (resp. modulem) označujeme objekt, který implementuje chování určitého okruhu. Jedná se tedy o jednu specifickou činnost, například ovládání vytápění v místnosti. Všechny třídy ovladačů jsou umístěny v adresáři `src/module`, sdílí jmenný prostor `module` a dědí od abstraktní třídy `module::BaseModule`. V abstraktní třídě předka jsou umístěny všechny potřebné atributy k vykonávání konkrétního obslužného algoritmu. Jedná

se o atributy z databáze, objekt pro práci s databází, konfigurátor pro získávání konfigurace z databáze, rozhraní pro přijímání a posílání zpráv, logger a eventLogger pro trasování stavů okruhu. Nejdůležitějším atributem je pole okruhů, patřící k danému ovladači. V nemodifikované verzi jsou do tohoto pole přiřazeny všechny okruhy, které mají být daným ovladačem ovládány. Je ovšem možné v konfiguraci omezit počet okruhů, které budou přidány do tohoto pole. Pokud bude celkový počet okruhů patřící k danému ovladači větší, vytvoří se další instance objektu stejného ovladače a okruhy budou předány do nově vytvořeného objektu ovladače. Důležitým faktem je, že obsluha každého vytvořeného ovladače probíhá ve speciálním vlákně. Maximálním počtem okruhů v poli ovladače tedy lze přímo ovlivnit počet běžících vláken.

Každý ovladač běžící v samostatném vlákně má svůj životní cyklus, který je zobrazen na obrázku 6.1 a je implementován ve třídě `core::ModuleThread`. Všechny metody zobrazené v životním cyklu ovladače (kromě metody `sleep`) jsou v předkovi deklarovány jako čistě virtuální, a tudíž je nutné je v konkrétním ovladači implementovat. Životní cyklus ovladače se skládá ze tří částí, z nichž je možné obsluhu ovladače ukončit pouze na konci části zvané výpočetní cyklus či fatální chybou v ovladači, která vyhodí výjimku. Ve třídě, která životní cyklus implementuje, jsou odchytávány a zaznamenávány všechny výjimky dědící od `std::exception`.



Obrázek 6.1: Životní cyklus ovladače

V první části zvané inicializace je vytvořen objekt konkrétního ovladače. Po vytvoření objektu je ihned jeho předek naplněn všemi potřebnými atributy a zavolána metoda `init`, která slouží k inicializaci ovladače a kterou nelze kvůli nedostupným atributům předka provést v konstruktoru. Tato metoda je volána jednou, pouze po vytvoření a naplnění objektu a je tedy vhodné použít ji k účelům předplacení si příchozích zpráv určitého předmětu či inicializaci vnitřních struktur potřebných pro pozdější vykonáváni v další části cyklu.

V části výpočetního cyklu jsou provolávány tři metody, které implementují chování konkrétního ovladače. První z těchto metod se jmenuje `setUp` a jejím úkolem je připravit vše potřebné pro bezproblémový výkon požadované práce v jednom cyklu. Metoda je vhodná pro zpracování přijatých zpráv či reflektování změn konfigurací. V metodě `work` je vykonávána hlavní práce ovladače nad všemi okruhy, které jsou umístěny v poli předka. Hlavním úkolem této metody je transformovat množinu polí vstupů (každé vstupní pole pro jeden okruh) a transformovat ji na množinu výstupních polí. Cílem je tedy ze všech dostupných informací (vnitřní stav, konfigurace, zprávy, naplánovaná hodnota, hodnoty ze senzorů) podle vlastního algoritmu vypočítat a nastavit nový stav aktuátorům. V metodě `tearDown` je vhodné provést úklid vnitřních struktur, pokud je to potřebné a nastavit metodou `delay(unsigned time)` čas v milisekundách, za který bude opět zavolána první metoda cyklu, pokud nenastane jeho konec. Zmiňovanou funkci je nutné volat v každém cyklu, jelikož vnitřní čítač pro zpoždění je se začátkem každého cyklu vynulován. Provedení zpoždění voláním jiné funkce je nevhodné z hlediska potenciálně nesprávného, resp. dlouho trvají-

cího ukončení vlákna patřícího ovladači. Funkce `delay` totiž pouze nastaví vnitřní čítač pro požadované zpoždění. Samotné zpoždění je realizování funkcí `sleep`, která pomocí série mikrospánků může plynule reagovat na požadavek pro ukončení vlákna, ve kterém výpočet ovladače probíhá.

Po vytvoření požadavku na konec vlákna a jeho indikováním v metodě `sleep` se přechází do třetí fáze životního cyklu, kterou je ukončení. V rámci této fáze je volána metoda `finalize`, která má zajistit uložení všech potřebných informací tak, aby při příštím startu bylo možné začít ve stejném stavu, ve kterém se nyní končí. Nakonec je volán destruktor objektu, který je zodpovědný za dealokaci všech vnitřních struktur dynamicky alokovaných.

6.6 Technologie

Jako technologie je v části řídicího programu označován objekt, skrze který lze komunikovat s fyzickými zařízeními jedné technologie (např. 1-Wire či Bluetooth Low Energy). Objekty technologií jsou umístěny v adresáří `src/technology` a sdílí jmenný prostor `technology`. Tyto objekty musí dědit od předka – abstraktního objektu `technology::Technology`, který zpřístupňuje údaje z databáze, seznam zařízení a logger. Kromě toho deklaruje metody, které jsou čistě virtuální a tudíž v nově vytvářeném objektu konkrétní technologie je nutné tyto metody implementovat. Jedná se následující metody:

- `void init()`
- `double getValue(device::Device *)`
- `void setValue(device::Device *, double)`
- `unsigned getNewDevices(std::vector<device::FoundDevice> &)`

První metoda slouží k inicializaci objektu, zejména té části inicializace, kterou nelze provést v konstruktoru, jelikož při konstrukci nejsou v objektu přítomny atributy, které obsahuje předek `technology::Technology`. Ty jsou do objektu vloženy ihned po jeho vytvoření a až poté je tato metoda volána. Pokud se vyskytne kritická chyba, kvůli které by nebylo možné obsluhovat požadavky zařízení, lze ji indikovat vyhozením výjimky `init_error`.

Metoda `getValue` vrací aktuální hodnotu daného zařízení, které přísluší této technologii a jehož ukazatel je parametrem. V této metodě je ukryta skutečná implementace komunikace s fyzickým zařízením. Pokud nelze ze zařízení získat hodnotu, je potřeba tento stav signalizovat vyhozením výjimky `value_error`. Pokud není vyhozena výjimka, vrácená hodnota je považována za platnou.

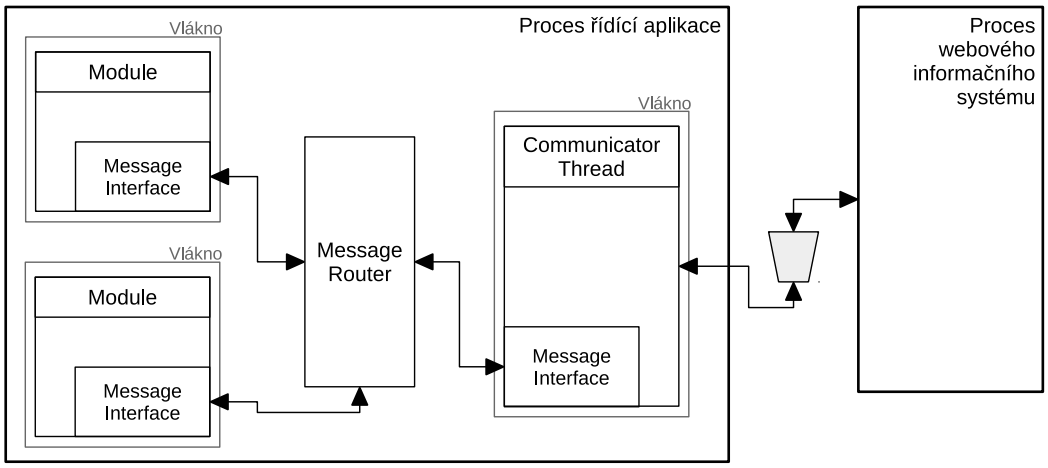
Metodou `setValue` lze nastavit hodnotu zařízení. Parametry metody jsou ukazatel na nastavované zařízení a nastavovaná hodnota. V některých případech nelze přes hodnotu typu double přenést hodnotu, kterou je potřeba zařízení nastavit (např. pokud je potřeba nastavit textový řetězec). V tom případě lze postupovat způsobem, že nastavovanou hodnotu zapouzdříme do objektu zařízení a v technologii provedeme statické přetypování. Takto je řešen problém nastavení řetězce do zařízení typu displej. Pokud se hodnotu pro dané zařízení z jakéhokoliv důvodu nepodaří nastavit, je tato skutečnost signalizována výjimkou `value_error`. Při vyhození výjimky je počítáno s tím, že stav zařízení zůstal nezměněn.

Poslední čistě virtuální metoda `getNewDevices` slouží k vyhledávání nových zařízení v technologii. Nové zařízení je charakterizováno jako takové, které bylo nalezeno v dané technologii a zároveň není přítomné v seznamu zařízení. Cílem metody je přidat zařízení

vyhovující uvedené podmínce do vektoru, jehož reference je předána parametrem. Nalezený objekt je do vektoru vložen jako instance třídy `FoundDevice`, což je speciální třída určená k účelům vyhledávání nových zařízení. Její vlastností je, že umí vygenerovat SQL dotaz, kterým lze nové zařízení přidat do databáze.

6.7 Komunikace

Komunikace je v rámci řídicí aplikace používána k předávání informací mezi ovladači, okruhy a webovým systémem pro ovládání. Komunikační systém lze rozdělit na vnější a vnitřní, kdy vnitřní systém má za úkol doručovat zprávy v rámci procesu aplikace a vnější má na starost komunikaci s jinými procesy. Komunikace vždy probíhá mezi dvěma body, resp. dvěma rozhraními schopnými vysílat/přijímat zprávy. Pro lepší představu toků komunikace bylo vytvořeno schéma, které je zobrazené na obrázku 6.2.



Obrázek 6.2: Schéma komunikace v rámci řídicí aplikace včetně interakce s okolím

6.7.1 Zpráva

Zprávou rozumíme objekt třídy `core::Message`, který lze serializovat, přenést a deserializovat. Tvar zprávy je pevně daný, jejími atributy jsou adresa odesílatele, adresa příjemce, předmět a čtyři volitelné argumenty. Povinnými položkami zprávy jsou obě adresy a předmět. Adresa je objekt třídy `core::Message::Address`, která určuje typ adresáta a jeho číselný identifikátor. Možné typy jsou následující

- MODULE – adresátem je ovladač, nastavený identifikátor značí id ovladače
- CIRCUIT – adresátem je okruh, nastavený identifikátor značí id okruhu
- FRONTEND – adresuje webový informační systém. Identifikátor není brán v potaz
- SYSTEM – adresovaným je vlákno pro přijímání zpráv, které umí na systémové zprávy vhodně reagovat

6.7.2 Vnitřní komunikace

Pro úspěšnou komunikaci je potřeba rozhraní, které bude schopné zprávy přijímat a odesílat. Toto rozhraní je implementováno ve třídě `core::MessageInterface`. Instanci tohoto objektu obsahuje každý ovladač a při startu jsou všechny okruhy daného ovladače zaregistrovány pro přijímání zpráv na toto rozhraní. Na jedno rozhraní tedy přichází zprávy pro různé adresáty a je na ovladači, aby je rozlišil a správně interpretoval. Rozhraní obsahuje vnitřní buffer pro uschování přijatých zpráv, do doby než budou vyzvednuty. Přehled o rozhraní třídy osvětlí následující výčet jeho metod:

- `bool receiveMessage(Message *message)` – přijímá zprávu z okolí objektu
- `bool sendMessage(Message *message)` – odesílá zprávu z daného rozhraní
- `Message *getMessage()` – vrací první přijatou zprávu, NULL pokud je vyrovnávací buffer plný
- `Message *getMessage4Circuit(Circuit *c)` – vrací první přijatou zprávu pro specifikovaný okruh
- `void subscribeMessage(std::string subject)` – předepsání příjmu zpráv
- `void useSubscribing(bool use)` – povolení/zakázání funkce předepsaných zpráv. Implicitně povoleno

Veškerá manipulace s vyrovnávacím bufferem pro zprávy je chráněna mutexem a tedy je možné v jednu chvíli zprávy buď přijímat nebo odesílat.

Pro odeslání zprávy z popsaného rozhraní je potřeba prostředníka, který bude schopný ověřit identitu odesílatele a zprávu doručit, jelikož jedno rozhraní nezná ostatní. K tomuto účelu slouží objekt třídy `core::MessageRouter`. Tento objekt je dostupný již před vytvořením všech rozhraní, aby bylo možné předat jeho ukazatel v konstruktoru rozhraní. Rozhraní následně zaregistruje svůj ovladač směrovači, společně se všemi okruhy, které jsou součástí ovladače. Směrovač tak má ve svých tabulkách uložené adresy všech rozhraní a je schopný zprávu doručit. Doručení zprávy je provedeno metodou `sendMessage`, která jako parametry přijímá doručovanou zprávu a rozhraní, ze kterého se zpráva odesílá. Směrovač zkontroluje identitu odesílatele oproti svým záznamům v tabulkách a nalezne rozhraní, na které bude zpráva doručena. V následném kroku se zprávu pokusí doručit a vrací pravdivostní hodnotu, zda byla zpráva doručena. Pokud nemohla být zpráva doručena, směrovač ji dealokuje.

6.7.3 Vnější komunikace

Vnější komunikace slouží k posílání zpráv mezi řídicí aplikací a webovým systémem pro ovládání. Ke komunikaci se využívá Unixového soketu, který je vytvořen při startu aplikace. V současném stavu je implementována jako jednosměrná, tzn. odesílatelem je vždy webový informační systém. K přijímání zpráv je vyhrazeno jedno vlákno, jehož obálka je třídy `core::CommunicatorThread`. Úkolem vlákna je čekat na příchozí zprávu a v případě jejího úspěšného deserializování ji odeslat adresátovi. Tato třída obsahuje rozhraní stejné jako moduly, rozdíl je v tom, že je ve směrovači zaregistrováno jako typ frontend.

Zprostředkování komunikace mají na starost dvě třídy. První z nich: `core::SocketCommunicator` má zodpovědnost za vytvoření soketu a čekání na nové připojení. Vytvoření

soketu má na starost třídní metoda `createUnixSocket(std::string path)`, jejímž argumentem je absolutní cesta k soketu, který má být vytvořen. Metoda vrací ukazatel na objekt popisované třídy. K čekání na nové připojení slouží metoda `wait4Connection`. Tato metoda je blokující a tedy až do momentu příchodu nového připojení se setrvává v těle této metody. Při příchodu nového připojení metoda vrátí objekt `core::SocketConnection`, ze kterého lze číst textový řetězec metodou `read` a zapisovat jej metodou `write`.

V rámci implementace této práce je komunikace využita hlavně pro oznamování změn v konfiguraci, které byly provedeny z webového ovládacího systému, pro detekci běžící aplikace, případně pro její vzdálený restart.

6.8 Plánovač

Zodpovědností plánovače je zpřístupnit naplánovanou hodnotu pro určitý okruh. V rámci databáze je tedy nejprve nutné zjistit, zda je na hledaný den v týdnu naplánován nějaký program, a pokud ano, najít hodnotu odpovídající času dotazu. Do toho vstupuje operativní změna, která může zastínit program, jelikož má vyšší prioritu a dočasně tak nastavit jinou hodnotu. Plánovanou hodnotou je vždy hodnota v pohyblivé řádové čárce typu double. Její interpretace kvůli univerzálnosti však záleží až na ovladači, který okruh řídí.

Pro plánovač je připravena abstraktní třída `core::SchedulerInterface`, která obsahuje přístupový bod k databázi a logger. Jak již z názvu vyplývá, jedná se pouze o jakési rozhraní, které vyžaduje implementaci a to z důvodu pozdějšího rozšiřování. Nabízí pouze jednu čistě virtuální metodu `getScheduledValue`, která jako parametr přijímá ukazatel na okruh, pro nějž má být vrácena naplánovaná hodnota a vrací objekt `core::ScheduledValue` s nalezenou naplánovanou hodnotou. Plánovač je záměrně postaven na abstraktní třídě, aby bylo možné jej v budoucnu rozšířit, například s využitím kešování a chytrého algoritmu, který bude stále udržovat obsah validní vůči měnícímu se stavu databáze.

Vrácený objekt s naplánovanou hodnotou nemusí hodnotu skutečně nést, je tomu tak v případě, kdy v databázi žádná hodnota naplánovaná není. V takovém případě je objekt ve stavu nenastavené hodnoty. Před každým použitím hodnoty z objektu je proto vhodné zkontrolovat metodou `isSet` přítomnost plánované hodnoty. Metodami `isRegular` a `isOperative` lze zjistit, zda je naplánovaná hodnota operativní změnou nebo běžným programem. Metodou `getValue` získáme naplánovanou hodnotu typu double z objektu. V případě zavolání nad objektem, který obsahuje nenaplánovanou hodnotu, je vyhozena výjimka `core::no_scheduled_value`.

Představenou abstraktní třídu pro použití v této práci implementuje třída `core::DirectScheduler`. Jak již název napovídá, jedná se o plánovač, který hodnotu získává jedním optimalizovaným dotazem z databáze.

id	**select_type**	**table**	**type**	**key**
1	PRIMARY	operative_change	ref	fk_operative_change_circuit
2	UNION	program_schedule	ref	fk_circuit_program_circuit
2	UNION	program	eq_ref	PRIMARY
2	UNION	program_record	ref	fk_program_record_program
null	UNION RESULT	<union1,2>	ALL	NULL

Tabulka 6.1: Výřez tabulky znázorňující zpracování dotazu

```
(SELECT DATE_FORMAT(o.from, '%Y%m%dT%H%i%s') AS iso_from,
   DATE_FORMAT(o.to, '%Y%m%dT%H%i%s') iso_to, value, 0 AS regular
    FROM operative_change o WHERE (NOW() BETWEEN o.from AND o.to)
    AND o.circuit_id = <id_okruhu> LIMIT 1)
UNION
(SELECT CONCAT(DATE_FORMAT(NOW(), '%Y%m%d'), 'T', DATE_FORMAT(
   program_record.from, '%H%i%s')) AS iso_from, CONCAT(
   DATE_FORMAT(NOW(), '%Y%m%d'), 'T', DATE_FORMAT(program_record.
   to, '%H%i%s')) AS iso_to, value, 1 AS regular FROM
   program_schedule JOIN program ON program.id = program_schedule
   .program_id JOIN program_record ON program_record.program_id =
    program.id WHERE DATE_FORMAT(NOW(), '%H:%i%S') BETWEEN
   program_record.from AND program_record.to AND program_schedule
   .day = (DAYOFWEEK(NOW()) - 1) AND program_schedule.circuit_id
   = <id_okruhu> LIMIT 1)
ORDER by~regular ASC LIMIT 1
```

Kód 6.1: Optimalizovaný SQL dotaz pro získání naplánované hodnoty

Optimalizovaný SQL dotaz, používaný pro získání aktuální naplánované hodnoty, je uveden v kódu 6.1. Jedná se o poměrně složitý dotaz, který vznikl spojením dvou poddotazů. První poddotaz vybírá operativní změnu z tabulky `operative_change`, která je platná v čas dotazu. Druhý dotaz vybírá naplánovanou hodnotu podle programu, který je přiřazen k dotazovanému okruhu. V plánovači programů podle aktuálního dne v týdnu nalezne přiřazený program a v programu podle aktuálního času nalezne příslušný záznam. Výsledky dotazů jsou spojeny a omezeny na jeden nalezený výsledek, přičemž operativní změna má přednost před naplánovaným programem.

Pomocí SQL příkazu `EXPLAIN`, který zobrazí plán provádění příkazu, byla provedena optimalizace dotazu a databázové struktury tak, aby se při jeho provádění používaly vytvořené indexy namísto procházení celých tabulek. Výřez výstupu tohoto příkazu lze shlédnout v tabulce 6.1.

6.9 Práce s chybovými hláškami

K nakládání s chybovými, ale i provozními hláškami slouží třídy, které dědí od rozhraní `core::Logger`. Všechny třídy tohoto druhu se nacházejí v adresáři `src/core/logging`. Představené rozhraní obsahuje dvě metody `log`, které je nutné v konkrétní logovací třídě implementovat a které se od sebe liší pouze posledním parametrem. Prvním parametrem obou metod je závažnost hlášení. Závažnosti jsou definovány výčtem `core::Logger::Severity`, který obsahuje následující položky:

- DEBUG – ladící informace určené pro odborníky
- INFO – provozní informace, většinou úspěšného charakteru
- WARNING – chybové hlášení oznamující popis chyby, ze které je možné se zotavit
- ERROR – chybové hlášení závažného charakteru, většinou vede k ukončení řídicí aplikace

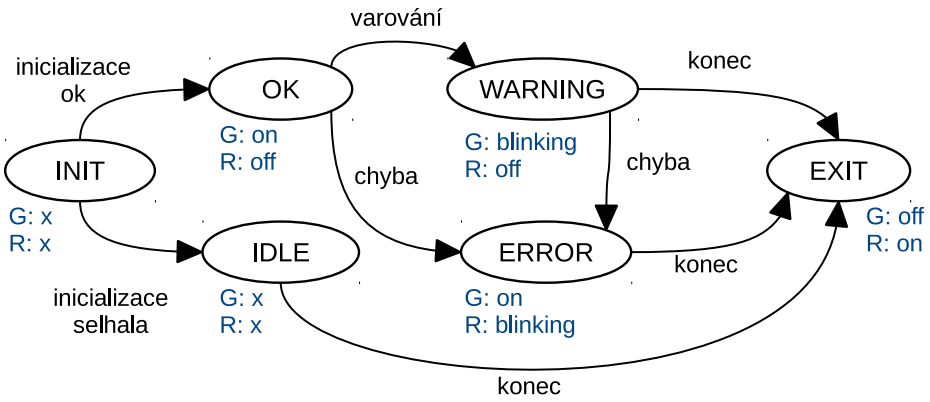
Druhým parametrem společným pro obě metody je krátký popisek chybového hlášení, resp. oznámení části aplikace, které se hlášení týká. Třetí parametr se u obou metod liší. V jedné je třetím parametrem textový řetězec s podrobným popisem vzniklé situace, ve druhé je parametrem reference na výjimku dědící od třídy `std::exception`. Tvorba popisku z výjimky záleží na konkrétní implementaci.

Rozhraní loggeru poskytuje určitou volnost v podobě možnosti ukládat hlášení na různá místa, na druhou stranu je na každou jeho implementaci kladen požadavek bezpečnosti vzhledem k provádění ve více vláknech. Rozhraní `core::Logger` je implementováno celkem čtyřikrát.

První implementací rozhraní je třída `core::LoggerHub`, která slouží pouze jako opakovač. Metodou `addLogger` lze přidat instance objektů implementující rozhraní loggeru. Při zavolání metody pro obsluhu hlášky je příslušná metoda delegována na všechny přidané loggery.

Druhou implementací je třída `core::FileLogger`, která ukládá chybové hlášky do souboru. Každý záznam začíná na novém řádku, lze ovšem nastavit oddělovač jednotlivých polí záznamu. Třetí implementace leží ve třídě `core::MySQLLogger`, která záznamy ukládá do databázové tabulky `protocol`.

Čtvrtá implementace leží ve třídě `core::LedStatusThread`. Jedná se o obálku, která je vykonávána v rámci jednoho vlákna a slouží pro ovládání LED diod na desce s budičem 1-Wire sběrnice. Díky implementaci rozhraní loggeru lze do vlákna asynchronně poslat informaci o chybovém stavu a vlákno pak může tuto chybu indikovat rozsvícením, případně blikáním LED diod. Třída implementuje jednoduchý konečný automat, kterým je indikace řízena a jeho stavy jsou měněny voláním funkcí `log`. Stavový automat je znázorněn na obrázku 6.3, jsou z něj patrné přechody na základě volání logovací metody s nastavenou důležitostí a také svit LED diod na modulu `bus-driver` v jednotlivých stavech.



Obrázek 6.3: Graf stavového automatu, který implementuje třída `LedStatusThread`

6.9.1 EventLogger

Třída `EventLogger` nemá na starost práci s chybovými hláškami, jde o nástroj pro trasování stavu ovladačů, do této sekce byl zařazen z důvodu podobnosti logovací třídě. Cílem této třídy je ukládat klíčové slovo stavu spolu se dvěma volitelnými parametry a aktuálním časem

do databáze k příslušnému okruhu, ze kterého je metoda uložení vyvolána. V objektu okruhu je předpřipravena metoda `event`, která vazbu na okruh zařídí, stačí ji zavolat s příslušnými hodnotami. V rámci aplikace je vytvořen pouze jeden objekt této třídy a ukazatel na něj je distribuován všem ovladačům, z toho důvodu bylo potřebné ošetřit souběžný přístup použitím mutexu.

Klíčová slova jsou specifická pro každý okruh a na konkrétní hlášku jsou překládána až v ovládací aplikaci. Díky dvěma parametrům lze předávat měnící se hodnoty. Parametry budou v ovládací aplikaci vloženy do textu díky tzv. placeholderům.

6.10 Konfigurace

Cílem konfigurace je poskytnout možnost upřesnit parametry pro běh ovladačů a okruhů. Konfigurace je uložena v databázových tabulkách `module_config` a `circuit_congfig` a je založena na principu klíč – hodnota, přičemž klíč je v rámci konfigurace jedné entity unikátní. Nastavování záznamů probíhá většinou z webového ovládacího systému a při jakékoliv její změně je odeslána zpráva s předmětem *config-changed* na rozhraní zpráv příslušného ovladače. Reakce na zprávu je kompletně v režii ovladače.

Jeden konfigurační záznam je reprezentován instancí třídy `core::Configuration`. Tuto instanci lze okamžitě použít k přiřazení či výpočtům, jelikož objekt lze explicitně přetypovat do jednoho z následujících datových typů: `string`, `int`, `double`, `bool` a `unsigned`. Pro získání řetězce s klíčem slouží metoda `getKey` a pro zjištění, zda klíčem je prázdný řetězec metoda `empty`.

K získávání konfigurace slouží třída `core::Configurator`, jejíž instance je nainicializována a připravena k použití v každém objektu okruhu a ovladače. Třída obsahuje velmi jednoduché rozhraní, skrz které lze s konfigurací dané entity manipulovat:

- `Configuration getConfig(std::string key)` – Vrací záznam konfigurace pro klíč zadaný parametrem. Pokud záznam s takovým klíčem v databázi neexistuje, je tato skutečnost indikována vyhozením výjimky `std::invalid_argument`.
- `void setConfig(Configuration c)` – Slouží pro uložení vytvořeného záznamu do konfigurace dané entity. Pokud záznam se stejným klíčem v konfiguraci již existuje, je přepsána pouze hodnota za novou.

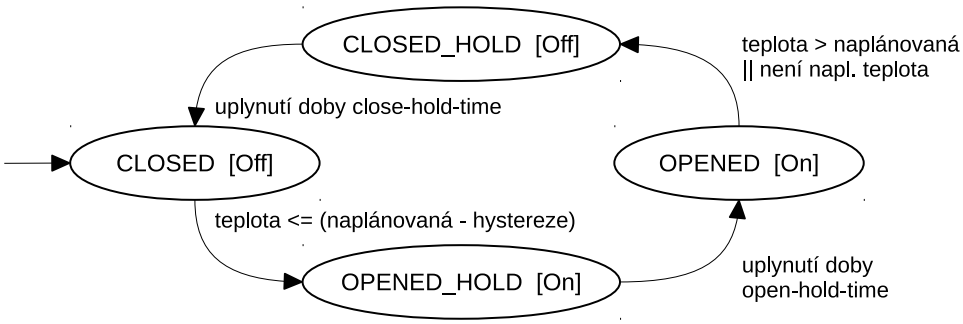
6.11 Modul pro ovládání radiátorů

Kromě návrhu automatizačního systému je součástí zadání práce i návrh řízení centrálního vytápění. Jelikož způsob regulace topné soustavy úzce souvisí s použitými komponentami, pro tuto práci budeme uvažovat klasické radiátory, na kterých je místo kohoutku nasazen termoelektrický pohon, který umožňuje dvoupolohovou regulaci. Jedná se o elektronickou hlavici, kterou lze přivedením napětí otevřít a jeho odebráním zavřít. Takovou hlavici lze řídit pomocí relé, pro její ovládání bude využit vyrobený modul s osmi relé.

Popisovanému termoelektrickému pohonu vyhovuje produkt MT4-024-NC [3] z nabídky firmy Honeywell. Tento pohon je ovládán napětím 24 V AC/DC. Termopohon je téměř neslyšitelný oproti hlavici s motorkem a převody, jelikož pohyb dříku je způsoben zahříváním voskového prvku PTC rezistorem. Tato vlastnost je však vykoupena průměrnou dobou úplného přestavení dříku, která trvá 4 minuty v případě 2,5 mm zdvihu a 6 minut v případě 6,5 mm zdvihu. Při srovnání s hlavicí obsahující motorek, které přestavení trvá do 10 vteřin,

se jedná o poměrně dlouhou dobu. Dle katalogového listu je odběr proudu při začátku přestavování (zhruba prvních 500 ms) menší než 0,2 A, trvalý proud je pak menší než 0,1 A.

Ovladač je implementován ve třídě `module::HeatingRadiatorModule`. Cílem modulu je obsluhovat všechny přiřazené okruhy pro řízení vytápění. Vstupem obsluhy jednoho okruhu je aktuální teplota v místnosti a naplánovaná teplota programem či operativní změnou. Výstupem je hodnota sepnuto/rozepnuto pro všechna relé ovládající hlavice topení přiřazené k danému okruhu. Obsluha tedy spočívá v získání vstupních hodnot, rozhodnutí se na základě vnitřního stavu a těchto hodnot o stavu následujícím a provedením nastavení výstupů. Ovladač tak implementuje jednoduchý stavový automat typu Moore, který na základě vstupních hodnot a vnitřního stavu rozhoduje o stavu následujícím. Výstup je funkcí jeho stavu a v grafu 6.4 je znázorněn v hranatých závorkách za názvem stavu.



Obrázek 6.4: Graf konečného stavového automatu pro řízení termopohonů radiátorů

Z grafu je patrné, že automat začíná ve stavu `CLOSED` a žádný stav není koncovým. Přechody mezi stavy tvoří smyčku, díky které lze plynule přecházet mezi stavy s otevřenými a zavřenými hlavicemi topení. Automat obsahuje celkem čtyři stavy, jejich význam je definován následovně:

- `CLOSED` – výchozí stav, radiátory zavřeny, čeká se na impulz pro otevření
- `OPENED_HOLD` – mezistav, který slouží pro nastavení minimální doby, po kterou budou radiátory otevřeny
- `OPENED` – radiátory otevřeny, čeká se na dosažení natápěné teploty nebo skončení naplánovaného programu
- `CLOSED_HOLD` – mezistav, který slouží pro nastavení minimální doby, po níž budou radiátory zavřeny

Hystereze je definována jako rozdíl mezi polohou zapnuto/vypnuto při dosažení požadované teploty. Příkladem může být modelová situace, kdy se natápí na 21 °C. V momentě dosažení požadované teploty je termoelektrická hlavice na topení zavřena. Znovu je otevřena, až teplota v místnosti klesne pod 20 °C. Hystereze může výrazně snížit počet sepnutí hlavic topení a tím pádem i počet sepnutí kotle za jednotku času, na druhou stranu příliš velká hystereze vede ke snížení teplotního komfortu. Je třeba najít optimální hodnotu mezi naznačenými dvěma extrémy, což je důvod možnosti nastavení hystereze v každém okruhu (místnosti) nezávisle.

Parametry ovladače

Parametry jsou myšleny klíče konfigurace, kterou je možné ovladači nastavit. Všechny parametry zobrazené v tabulce 6.2, kromě prvního, je možné nastavit do konfigurace ovladače i do konfigurace okruhu, přičemž platí, že konfigurace ovladače má vyšší prioritu. První parametr lze nastavit pouze do konfigurace ovladače, jelikož se vztahuje na cyklus celého vlákna. Parametry minimálního počtu cyklů byly zavedeny z důvodu různé doby otevírání hlavic. Doba otevření, resp. zavření by pak měla být o něco delší než doba potřebná pro přestavení hlavice. Nastavované parametry by měly reflektovat i parametry použitého kotle. Výpočet probíhá v cyklech a za dokončení každého cyklu je vsunuto časové zpoždění, než začne cyklus znovu. Vhodným nastavením počtu cyklů za jednotku času lze dosáhnout nižší výpočetní zátěže, naopak pokud je perioda příliš dlouhá, zvyšuje se i doba odezvy regulace.

Klíč	**Výchozí**	**Popis**
workcycle-period	10000	Perioda v milisekundách, se kterou se opakuje výpočetní cyklus všech okruhů
hysteresis	1.0	Hystereze ve °C
open-hold-time	6	Minimální počet cyklů, po které bude hlavice otevřená
close-hold-time	6	Minimální počet cyklů, po které bude hlavice zavřená

Tabulka 6.2: Přehled parametrů ovladače radiátorů

6.12 Modul pro ovládání kotle

Aby byla topná sestava kompletní a funkční, musí se v ní kromě radiátorů vyskytovat prvek, který vyrábí teplo a umožňuje cirkulaci vody v soustavě. Pro tuto práci budeme uvažovat plynový kotel, jehož výkon je automaticky regulován podle potřeby a který lze spínat opět pomocí relé. V sepnutém stavu kotel natápí na interně určenou teplotu a zapíná čerpadlo, které zajistí cirkulaci vody soustavou.

6.12.1 Definice požadavků

Požadavek na vytápění nastává v případě otevření alespoň jedné ze sledovaných hlavic topení. Po jejím otevření by měl kotel spustit svou aktivitu. Pokud je ovšem doba otevření hlavice delší, je vhodné, aby byla možnost oddálit spuštění kotle v čase, z důvodu, aby se nesnažil zbytečně cirkulovat vodu skrz zavřený systém (krátkou smyčku). Po oddálení se kotel spíná a začíná natápět. V momentě, kdy již není požadavek na vytápění (všechny sledované hlavice jsou zavřené), by měl kotel také skončit svou aktivitu. Pokud je doba zavření delší, může být vyžadováno zpožděné vypnutí za definovaný časový interval. Po skončení aktivity by měl kotel určitou dobu setrvat ve stavu klidu před zahájením dalšího aktivního cyklu. Tento parametr se liší v závislosti na použitém kotli a otopné soustavě. Po uplynutí klidového času se opět čeká na otevření alespoň jedné ze sledovaných hlavic, aby cyklus mohl začít znovu. Pro dodržení provozních parametrů kotle je vhodné mít možnost omezit počet sepnutí kotle za jednotku času.

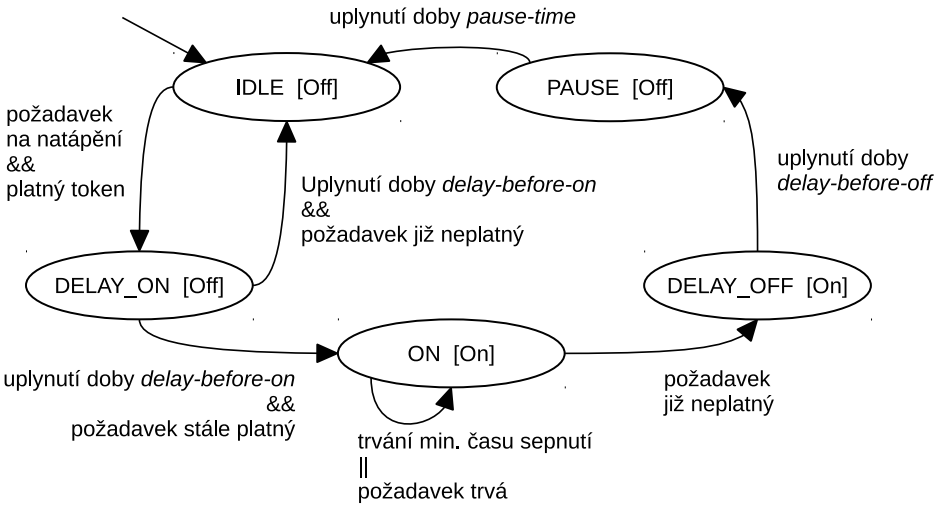
6.12.2 Návrh ovladače

Ovladač pro řízení okruhů, ve kterých jsou umístěny kotle, je implementován ve třídě `module::HeatingBoilerModule`. Zodpovědností tohoto ovladače je řídit jeden či více kotlů – podle počtu přiřazených okruhů ovladači. V rámci okruhu jsou definovány dva sokety pro připojení zařízení:

- `temp_head` – všechny hlavice, na jejichž základě se bude kotel spínat. Hlavice jsou typu `relay` a socket slouží pouze pro čtení.
- `boiler_relay` – jedno či více relé řídící činnost kotle (kotlů). Soket je virtuálního typu `relay` a je aktivně řízen.

Náplní ovladače je v pravidelném cyklu provádět výpočet nad všemi přiřazenými okruhy pro řízení kotle a na základě přijaté zprávy o změně konfigurace ji přepočítat. Na konec tohoto cyklu je zařazeno krátké zpoždění před spuštěním dalšího cyklu, což umožňuje šetřit výpočetní čas stroje.

Každý okruh je řízen svým stavovým automatem typu Moore, jehož výstupem je povel pro kotel zapnuto/vypnuto a který zahrnuje chování popsané v definici požadavků. Graf tohoto konečného automatu je zobrazen na obrázku 6.5. Vstupem automatu je údaj o otevření alespoň jedné z přiřazených hlavic topení, ty jsou řízeny ovladačem pro ovládání radiátorů a v tomto ovladači se provádí pouze čtení jejich stavu. Na základě tohoto vstupu a aktuálního stavu je vypočítán nový stav a řízen výstup. Z toho důvodu je každý okruh uzavřen v obálce (objektu) třídy `module::HeatingBoilerModule::CircuitWrapper`, který zapouzdřuje aktuální stav, nastíněný automat a konfiguraci daného okruhu. Přepočítání stavu provádí ovladač invokací metody `work` nad touto obálkou. Vypočtení a nastavení konfigurace je taktéž zodpovědností ovladače a provádí se settery s názvy příslušných klíčů konfigurací.



Obrázek 6.5: Graf konečného stavového automatu pro řízení kotle

Požadavek na omezení počtu sepnutí kotle v rámci časové jednotky je v ovladači vztažen k jedné hodině a je možné jej měnit nastavováním konfigurace. V rámci obálky okruhu, která

udržuje jeho stav, se nachází počítadlo dostupných tokenů, které je každou celou hodinu doplňováno do maximálního počtu, zadaného v konfiguraci. Při požadavku na natápění ve stavu `IDLE` je při průchodu do zpožďovacího stavu `DELAY_ON` odebrán jeden token. Pokud je počítadlo tokenů prázdné a požadavek stále trvá, setrvává se ve stavu `IDLE` do doby doplnění tokenů. V případě, že po zpožďovacím stavu je zjištěno, že požadavek již není platný, token je vrácen zpátky do počítadla a přechází se do stavu `IDLE`.

Parametry ovladače

Všechny nastavitelné parametry ovladače jsou uvedeny v tabulce 6.3. Pokud se záznam týká času (všechny vyjma prvního), předpokládanou jednotkou jsou sekundy. Zobrazené parametry je možné nastavit jak ovladači, tak i jednotlivým okruhům. Nastavení parametrů ovladači má globální dopad na všechny okruhy, pokud nemají stejný klíč nastaven v rámci konfigurace, která má vyšší prioritu. Výchozí hodnoty není nutné zadávat, jsou vytvořeny automaticky při neúspěšném pokusu o získání konfigurace.

Klíč	**Výchozí**	**Popis**
max-cycles-perhour	6	Maximální počet cyklů sepnutí–rozepnutí v rámci jedné hodiny
min-switching-time	6	Minimální čas, po který bude kotel sepnut
pause-time	6	Minimální čas, po který bude kotel v klidovém stavu, než jej bude opět možné sepnout
delay-before-on	0	Zpoždění před sepnutím kotle po obdržení požadavku na natápění
delay-before-off	0	Zpoždění před rozepnutím kotle po zaniknutí požadavku na natápění

Tabulka 6.3: Přehled parametrů ovladače kotle (kotlů)

V rámci uspokojivé regulace je nutné sladit parametry dvou popsaných ovladačů, které mají za vytápění zodpovědnost. Tyto parametry jsou určeny použitým otopným systémem, hlavicemi, které otevírají ventily radiátorů a kotle. Je nutné dodržet zejména časování doby otevření hlavice a doby zpoždění sepnutí kotle, aby nedocházelo k čerpání vody do zavřeného systému. Podobná situace nastává i při zavírání hlavic topení. Dalším důležitým parametrem je hystereze, která významně ovlivňuje počet cyklů otevření hlavic a sepnutí kotle. Tento parametr se odvíjí především od velikosti místnosti, použitého radiátoru a schopnosti stěn akumulovat teplo.

Systémem je možné řídit i více nezávislých topných okruhů v jednom objektu. Velmi důležité je pak správné propojení hlavic v soketech jednotlivých ovladačů, aby nedošlo k překřížení dvou nezávislých topných okruhů. Zejména se jedná o sokety, na kterých ovladač kotlů sleduje aktivitu hlavic topení.

6.13 Modul pro ovládání relé

V rámci vhodné demonstrace systému byl navržen ovladač pro spínání relé, který může být využit v celé škále aplikací, ve kterých je koncové zařízení ovládáno dvěma stavy (zapnuto/vypnuto). Do této škály lze zařadit například jednoduché osvětlení, otevírání dveří či spínání zásuvek.

Ovládání relé by pak mohlo fungovat ve dvou režimech. První režim je automatický, kdy si uživatel připraví programy, ve kterých bude nadefinováno, v kolik hodin má relé sepnout a v kolik rozepnout. Tyto programy poté přiřadí k jednotlivým dnům v týdnu. Ve druhém režimu relé funguje na základě přímého uživatelova podnětu na sepnutí, resp. rozepnutí. Tento režim je nazýván manuální. Oba režimy je možné kombinovat a mít tak po většinu času relé ovládáno automaticky a pouze v případě potřeby změny jej přepnout na manuální režim.

Manuální režim je řešen přes konfiguraci okruhu, což plně vyhovuje specifikaci z hlediska rychlosti – při změně konfigurace je tato skutečnost oznámena zprávou, pokud má rozhraní zpráv tento typ povolen. Dalším hlediskem je perzistentnost hodnoty a tudíž po restartu systému se stav okruhu sám vrátí do stavu před restartem. Zprávy o změně konfigurace jsou periodicky odchytávány každých 125 ms.

Popsaný ovladač je implementován ve třídě `module::GPRelayModule`. Pro každý okruh, který je připojen k tomuto ovladači, je vytvořena vlastní obálka – instance třídy `module::GPRelayModule::CircuitContext`, která nese zodpovědnost za uchování a změny stavu. Metodou `update` ovladač vyzývá obálku, aby přepočítala svůj stav a řídila koncové relé. V rámci updatu se rozlišují dvě hloubky, které jsou řízeny parametrem metody. První hloubkou je běžný update, který získává naplánovanou hodnotu programu a podle toho přepočítá stav. Tento režim obnovy může ovlivnit pouze okruh v automatickém režimu a je volán každou minutu. Druhou hloubkou je kompletní update, při kterém se zohledňuje i manuální režim, což s sebou nese podstatně větší zátěž na databázi a z toho důvodu je vykonáván každých 5 minut. Pokud přijde zpráva o změně stavu konfigurace okruhu, je volána právě tato hloubka obnovy, která zajišťuje rychlou reakci.

Parametry ovladače

Pro přímočarost ovladače není parametrů mnoho. Všechny použitelné parametry, včetně výchozích hodnot, jsou uvedeny v tabulce 6.3. Jedná se pouze o parametry, které ovlivňují manuální mód a jeho stav. Hodnoty obou parametrů jsou typu boolean a proto je třeba je vhodně nastavit. Bezpečné hodnoty, které jsou mimo jiné používané i ovládací aplikací, jsou „0“ pro vypnuto a „1“ pro zapnuto.

Klíč	**Výchozí**	**Popis**
manual-mode	0	Ovládání manuálního módu.
manual-mode-value	0	Hodnota, která má být nastavena v manuálním módu

Tabulka 6.4: Přehled parametrů ovladače obecného relé

6.14 Modul pro ovládání displeje

Z předchozích kapitol máme navrhnuto připojení displeje k systému. Smyslem displeje je zobrazovat aktuální hodnoty zařízení v systému. Zobrazovat se budou pouze ta zařízení, která si uživatel zvolí v ovládací aplikaci.

Displej se v aplikaci tváří jako aktuátor, potíž je v tom, že standardnímu zařízení lze nastavovat a získávat pouze hodnoty typu double. Zařízení displeje je ovšem implementováno ve své třídě dědící od abstraktní třídy pro zařízení a tak je možné přidat metody pro práci

s řetězci a rozšířit tak standardní rozhraní, které pracuje pouze s hodnotami double. Jelikož je známo, na kterém místě se bude vyskytovat objekt této třídy (ve kterém soketu), je možné jej staticky přetypovat a používat jeho rozšířené rozhraní. Pro funkčnost je nutné pracovat s přetypovanou třídou v ovladači i v technologii, která řetězec nastaví fyzickému displeji.

Navrhovaný ovladač je umístěn ve třídě `module::LcdDisplayModule`. Jeho cílem je periodicky obnovovat text na displeji a reagovat na změnu konfigurace. Každý okruh s displejem používá třídu `module::DisplayState` k uchování kontextu zobrazování, z důvodu, aby v příštím vykreslovacím cyklu bylo možné pokračovat v místě konce současného cyklu. Stavový objekt definuje výřez, který bude na displej zobrazen a umožňuje tak stránkovat zařízení, jejichž stav se má zobrazovat, pokud je jich více, než je řádků displeje. S každým vykreslovacím cyklem je tento výřez posunut. Ovladač obsahuje dva sokety, na které je možno připojit zařízení:

- `devices_to_show` – přípojný bod pro všechna zařízení, jejichž stav je potřeba zobrazit
- `display` – přípojný bod pro lcd displej

Do prvního soketu lze v rámci ovládací aplikace připojit libovolné zařízení, jelikož jako typ hodnoty používá hodnotu „all“. Ovladač používá standardní rozhraní pro zařízení, ze kterého získá jméno zařízení, které v případě potřeby zkrátí na požadovaný počet znaků. Dále používá metodu `getFormattedValue`, která vrací řetězec s naformátovanou hodnotou konkrétního zařízení. Ve výchozím nastavení je na prvním řádku displeje zobrazen aktuální datum a čas.

Druhý soket slouží pro připojení zařízení displeje. Do tohoto soketu je možné připojit pouze jedno zařízení, jelikož při připojení více zařízení by nebylo jednoznačné, jak naložit s displeji, které mají různé rozměry. Pokud by byla potřeba vypisovat na dva displeje stejné informace, lze toho dosáhnout vytvořením samostatného okruhu pro každý displej a přiřazením stejných zařízení k zobrazení.

Parametry ovladače

Všechny parametry ovladače jsou uvedeny v tabulce 6.5. Pro dolazení intervalu překreslování slouží parametr `refresh-rate`. Příliš nízká hodnota tohoto parametru by mohla značně zvýšit zátěž komunikačního kanálu v konkrétní technologii, proto je omezena spodní hranící 2000 ms. Zbylé dva parametry jsou typu boolean a určují, zda zobrazovat hodiny na prvním řádku displeje a povolení displeje k zobrazování. Při vypnutém displeji je obsah displeje smazán a čeká se na signál k povolení. Tyto parametry je možné nastavit do konfigurace ovladače, kde budou mít globální význam pro všechny okruhy. Taktéž je možné je nastavit do konfigurace okruhu a přepsat tak v rámci okruhu globální nastavení.

Klíč	**Výchozí**	**Popis**
refresh-rate	10000	Interval pro překreslování v milisekundách
display-time	1	Povolení zobrazování hodin na prvním řádku
disabled	0	Zakázání činnosti displeje

Tabulka 6.5: Přehled parametrů ovladače displeje

Kapitola 7

Návrh a implementace ovládací aplikace

V této kapitole bude popsán návrh a implementace webové aplikace pro ovládání systému. Cílem této aplikace je poskytnout uživateli nástroj, skrze který bude mít možnost monitorovat a měnit nastavení systému interaktivní grafickou cestou.

7.1 Cílové požadavky na aplikaci a její rozhraní

Cílem uživatelského rozhraní této aplikace je prezentovat uživateli nasbíraná data v co nejsrozumitelnější podobě tak, aby se na jejich základě mohl rozhodovat o dalším plánování budoucích akcí. Plánováním akcí se rozumí vytváření programů a jejich přiřazení k určitému dnu v týdnu do plánovače vybraného okruhu či manuální konfigurace za pomocí registru konfigurace. Kvůli velkému množství naměřených dat ze zařízení bude nutné tato data nejprve agregovat a poté zobrazit v přehledném grafu. Taktéž nastavování programu by se mělo odehrávat grafickou formou namísto zápisu hodnot do tabulky.

Aplikace by dále měla nabízet rozhraní pro vytváření základních entit, jako jsou místnosti a okruhy v nich umístěné. Další požadovanou funkčností je správa připojených zařízení, která zahrnuje jejich vyhledání, přidání do systému, úpravy, odstranění či přiřazení do soketů vybraných okruhů. Žádoucí je též správa uživatelských účtů a možnost zařazení uživatele do určité role, podle které by získal oprávnění pracovat s určitou částí aplikace. Jelikož se jedná o rozšířitelnou aplikaci je žádoucí poskytnout rozhraní (zejména programátorské) pro budoucí rozšíření o nové technologie, ovladače a zařízení.

Posledním důležitým aspektem je nezávislost rozhraní na použitém zařízení a operačním systému, který zařízení používá. Aplikace by měla být responsivní a přizpůsobit se šířce displeje na zobrazovaném zařízení tak, aby bylo možné systém ovládat pohodlně z telefonu, tabletu či osobního počítače.

7.2 Definice skupin uživatelů

Cílovou skupinou jsou především technicky zdatnější uživatelé, kteří využijí potenciálu, který systém nabízí a vytvoří si v případě specifických potřeb vlastní modul, případně využijí některý z hotových modulů. Druhou cílovou skupinu tvoří uživatelé, kteří si systém nechají nainstalovat od technika a budou s ním komunikovat pouze přes navrhované rozhraní. Obě skupiny spojuje zájem o automatizaci konkrétních činností v domácích podmínkách.

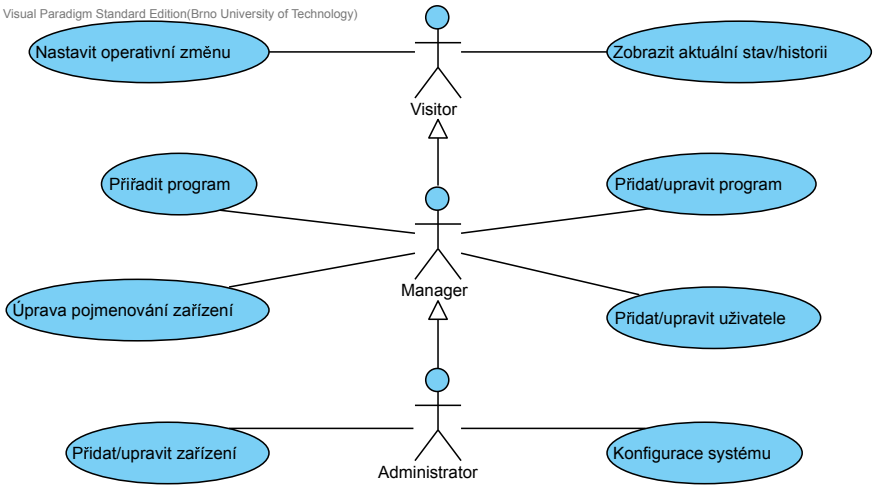
Cílová skupina uživatelů není limitována věkem, ovšem z různých důvodů může být žádoucí omezit práva určitých uživatelů v systému tak, aby nemohlo dojít k poškození systému nesprávným nastavením či znehodnocení připojených zařízení. Z toho důvodu je vhodné zavést uživatelské role. V rámci této práce budou zavedeny tři role pro uživatele:

- *Visitor* – Uživatel s touto rolí může přistupovat k systému způsobem, ve kterém nelze změnit žádné z jeho nastavení. Jedná se tak o uživatele, který bude mít možnost monitorovat aktuální dění na vybraných zařízeních a do jejich výkonu zasáhnout maximálně vytvořením operativní změny, která může změnit průběh pouze dočasně.
- *Manager* – Uživatel s běžným oprávněním, který může monitorovat systém podobně jako uživatel s předchozí rolí, ovšem navíc má možnost aktivně zasahovat do dění systému vytvářením nebo editací programu a jeho následným přiřazením k určitému okruhu a dnu v týdnu. Uživatel s tímto oprávněním má dále právo vytvářet nové místnosti, okruhy a umísťovat do nich již nalezená zařízení.
- *Administrátor* – Administrátor může vykonávat veškeré činnosti jako jeho předchůdci a navíc může zasahovat do nastavení systému, spravovat připojená zařízení a měnit konfiguraci ovladačů a okruhů.

Nový uživatel může být vytvořen uživatelem s běžným či administrátorským oprávněním, a to tak, že vytvářený uživatel nemůže mít vyšší oprávnění než ten, který jej vytváří.

Diagram případů užití je zobrazen na obrázku 7.1. Z hlediska nejčastějšího použití lze rozlišit dva případy užití. V prvním má uživatel okamžitou potřebu reagovat na současnou situaci. Pokud tento případ vztáhneme na vytápění, lze říci, že uživateli je zima a chce si zatopit. Řešením této situace je nastavit operativní změnu, která se aplikuje na daný okruh namísto běžně nastaveného programu. Tento případ užití je možné provést i s nejnižším oprávněním.

Druhým běžným případem užití je monitorování aktuálního stavu systému. Je možné prohlížet si historii ze senzorů a aktuátorů v porovnání s naplánovanými hodnotami. S tímto souvisí i další bod případu užití – plánování programů. Pro tento případ užití vyžaduje mít alespoň uživatelskou roli *manager*.

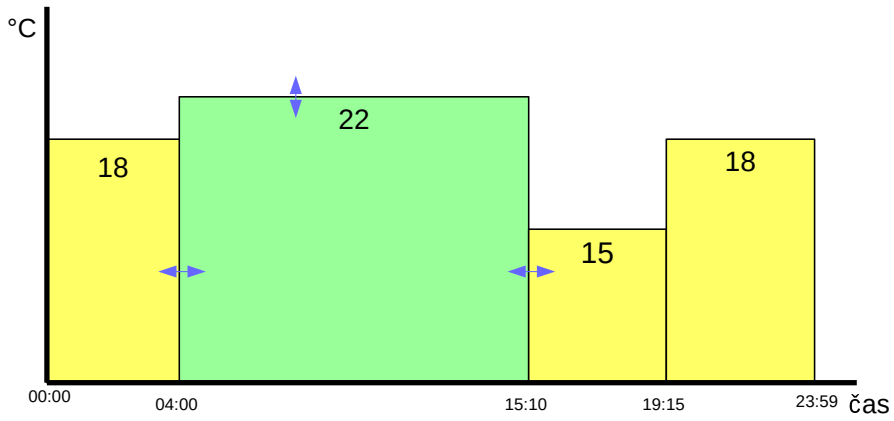


Obrázek 7.1: Diagram případů užití

7.3 Návrh GUI

Grafické uživatelské rozhraní by mělo uživateli zpřístupnit úvodní přehledovou stránku, na které nalezne informace o stavu systému a varování v případě potíží. Z úvodní stránky bude možné přejít na přehled místností, který bude rozdělený podle pater v domě, nebo vstoupit přímo do místnosti z menu. Při přechodu na stránku místnosti bude zobrazen přehled všech vytvořených okruhů vždy s krátkou informací o jejich stavu. Okruhy bude možné na přehledové stránce do místnosti přidávat. Jiným typem přehledu by měl být přehled na základě patra. Uživatel bude mít možnost při vytváření nebo úpravě patra nahrát jeho nákres (například půdorys) a na něj umístit zařízení, která se v daném patře nacházejí. Na nákresu pak bude možné sledovat stav umístěných zařízení.

Po přechodu na detail konkrétního okruhu bude zobrazen jeho aktuální stav, možnosti vytvoření operativní změny, nastavení programu a graf s historií, který bude možné filtrovat podle data. Ze stránky okruhu bude možné přejít na stranu s nastavením okruhu, kde bude možné změnit vše od názvu přes připojení zařízení do jednotlivých soketů až po jeho konfiguraci. Z detailní stránky okruhu bude možné přejít na vytvoření nového programu pro vybraný okruh. Po vytvoření nového programu jej bude možné přiřadit v rozvrhu k některému ze dnů v týdnu. Pro změnu programu bude možné přejít z menu k přehledu programů a zde vytvořený program najít a upravit. Programem se rozumí posloupnost hodnot, které budou rozloženy v rámci jednoho dne. Nastavení programu by mělo probíhat grafickou cestou – prací s interaktivním grafem, který půjde rozdělovat v časové rovině a měnit hodnoty tažením v rovině hodnot. Návrh takového ovládání je zobrazen na obrázku 7.2.



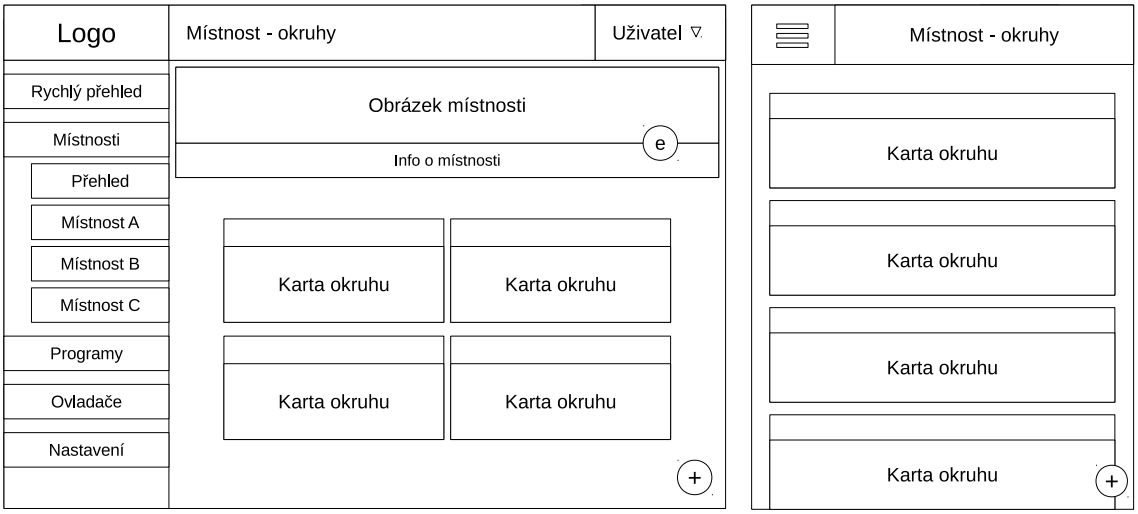
Obrázek 7.2: Wireframe rozhraní pro nastavování programu

Z menu bude možné přejít do nastavení systému, ve kterém bude možnost spravovat uživatelské účty, importovat definice uživatelských entit nebo spravovat zařízení připojená k systému. Poslední položkou v menu jsou ovladače, kde je možné zobrazit všechny dostupné ovladače a upravovat jejich konfiguraci, která má globální dopad na všechny okruhy.

Jedním z hlavních požadavků aplikace je možnost jejího provozu na různých zařízeních. Pro realizaci grafického uživatelského prostředí budou použity webové technologie, jejichž podporu lze nalézt v každém moderním zařízení, ve kterém by mohla být aplikace zobrazována. Grafické uživatelské prostředí bude využívat zásad Material Designu, navrže-

ného společností Google, pro sjednocení grafické stránky rozhraní s použitými rozhraními v moderních operačních systémech. Pro implementaci bude využito front-end frameworku MaterializeCss, který dodržuje zásady Material Designu a poskytuje komponenty pro sestavení stránky. S použitím různých zařízení souvisí i proměnné rozměry displejů, na kterých bude aplikace zobrazována a jejichž šířkám se bude muset přizpůsobit. Návrh zobrazení aplikace v závislosti na šířce displeje je zobrazen na obrázku 7.3.

Vykreslování grafů bude probíhat na klientské straně na základě Javascriptu s pomocí knihoven k tomu určených. Pro vykreslování větších grafů bude použita knihovna `Flot`, pro menší bude použita knihovna `Chart.js`.



(a) Počítač (b) Mobilní zařízení

Obrázek 7.3: Wireframe návrhu GUI – strana detailu místnosti v závislosti na použité šířce displeje

7.4 Návrh backendu

V předchozí kapitole byly pro realizaci zvoleny webové technologie, které pro zprostředkování aplikace uživateli vyžadují použití webového serveru. V rámci této práce byl použit server Apache2, který je dostupný v repozitářích cílové platformy. Jelikož se jedná o dynamickou webovou aplikaci, bude nutné použít skriptovací jazyk pro doplňování dynamického obsahu. V rámci práce je použit jazyk PHP, jehož interpret je nainstalován ve webovém serveru.

V rámci zjednodušení implementace bude vhodné použít existující back-end framework. Vhodným kandidátem je open-source MVC framework Nette[1], který nabízí spoustu užitečných vlastností v čele s podporou objektově orientovaného programování, které je základem rozšiřitelnosti této aplikace. Framework navíc nabízí propracovaný šablonovací systém a systém komponent, které budou základem pro zobrazování rozšiřitelných částí.

[1]Nette Framework: https://nette.org/

7.4.1 Úložiště dat

Důležitou částí backendu je perzistentní úložiště dat, které uchovává stav systému. V rámci této práce byla použita relační databáze MySQL. ER diagram vytvořené databáze je zobrazen v příloze C. V diagramu lze nalézt základní entitní množiny popisované v kapitole 4.2, rozšířené o potřebné atributy. Dále přibyly entitní množiny pro vytváření programů a jejich naplánování k určitému okruhu na určitý den v týdnu.

Z důvodu univerzálnosti systému se v databázovém schématu nachází tabulka `value_type`, jejímž cílem je odlišit interpretované hodnoty, aby bylo možné plánovat okruhu pouze ty programy, které vrací stejný typ hodnoty. Typem hodnoty je myšlen význam naplánované hodnoty – teplota, stav vypnuto/zapnuto atd. Tato hodnota je udána u každého zařízení, programu, předka soketu a ovladače. V rámci programu a zařízení se jedná o označení interpretace vrácené hodnoty. U předka soketu se jedná o typ zařízení, které je možné do konkrétního soketu připojit. Typ hodnoty u ovladače určuje množinu programů, které je možné naplánovat (program musí vracet stejný typ virtuální hodnoty). Entitní množina operativní změny se váže přímo na okruh, a proto není nutné rozlišovat interpretaci vrácené hodnoty.

Další entitní množinou je historie zařízení, která je do databáze ukládána z běžící řídicí služby. Přibyly tabulky pro konfiguraci ovladačů a okruhů a tabulky pro uchování záznamů o uživatelích, jejich rolí, včetně informací pro autorizátor, který povoluje uživateli vykonat určitou akci. Entitní množina `protocol` slouží k ukládání záznamů z řídicí služby, jedná se především o běhové informace a chybové hlášky. V entitní množině `event_log` jsou uchovávány záznamy o aktivitě ovladače v rámci okruhu.

Pro práci s databází bude použita vrstva Nextras\Orm[2], která zajišťuje mapování databázových entit na objekty. V rámci použitého frameworku bude zastupovat modelovou vrstvu. Strukturu objektů modelové vrstvy bude nutné vytvořit pomocí dědičnosti. Pro správné mapování je nutné vytvořit pro každou entitní množinu třídu, která reprezentuje repozitář, třídu reprezentující mezivrstvu směrem k databázi a třídu, jejíž instance bude reprezentovat jednu entitu.

7.5 Implementace ovládací aplikace

Implementace se odvíjí od použitého back-end frameworku. Pro zobrazení rozšiřitelných částí bylo využito komponent, které framework nabízí. Důležitou částí je i šablonovací systém *Latte*, který je součástí *Nette* a který umožňuje oddělit hlavní šablonu od šablon jednotlivých stránek. Podobně je tomu i u komponent, kde má také každá komponenta vlastní šablonu. O složení šablon dohromady se stará framework při vykreslování. V rámci této kapitoly nebudou popisovány části týkající se samotného nastavení frameworku a vytváření jednotlivých stránek. Kapitola by měla poskytnout přehled o funkci částí postavených nad frameworkem se zaměřením na rozhraní rozšiřitelných částí.

7.5.1 Ovladač

Zobrazení detailní stránky okruhu se odvíjí od použitého ovladače, kterým je daný okruh řízen. Jelikož každý ovladač má svá specifika a zaměřuje se na jinou automatizační činnost, nelze vytvořit univerzální stránku, která by sloužila pro zobrazení všech okruhů. Na druhou stranu by bylo vhodné, aby se výsledná url adresa lišila pouze v identifikátoru zobrazovaného

[2] Nextras\Orm https://nextras.org/orm/docs/2.0/

okruhu a nezávisela na použitém ovladači. Dalším požadavkem je vytvořit náhledovou kartu okruhu, která bude zobrazena v detailu místnosti a jejíž obsah se bude také lišit s použitým ovladačem. Zároveň by bylo vhodné, aby se při budoucím rozšíření nemuselo zasahovat do struktury použitého frameworku.

Řešením této situace je vytvořit komponentu pro každý ovladač. Komponenta v Nette Frameworku představuje vykreslitelný objekt, který lze dále rozšiřovat. V rámci komponenty pak lze používat jiné komponenty, vykreslovat vlastní šablonu či definovat vlastní signály, které jsou v konečném důsledku odkazy. Každá komponenta v použitém frameworku je potomek třídy `Nette\Application\UI\Control` a její vykreslení se spouští metodou `render`. Vykreslovacích metod může být více a dají se rozlišovat v rámci šablonovacího systému, který framework používá.

Pro zjednodušení rozšiřování aplikace o komponenty nových ovladačů byl vytvořen jednotný přístup k tvorbě jejich komponent. Třídy těchto komponent jsou umístěny v adresáři `app/modules` a jejich šablony v podadresáři `templates`. Všechny tyto třídy sdílí společný jmenný prostor `App\Modules` a jejich název odpovídá unikátnímu názvu, který je u každého ovladače uveden v databázi v tabulce `module`. V rámci zjednodušení implementace byla předpřipravena abstraktní třída `ModuleComponent` ve stejném jmenném prostoru, která dědí od zmiňované třídy `Control` a obsahuje předpřipravené atributy a metody, včetně dvou metod, které je nutné v konkrétní třídě ovladače implementovat:

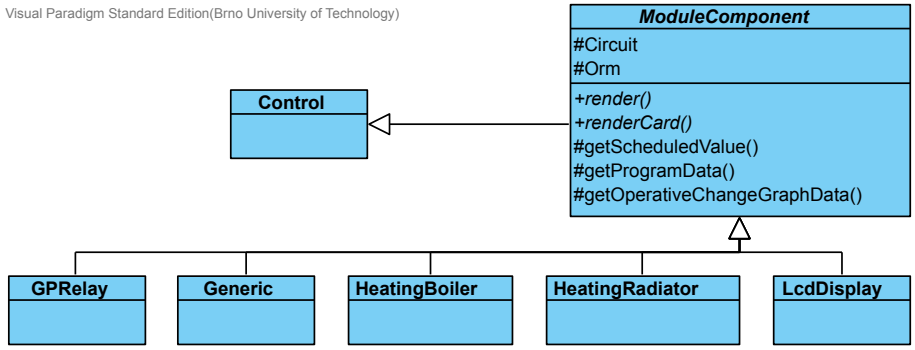
- `render()` – vykreslí komponentu, které bude věnována celá stránka detailu okruhu. Komponenta by měla zobrazovat aktuální stav příslušného okruhu, možnosti jeho změny a historii, ideálně porovnání naplánovaných hodnot se skutečnými.
- `renderCard()` – vykreslí obsah karty, která bude zobrazena na detailní stránce místnosti. Karta by měla obsahovat název okruhu a krátký detail o aktuálním stavu okruhu tak, aby například v případě pouhého zjištění teploty v místnosti nebylo nutné otevírat detail okruhu, který teplotu měří.

Nová komponenta ovladače musí dědit právě od této abstraktní třídy, aby mohla být použita. Abstraktní třída zpřístupňuje komponentě objekt okruhu z modelové vrstvy, jehož detail má být vykreslen, přístup k modelové vrstvě a přístup ke konfiguraci okruhu. Nabízí také metody pro získání naplánované hodnoty nebo získání dat k vykreslení naplánovaného programu či operativních změn do grafu. Každá z vykreslovacích metod má svou šablonu, ze které bude vykreslovat, umístěnou ve výše zmíněném adresáři pro šablony. Diagram tříd, popisující dědičnost komponent ovladačů, je zobrazen na obrázku 7.4.

Pokud je využito ukládání klíčových slov o stavu okruhu, je v ovladači nutné pro správné zobrazení naplnit asociativní pole `eventLogTranslation`, které definuje abstraktní třída, z níž se dědí. Toto pole je využito při překladu klíčového slova znamenajícího stav okruhu na srozumitelnou textovou hlášku. Klíčem do pole je tedy konkrétní klíčové slovo a hodnotou je srozumitelná textová hláška, která může využívat tzv. placeholdery. Jedná se o místa, kam budou při překladu hlášky vloženy argumenty. Nahrazovací placeholdery mají tvar *%1* nebo *%2*. Překlad je prováděn v rámci abstraktní třídy předka každého ovladače, takže jej není nutné implementovat v každém ovladači zvlášť a stačí pouze naplnit překladové asociativní pole.

Dodržení výše uvedeného přístupu zaručuje správné vytvoření komponenty. Komponenty jsou vytvářeny továrnou `App\Services\Factory\ModuleFactory` dynamicky podle názvu ovladače. Pokud komponentu ovladače nelze vytvořit, je vytvořena obecná komponenta `Generic`, která ovšem nemůže poskytnout bližší informace o stavu, jelikož nezná účel

daného ovladače a nemá z čeho odvodit způsob, jak interpretovat hodnoty zařízení v soketech.



Obrázek 7.4: Diagram tříd popisující dědičnost komponent ovladačů

7.5.2 Technologie

Technologie je další z rozšiřitelných částí systému. V ovládacím systému je po ní požadován jediný úkon, kterým je převod nalezeného zařízení, které je uloženo v databázi s příznakem `found_only`, na skutečná zařízení. Nalezené zařízení totiž může být interpretováno různými způsoby – použito v různých zapojeních. Příkladem může být nalezené zařízení `DS2408` v používané technologii 1-Wire. Ze zařízení nelze poznat, co se skrývá za ním, jelikož ani konkrétní technologie tento krok neumí učinit. Je proto nutné tento krok vykonat ručně a zvolit správnou z nabízených variant.

Jelikož jsou instance objektů konkrétních technologií opět vytvářeny dynamicky, je nutné dodržovat následující konvence. Každá technologie je umístěna ve své třídě, jejíž název je shodný s jednoznačným názvem uvedeným v databázi. Třídy technologií sdílí jmenný prostor `App\Technologies`. Zároveň dědí od abstraktní třídy `Technology` umístěné ve stejném jmenném prostoru a adresáři.

V abstraktní třídě jsou již vytvořeny mechanizmy, pomocí kterých lze získat seznam možných variant pro nalezené zařízení a vybranou variantu realizovat. V třídě konkrétní technologie je nutné pouze vytvořit všechny varianty a registrovat je pomocí připravené metody. Jednu variantu reprezentuje objekt, který je potomkem třídy `Variant`. Varianta má své jméno, popisek a metodu `perform` pro provedení transformace nalezeného zařízení na pole nových zařízení, kterou je nutné v každé třídě reprezentující konkrétní variantu implementovat. Tato metoda získá jako parametr objekt s nalezeným zařízením z modelové vrstvy a přemění jej na pole objektů třídy `NewDevice`. Zmiňovaná třída slouží pouze jako obálka, jelikož technologie nemá přístup k databázi, atributy ovšem zůstávají stejné jako u objektu zařízení z modelové vrstvy. O samotné odstranění nalezeného zařízení a vložení nových zařízení do databáze se stará metoda `performVariant` ze třídy `Technology`. Registrace varianty ke konkrétní technologii se provádí metodou `addVariant` v konstruktoru, prvním parametrem je typ zařízení, který je zapsán do databáze při jeho nalezení (sloupec `dev_type`) a který jednoznačně vymezuje seznam možných variant. Druhým parametrem je objekt reprezentující konkrétní variantu.

Proces přidání nových zařízení se skládá z několika kroků. V prvním kroku je nutné

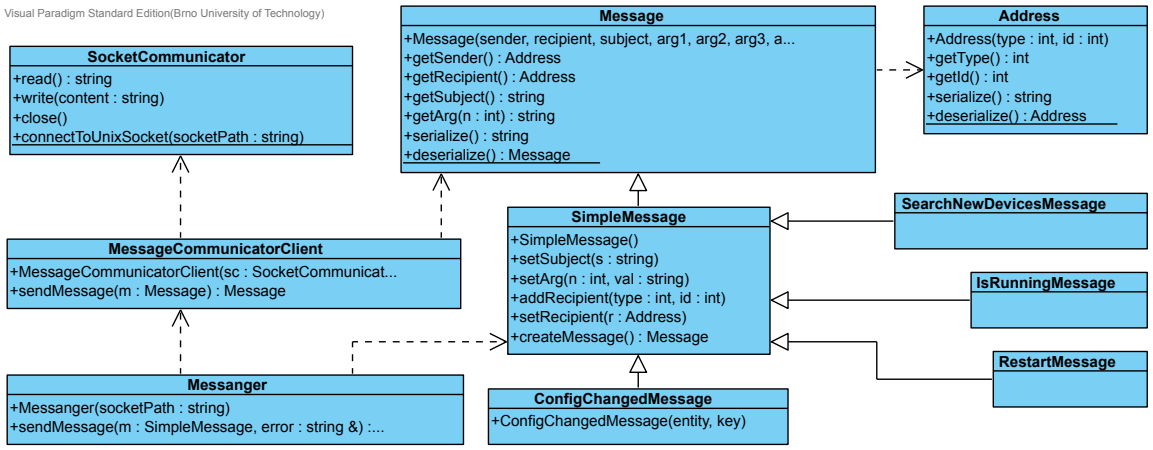
přidávané zařízení fyzicky připojit k systému. Dalším krokem je spuštění hledání zařízení v ovládací aplikaci, které lze nalézt v menu nastavení → zařízení → nalezená zařízení. Pokyn k vyhledání nových zařízení ve všech technologiích lze zadat stiskem tlačítka *Hledat nová zařízení*. Po nalezení nových zařízení se tlačítkem *Přidat* nad nalezeným zařízením zobrazí formulář pro zadání názvu zařízení a vybrání varianty. Napravo od formuláře je zobrazen seznam možných variant i s jejich popisem.

7.5.3 Komunikace s řídicí službou

Komunikace s řídicí aplikací probíhá skrz Unixový soket vytvořený řídicí aplikací. V zásadě je implementována jako jednosměrná, nazpátek se vrací pouze zpráva potvrzující úspěšný příjem a dekódování odeslané zprávy. Řídicí aplikace při komunikaci vystupuje v roli klienta, který se připojí k předem domluvenému soketu, odešle zprávu, vyčká na odpověď o příjmu a odpojí se.

Komunikační rozhraní je analogické k řídicí aplikaci. Nejprve se vytvoří objekt `SocketCommunicator` pomocí jeho statické metody `connectToUnixSocket`. Nad tímto objektem poté lze volat metody `read`, `write` a `close`. Tento objekt využívá objekt třídy `MessageCommunicatorClient` ke komunikaci skrz soket. V druhé jmenované třídě je implementována metoda `sendMessage`, která umí odeslat zprávu a vrátit zprávu potvrzující. Zpráva (objekt třídy `Message`) je analogická ke zprávě známé z řídicí aplikace a proto zde nebude blíže popisována.

Pro pohodlnější odesílání zpráv a jednodušší manipulaci s chybami byla vytvořena třída `Messanger`, která ke komunikaci používá instanci objektu `SocketCommunicatorClient`. Implementuje opět metodu `sendMessage`, parametrem je tentokrát třída `SimpleMessage`, která je potomkem třídy pro zprávu a zjednodušuje její vytváření na minimum – nastavení příjemce, předmětu a argumentů. Za pomocí dědičnosti jsou z tohoto typu zprávy vytvořeny konkrétní zprávy. Messanger je vytvářen pomocí DI kontejneru až v momentě, kdy je skutečně potřeba. Díky vytváření skrz kontejner je možné mít nastavení cesty k soketu v konfiguračním souboru `config.neon`. Diagram tříd související s komunikací lze nalézt na obrázku 7.5.



Obrázek 7.5: Diagram tříd znázorňující třídy související s komunikací

7.5.4 Konfigurace

Konfigurací je v rámci této podkapitoly myšlena konfigurace ovladačů a okruhů, fungující na principu registru, který obsahuje klíč a hodnotu. Konfigurace má mít přímý vliv na průběh výpočtu ve vlákně příslušného ovladače. Proto je nutné její změnu okamžitě oznámit zprávou výkonné službě, běžící nepřetržitě na pozadí.

Pro snazší manipulaci s konfigurací z komponenty ovladače byla vytvořena třída `App\Services\Configurator`, která je připravena k použití v povinném abstraktním předkovi každé z komponent ovladače. Nad tímto objektem je možné volat následující metody pro práci s konfigurací:

- `set($key, $value)` – nastaví hodnotu ke specifikovanému klíči v registru. Pokud záznam s tímto klíčem neexistuje, vytvoří ho.
- `get($key)` – vrací objekt z modelové vrstvy se záznamem, který má požadovaný klíč. Pokud není záznam nalezen, je vráceno NULL.
- `remove($key)` – odstraní vybraný klíč i s jeho hodnotou z registru, pokud se v něm nachází.

Při jakékoliv změně registru je tato skutečnost automaticky oznámena zprávou řídicí službě se správně nastaveným příjemcem a klíčem, který se změnil. Další výhodou popsané třídy je její znovupoužitelnost pro konfiguraci ovladače, vše se liší pouze při vytváření objektu, kdy se do konstruktoru předá jiná entita z modelové vrstvy. Na změnách konfigurace je postaven například manuální mód u GPRelay ovladače.

Pro vizuální konfiguraci typu registr je připravena komponenta, která využívá popsanou třídu a rozšiřuje ji o grafickou stránku. Jedná se o komponentu `App\Components\ConfigurationComponent`, která zobrazuje hodnoty v tabulce klíč – hodnota a umožňuje provádět všechny výše popsané operace. Tato komponenta je použita v nastavení okruhu v záložce *Konfigurace* i v nastavení ovladače ve stejnojmenné záložce.

7.5.5 Grafy

Vykreslování hlavních grafů bude probíhat na klientské straně pomocí Javascriptové knihovny `Flot`. Tato knihovna zvládne zobrazit několik různých sérií dat najednou, což je pro porovnání skutečné a naplánované hodnoty vyžadováno. Pro zobrazení naplánovaných hodnot jako série horizontálních sloupců vedle sebe bylo nutné mírně upravit zdrojový soubor knihovny, jelikož tento typ grafu není v knihovně standardně nabízen (podobně jako v jiných knihovnách). Graf bude možné filtrovat podle data a určovat tak rozmezí, jaké má graf zobrazovat. V grafu bude navíc možné zvětšit určitou oblast jejím vybráním a možnost později tento výběr zrušit.

Veškerá komunikace grafu, který je zobrazován u klienta se serverem, na kterém jsou připravována data pro vykreslení, probíhá pomocí technologie AJAX až po načtení stránky, na které se graf nachází. Důvodem je snížení latence při zobrazení stránky s grafem. Stránka se zobrazí bez zbytečného zdržování a až poté vznikne asynchronně požadavek na data pro graf, která se na serveru začnou připravovat a po připravení jsou ihned odeslána a vykreslena na klientské straně. Během čekání na data je místo grafu zobrazena zpětná vazba pro uživatele grafickým indikátorem. Při změně rozsahu grafu je znovu asynchronně bez obnovení stránky požádán server o dodání nových dat. Během čekání na nová data je znovu

zobrazen indikátor načítání. V případě, že se nepodaří data připravit, indikátor změní barvu ze zelené na červenou.

Pro znovupoužitelnost grafu ve více komponentách ovladačů a možnost spravovat graf z jednoho místa byla vytvořena komponenta `App\Components\Graph\GraphComponent`, která graf obaluje a poskytuje veškerou popsanou funkcionalitu. Pro použití v komponentě ovladače je potřebné vytvořit továrnu, která bude komponentu grafu vytvářet a přidat makro pro vykreslení komponenty do šablony. V továrně lze přidat objektu grafu série, které mají být vykresleny pomocí dvou metod:

- `addLineSerie($name)` – vytvoří objekt pro vykreslení křivky, který přidá do grafu a vrátí jej
- `addBarSerie($name)` – vytvoří objekt pro vykreslení horizontálních sloupců s určitou šířkou a vrátí jej

Nad vrácenými objekty z popsaných metod lze volat další metody pro nastavení vlastností přidávané série (např. barvy) a přidání dat. Data lze přidat buď jako pole hodnot nebo pomocí callbacku, který je volán až v případě potřeby získat data. Pomocí callback funkce je řešen problém filtrování hodnot podle časové osy (interval od – do). V rámci této práce plně dostačují uvedené dva druhy sérií. Pokud by bylo potřeba přidat jiný druh, lze to provést vytvořením nové třídy, která jej bude reprezentovat a současně bude potomkem třídy `GraphSerie`.

7.5.6 Programy

Tvorba programů má dle specifikace probíhat grafickou cestou, která je naznačena na obrázku 7.2. Pro nastavení programu bude sloužit sloupcový graf, jehož vodorovná osa představuje čas v rámci jednoho dne a na svislé ose jsou hodnoty, které je možné nastavit. Tyto hodnoty se liší v závislosti na vytvářeném typu programu (sloupec `value_type` v databázi). Typ programu je tedy určen virtuálním typem návratové hodnoty (např. teplota, relé). Po vytvoření nového programu je v grafu pouze jeden sloupec. Pro vytvoření programu znázorněnou metodou budou potřeba tři operace:

- *Posunutí* – chytnutím sloupce za jeden z okrajů a jeho tažením ve vodorovném směru lze měnit začátek a konec programového záznamu. Chytnutím sloupce za horní hranu a tažením nahoru, resp. dolů lze měnit jeho hodnotu.
- *Rozdělení* – kliknutí do plochy sloupce rozdělí v místě kliknutí tento sloupec na dva. Nový sloupec přebírá hodnotu od rozděleného a mohou na něj být aplikovány další operace.
- *Odstranění* – sloupec lze odstranit kliknutím do jeho plochy. Po odstranění sloupce se roztáhne sloupec vlevo od odstraňovaného. Odstranit nelze pouze první sloupec, protože program musí vždy nabývat nějaké hodnoty.

Jelikož se nepodařilo nalézt knihovnu, která by poskytovala popisovanou funkčnost, byla pro tuto práci vytvořena nová Javascriptová knihovna *TGraph*, jejímž hlavním cílem je právě nastavování hodnot v individuálních časových úsecích v rámci 24 hodin. Vytvořená knihovna používá funkce ze třech existujících knihoven: JQuery[3] pro manipulaci s objekty,

[3] jQuery – write less, do more: https://jquery.com/

JQuery UI[4] pro změny velikosti html elementů a JQuery UI Touch Punch[5] pro podporu dotykových obrazovek. Výsledný graf je tvořen pouze html elementy, které jsou při jeho konstrukci vytvořeny.

Interaktivní graf je možné vytvořit metodou `TGraph.create(placeholder, data, options)`, které je nutné předat tři parametry. Prvním parametrem je objekt s vybraným kontejnerem, ve kterém bude graf vykreslen. Druhým parametrem jsou data, ze kterých má být graf načten. Data jsou pole objektů, které mají nastavené atributy *from*, *to* a *value*, nepřekrývají se v čase a jsou seřazena vzestupně podle atributu *from*. Třetím parametrem je objekt nastavení grafu, kde lze ovlivňovat jeho podobu a vnutit používání vlastních hodnot a popisků vložením asociativního pole. Funkce vykreslí graf podle zadaných parametrů a vrátí objekt `TGraph.Graph`, který graf reprezentuje. Nad tímto objektem lze volat metody pro změnu aktivní operace a metodu `getData` pro získání aktuálně nastavených dat. Vrácená data jsou ve stejném formátu jako vstupní parametr *data* metody `create`. Vytvořený interaktivní graf pro nastavování programu v rámci jednoho dne je zobrazen na obrázku E.3.

Program bude vytvářen vždy pro jeden konkrétní typ hodnoty (např. teplotu, relé). Tyto typy hodnot jsou uloženy v databázové tabulce `value_type`. Návratový typ vytvářeného programu je odvozen od typu hodnoty, který lze naplánovat okruhu, pro který je program vytvářen. Po vytvoření programu je možné jej naplánovat i jinému okruhu, podmínkou je pouze, aby virtuální typ hodnoty, který program vrací, byl stejný jako virtuální typ hodnoty, který lze okruhu naplánovat.

V rámci backendu bylo nutné vyřešit situaci, kdy pro různé typy virtuálních hodnot uložené v databázové tabulce `value_type` je třeba nastavovat různé hodnoty. Pro nastavení programu relé jsou těmito hodnotami dva stavy „on“ resp. „off“, pro nastavení programu teploty je těmito hodnotami rozsah teplot, například 12 – 28 °C. Aby mohlo být řešení rozšiřitelné, byla vytvořena abstraktní třída `App\ValueType\ValueType`, která definuje mechanizmy pro práci s těmito hodnotami v rámci různých typů virtuálních hodnot. Pro konkrétní typ virtuální hodnoty pak stačí vytvořit třídu ve stejném jmenném prostoru, která je potomkem zmiňované abstraktní třídy, a jejíž název je stejný jako v databázovém sloupci `type` použité tabulky s typy hodnot. V této třídě stačí při vytváření přidat do asociativního pole předka hodnoty, které mohou být nastavovány. Klíčem v tomto poli je hodnota, která bude nastavena do programového záznamu při uložení programu do databáze a hodnotou je popisek, který bude v grafu při nastavování zobrazen.

Objekty konkrétního typu hodnot jsou vytvářeny dynamicky podle jména v momentě, kdy je to opravdu potřeba. Pokud objekt nelze vytvořit, je tato skutečnost interpretována tak, že pro daný typ virtuální hodnoty nelze program vytvořit.

7.5.7 Import definic

Import definic slouží k přidání nebo aktualizování tabulek v databázi, které mají význam rozšiřitelných částí, jednoduchou cestou přímo z ovládací aplikace. Jedná se především o entity ovladačů, vzorů soketů, technologií a virtuálních návratových typů. Import lze nalézt v nastavení a provést jej může pouze administrátor.

Samotný import se skládá pouze z nahrání validního XML souboru s definicemi. Vše ostatní, jako například duplicity nebo aktualizace, si vyřeší nástroj sám. Formát importovaného souboru lze nalézt na přiloženém CD v adresáři ovládací aplikace, kde je předpřipraven

[4] jQuery User Interface: https://jqueryui.com/
[5] Touch Event Support for jQuery UI: http://touchpunch.furf.com/

soubor *exampleImport.xml*, který byl použit v rámci předvedení této práce. Import probíhá v transakci, což znamená, že pokud se vyskytne jediná chyba, stav databáze zůstane nezměněn a uživateli je oznámena chybovou hláškou.

V kořenovém elementu XML souboru jsou umístěny tři kolekce struktur postupně pro všechny rozšiřitelné části: *technologies*, *modules* a *valueTypes*. Informace o vzorech soketů jsou součástí ovladače. V rámci každé kolekce pak může být libovolný počet struktur. Názvy atributů jednotlivých struktur vychází z názvů z databáze.

7.5.8 Chybové hlášky

Pro zobrazení chybových hlášek byla vytvořena komponenta `ProtocolComponent`, která umí vykreslovat jak kartu s posledními několika hláškami, tak i celostránkový přehled, včetně možnosti stránkování a filtrování podle důležitosti hlášky. Stránkování a filtrování probíhá s využitím technologie AJAX, tedy na pozadí bez nutnosti obnovení stránky. Pokud nelze v prohlížeči použít Javascript, stránka bude fungovat bez vizuálních rozdílů oproti použití technologie AJAX. Jediný rozdíl bude ve změně url adresy při každém přechodu na novou stránku nebo při změně filtru. Náhledová karta je použita na úvodní přehledové stránce systému, odkud je možné přes odkaz přejít na celostránkový přehled.

Zobrazení hlášek stavu okruhu (EventLog) je také přístupné jako náhledová komponenta (třída `EventLogPreviewComponent`) a zároveň jako celostránkový přehled s použitím metody `renderEventLog` nad komponentou ovladače.

Kapitola 8

Rozšiřitelnost systému

Důležitou částí systému je jeho rozšiřitelnost. Tato kapitola popisuje kroky, které je nutné provést při rozšiřování systému o novou funkcionalitu. Systém se skládá ze tří částí: ovládací služby, řídicí aplikace a databáze, které spolu komunikují. Při jeho rozšiřování je žádoucí reflektovat změny ve všech zmiňovaných částech. Rozšířením systému se rozumí přidání nové funkcionality, nových typů zařízení a technologie, která má zodpovědnost za komunikaci s fyzickými zařízeními.

Systém bez jakéhokoliv rozšíření obsahuje pouze jádro. V rámci řídicí služby jsou spuštěny nástroje pro komunikaci s ovládací aplikací, ovládání led diod na modulu *bus-driver* a odchytávání systémových signálů. Aplikace po takovém spuštění navenek nevykonává žádnou automatizační činnost. V rámci ovládací aplikace pak lze měnit stav databáze, který nesouvisí s rozšiřitelnými částmi (např. uživatelská správa, správa místností, protokol). Ostatní funkcionalita je zpřístupněna až v okamžiku dostupnosti rozšiřitelných částí. Stav rozšiřitelných částí v databázi je důležitý, jelikož právě podle něj jsou dynamicky vytvářeny v ovládací službě a nastavovány v řídicí aplikaci.

V rámci řídicí služby jsou rozšiřitelné části vytvářené továrnami, které na základě unikátního jména vrací ukazatel na nově vytvořenou instanci rozšiřitelné části. Každá z rozšiřitelných částí má svou továrnu umístěnou ve třídě `Factory` v příslušném podadresáři rozšiřované části. Při přidávání je proto nutné zaregistrovat továrnu pro vyrobení instance příslušné třídy reprezentující rozšíření do příslušné třídy `Factory`. Továrna pro vytvoření má tvar metody, která vrací ukazatel na příslušnou rozšiřitelnou část.

Při rozšiřování je důležité dodržet použití stejného unikátního názvu pro novou část ve všech částech systému, jelikož právě podle tohoto identifikátoru je možné spojit rozšiřovanou část v jednotlivých částech systému. Po jakémkoli rozšíření řídicí aplikace je nutné ji znovu přeložit a restartovat běžící službu *automation*.

8.1 Zařízení

Zařízení v kontextu rozšiřitelnosti představuje pouze obálku, která definuje jeho chování, ukládání hodnot či jejich interpretaci. Konkrétní zařízení vzniká až ve spojení s technologií, která bude ovládat jeho fyzický protějšek.

Řídicí službu lze rozšířit o nová zařízení vytvořením třídy nesoucí název tohoto zařízení, podle pravidel popsaných v kapitole 6.3. Důležité je zaregistrovat továrnu nově vytvořené třídy, aby bylo možné zařízení vytvořit na základě unikátního jména.

V ovládací aplikaci je vhodné vytvořit komponentu pro nové zařízení (resp. typ virtuální

hodnoty, kterou vrací), aby mohlo být zobrazeno na mapě patra spolu s interpretovanou hodnotou. Pokud tato komponenta neexistuje, zařízení nebude možné zobrazit na mapě, ale ostatní funkčnost nebude ovlivněna.

V databázi je nutné přidat do tabulky s typy hodnot nový virtuální typ hodnoty, který toto zařízení vrací. Nejjednodušší cestou je použít import definic popsaný v podkapitole 7.5.7.

8.2 Technologie

Technologií je v rámci této práce myšlen přístupový bod pro komunikaci s fyzickými zařízeními, která pracují na stejné technologii, tentokrát je myšlena konkrétní realizační technologie (např. 1-Wire, Bluetooth Low Energy). Po dodání obálky zařízení umožní technologie provádět operace vstupu a výstupu nad tímto zařízením v závislosti na jeho povaze – senzor/aktuátor.

V řídicí službě se technologie rozšiřuje vytvořením nové třídy, podle pravidel popsaných v kapitole 6.6. Důležité je zaregistrovat její tovární metodu v továrně na výrobu tříd technologií podle jména, která se nachází ve třídě `Technology::Factory`.

V rámci ovládací aplikace je nutné pro novou technologii vytvořit třídu, která ji reprezentuje a která umožní vyhledávat a přidávat nová zařízení nad touto technologií. Pravidla pro vytvoření této třídy jsou popsána v podkapitole 7.5.2.

Definici třídy je nutné přidat i do databáze, kde se jedná především o její unikátní jméno, které ji umožní stmelit v jeden celek v rámci ovládací a řídicí aplikace. Pro vložení této definice lze použít import definic.

8.3 Rozšíření funkcionality

Rozšíření funkcionality představuje hlavní náplň v rozšiřování. Jedná se o algoritmy pro řízení konkrétních automatizačních činností. Funkcionalita je reprezentována ovladačem, resp. modulem. Ovladač ke svému chodu může vyžadovat přítomnost určitých typů zařízení.

Řídicí službu lze o nový ovladač rozšířit vytvořením nové třídy podle pravidel popsaných v kapitole 6.5. Opět je nutné zaregistrovat jeho tovární metodu ve třídě `Module::Factory`.

V ovládací aplikaci je vyžadováno vytvořit komponentu pro zobrazování náhledu ovladače a jeho detailu podle pravidel popsaných v podkapitole 7.5.1. Přítomnost této komponenty je důležitá, bez ní nebude možné zobrazit v ovládací aplikaci detaily o stavu okruhu používající tento ovladač.

Do databáze je nutné přidat definiční záznam o novém ovladači, jelikož právě díky tomuto záznamu bude možné ovladač vytvořit a používat. Definice ovladače se skládá ze záznamu v databázové tabulce `modul`, kde lze určit, jaký typ hodnoty může být ovladači plánován, a záznamu o jeho přípojných bodech v tabulce `socket_pattern`, kde lze v každém záznamu určit, jaké zařízení lze do přípojného bodu připojit (podle virtuálního typu hodnoty, který zařízení vrací) a jak bude se zařízením nakládáno – čtení nebo čtení/zápis.

Kapitola 9

Závěr

V rámci této práce byl navrhnut systém pro automatizaci domácnosti za použití počítače Raspberry Pi jako centrálního prvku. V první fázi byly navrhnuty hardwarové moduly potřebné pro automatizační činnosti, které specifikuje zadání práce. V další fázi byla navrhnuta řídicí aplikace, která je spuštěna operačním systémem ihned po startu a jejímž úkolem je provádět automatizační činnosti na základě stavu databáze. S databází manipuluje ovládací aplikace, která umožňuje monitorovat, nastavovat průběhy automatizačních činností a plánovat programy s přednastavenými hodnotami závislými na čase. Při návrhu obou aplikací byl kladen důraz na jejich rozšiřitelnost. Výsledkem je jádro, které poskytuje určité funkce, ale samo o sobě žádnou automatizační činnost nevykonává. Tato činnost je vykonávána ovladači, které jsou jednou z rozšiřitelných částí. V rámci práce byly navrženy a implementovány ovladače pro automatizaci centrálního vytápění, řízení kotle, spínání spotřebičů a vypisování informací na LCD displej.

Po fázi návrhu byly navržené moduly vyrobeny a otestovány. Výroba modulů se skládala z vyrobení desky plošného spoje, jeho osazení a oživení. Zároveň s výrobou modulů probíhala implementace obou navržených aplikací. V rámci implementace a testování navrženého softwaru byl vytvořen manuál pro zprovoznění této práce na použitém počítači. Tento manuál shrnuje všechny kroky, které je nutné provést ke spuštění aplikací. Součástí manuálu jsou i instalační skripty, které instalaci zjednodušují.

V rámci implementace ovládací aplikace byl vytvořen nástroj *TGraph*, který umožňuje graficky plánovat hodnoty programu. Jedná se o interaktivní graf, kde na vodorovné ose je čas v rámci jednoho dne a na svislé ose jsou hodnoty, které je možné nastavit. V grafu jsou vedle sebe postaveny sloupce reprezentující programové záznamy. Dělením a posouváním sloupců pak lze vytvořit program.

Výsledkem práce je fungující systém, který byl v rámci předvedení zapojen a vyzkoušen na předváděcí desce. Jedná se o systém, který je možné rozšířit o nové automatizační činnosti, typy zařízení nebo technologie jejich připojení. Pro úspěšné rozšíření je třeba dodržet určitá pravidla, která byla shrnuta v kapitole 8. V rámci této práce bylo použito pouze technologie 1-Wire pro komunikaci se zařízeními. Komunikace probíhá pouze po sběrnici. Vhodným rozšířením práce by bylo vytvořit bezdrátová zařízení a doimplementovat jejich obsluhu. Dále by bylo vhodné rozšířit automatizaci vytápění o možnost ovládání elektronických hlavic se servopohonem. Pro tento účel by bylo vhodné vytvořit nový ovladač, případně modifikovat stávající a vytvořit hardwarový modul, jehož výstupem bude analogový signál 0 – 10 V, na základě kterého dokáže servopohon plynule ovládat pohyb dříku.

Literatura

[1] ELKO EP: iNELS smart home solutions - inteligentní řešení. http://www.elkoep.cz/produkty/inels-bus-system/, 2016 [cit. 2016-04-20].

[2] Etatherm: Regulační systém Etatherm. http://www.etatherm.cz/cesky/sys2_a.htm, 19. 4. 2016 [cit. 2016-04-22].

[3] Honeywell: MT4-024/MT4-230 Malý lineární termoelektrický pohon. https://products.ecc.emea.honeywell.com/cz/pdf/en0b0490-cz01r0213.pdf, 2013 [cit. 2016-03-19].

[4] Jablotron: Automatizace s JABLOTRON 100. http://www.jablotron.com/cz/automatizace/automatizace-s-jablotron-100/, 2016 [cit. 2016-04-20].

[5] Maxim Integrated: DS2406: Dual Addressable Switch Plus 1Kb Memory. http://datasheets.maximintegrated.com/en/ds/DS2406.pdf, 2009 [cit. 2016-03-15].

[6] Maxim Integrated: DS18B20: Programmable Resolution 1-Wire Digital Thermometer. https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf, 2015 [cit. 2016-03-15].

[7] Maxim Integrated: DS2408: 1-Wire 8-Channel Addressable Switch. http://datasheets.maximintegrated.com/en/ds/DS2408.pdf, 2015 [cit. 2016-03-15].

[8] Maxim Integrated: DS2482-100: Single-Channel 1-Wire Master. https://datasheets.maximintegrated.com/en/ds/DS2482-100.pdf, 2015 [cit. 2016-03-15].

[9] Norris, D.: *Raspberry Pi: projekty*. Brno: Computer Press, 2015, ISBN 978-80-251-4346-9.

[10] OWFS 1-Wire File System: LCD screen controllers. http://owfs.org/index.php?page=lcd, 2003-2016 [cit. 2016-03-18].

[11] Plíva, Z.: *EAGLE prakticky: řešení problémů při běžné práci*. Praha: BEN – technická literatura, 2010, ISBN 978-80-7300-252-7.

[12] Prata, S.; Sokol, B.: *Mistrovství v C++*. Brno: Computer Press, 2013, ISBN 978-80-251-3828-1.

[13] Texas Instruments: SN54HC240, SN75HC240 Octal Buffers And Line Drivers With 3-State Outputs. http://www.ti.com/lit/ds/symlink/sn54hc240.pdf, 2003 [cit. 2016-03-11].

[14] Texas Instruments: LM2940x 1-A Low Dropout Regulator. http://www.ti.com/lit/ds/symlink/lm2940c.pdf, 2014 [cit. 2016-03-11].

[15] Texas Instruments: ULN2803A Darlington Transistor Arrays. http://www.ti.com/lit/ds/symlink/uln2803a.pdf, 2015 [cit. 2016-03-11].

[16] Upton, E.; Halfacree, G.; Goner, J.: *Raspberry Pi : uživatelská příručka 1. vyd.* Brno: Computer Press, 2013, ISBN 978-80-251-4116-8.

[17] Šandera, J.: *Návrh plošných spojů pro povrchovou montáž.* Praha: BEN – technická literatura, 2006, ISBN 80-7300-181-0.

Přílohy

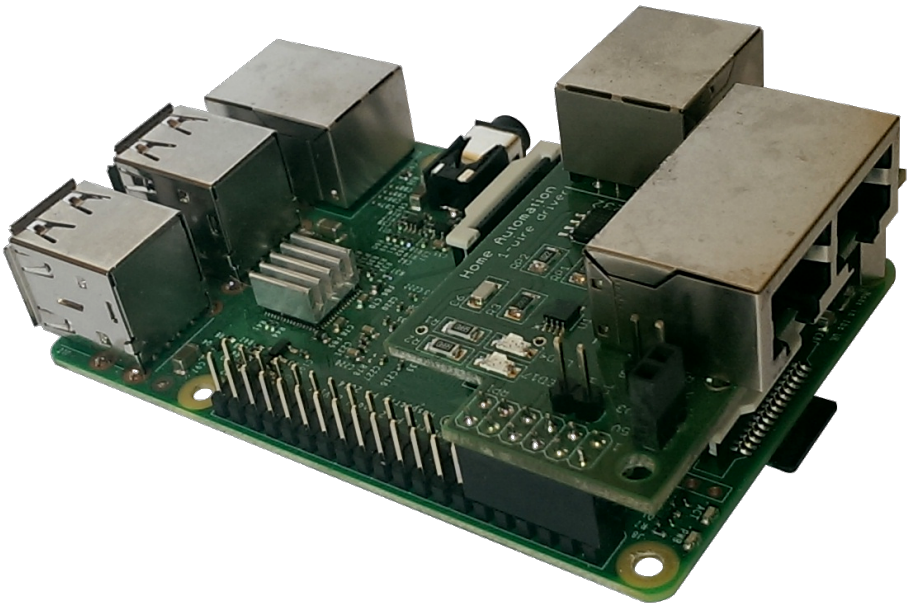
Seznam příloh

Příloha A

Vyrobené moduly

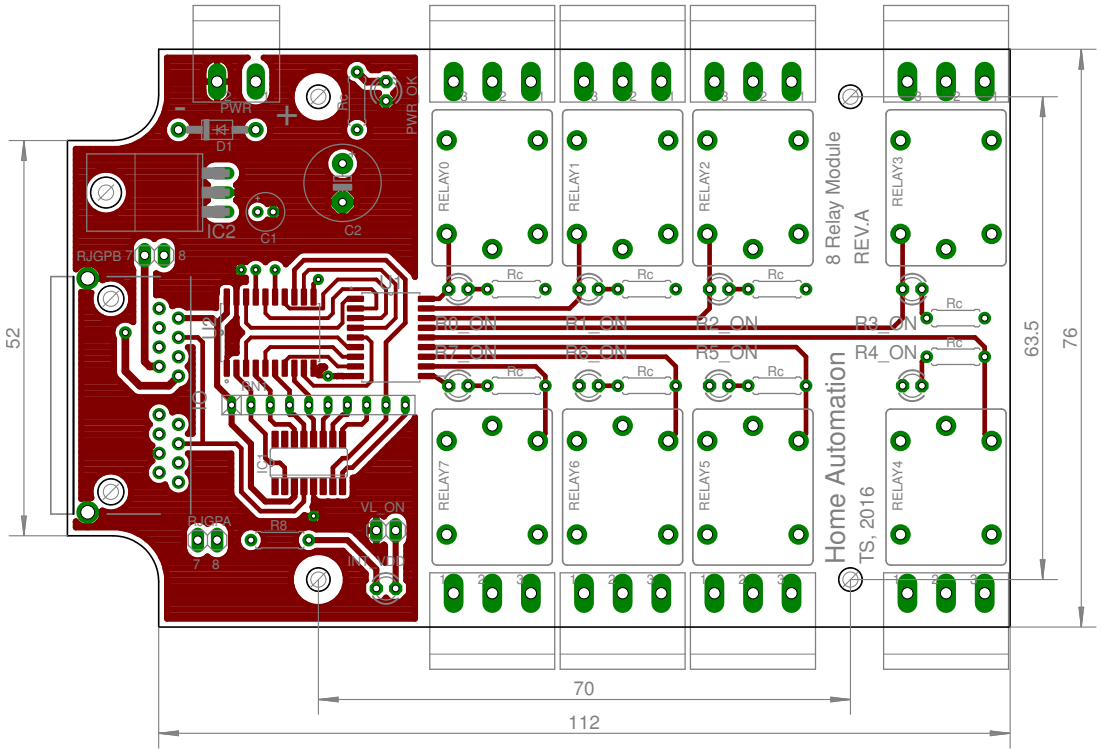
V této příloze lze nalézt fotografie všech vyrobených modulů. Nachází se zde i otisky desek, které byly příliš velké a nemohly být vloženy do kapitoly o jejich návrhu. Více fotografií jednotlivých modulů lze nalézt na přiloženém CD.

Modul bus-driver

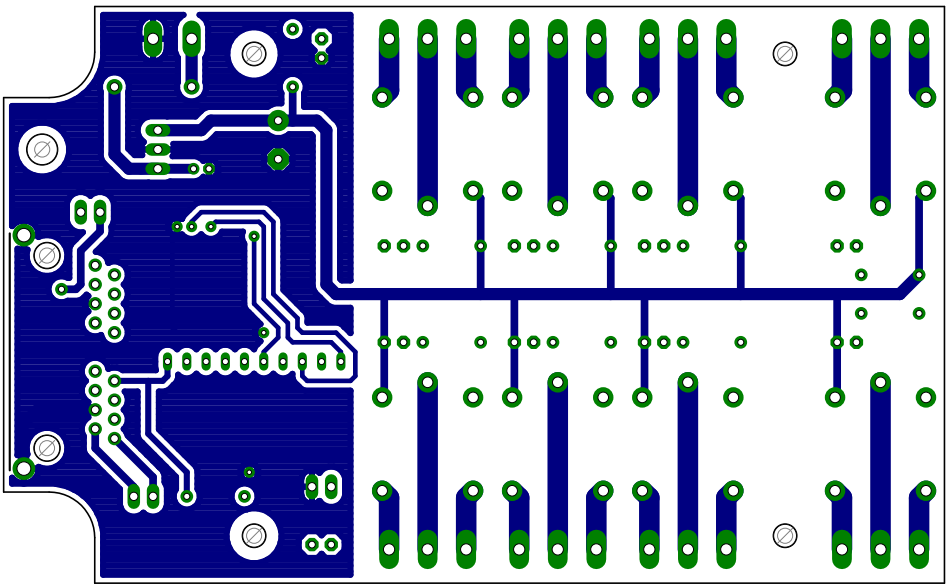


Obrázek A.1: Vyrobený modul *bus-driver* na počítači Raspberry Pi

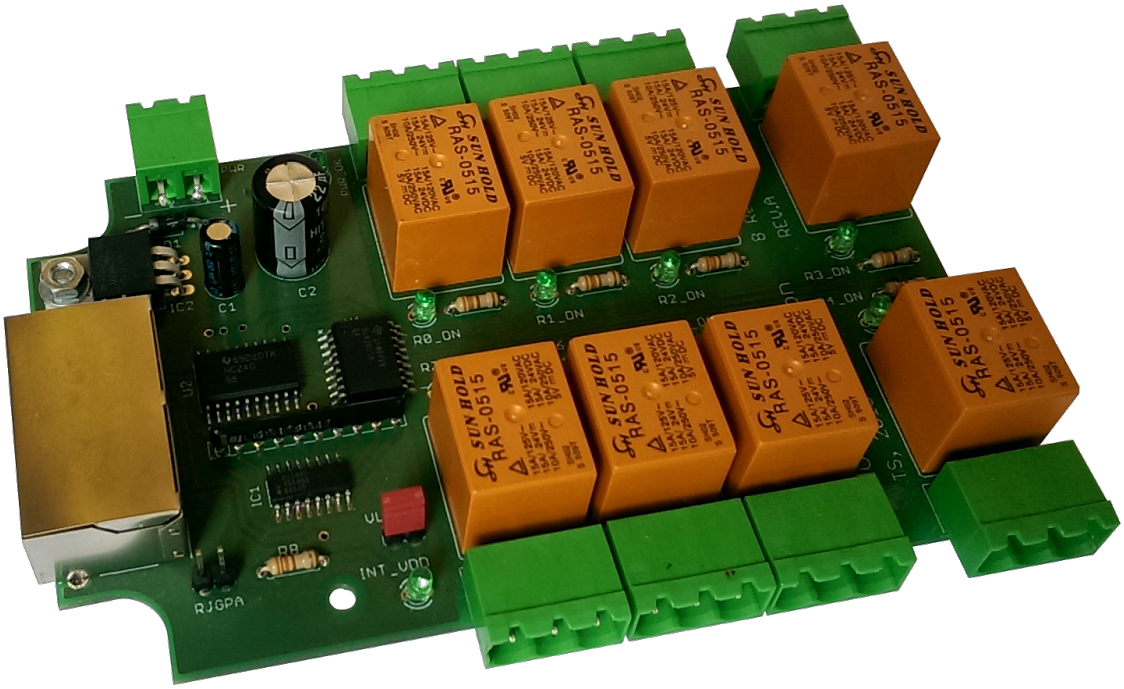
Modul s osmi relé



Obrázek A.2: Vrchní strana otisku desky modulu s osmi relé se součástkami

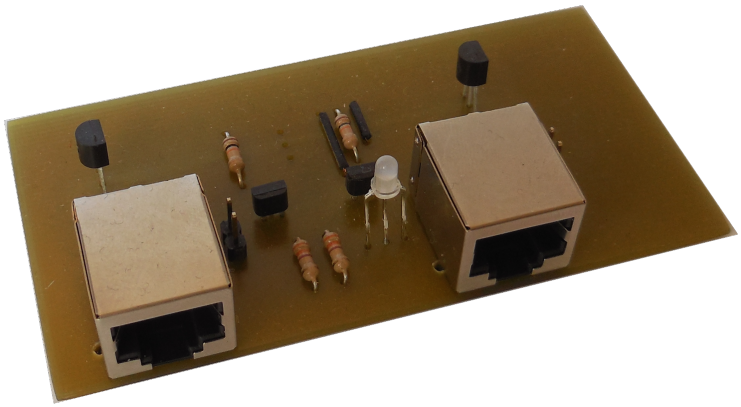


Obrázek A.3: Spodní strana otisku desky modulu s osmi relé



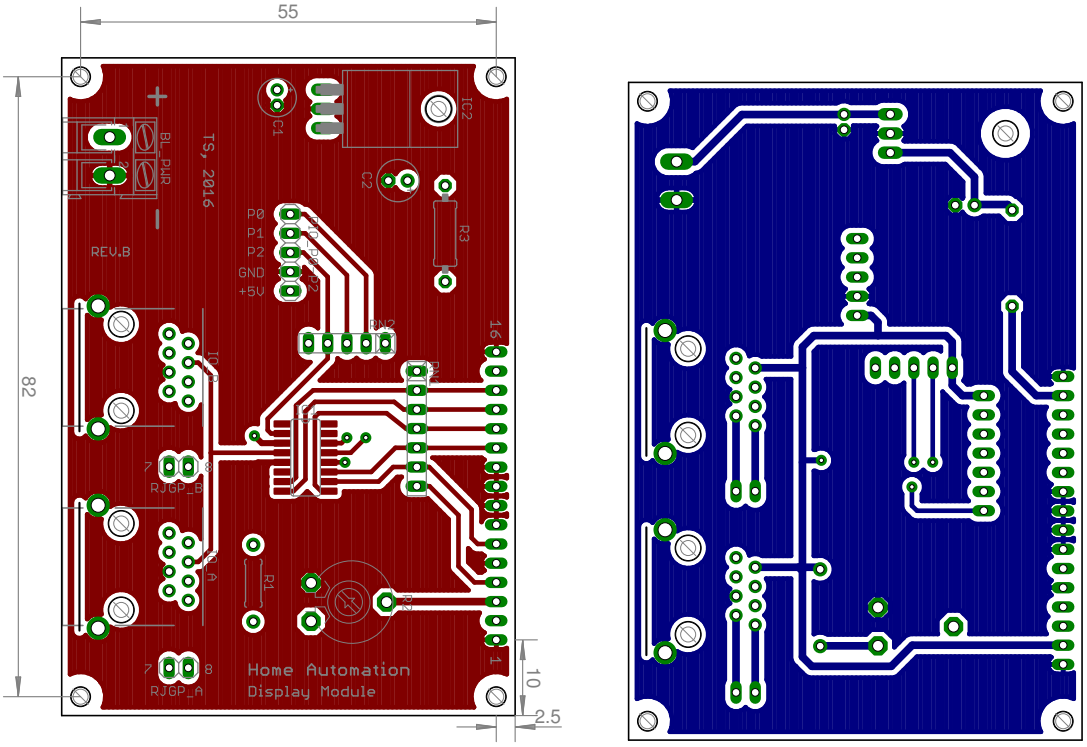
Obrázek A.4: Vyrobený modul s osmi relé

Senzorový modul



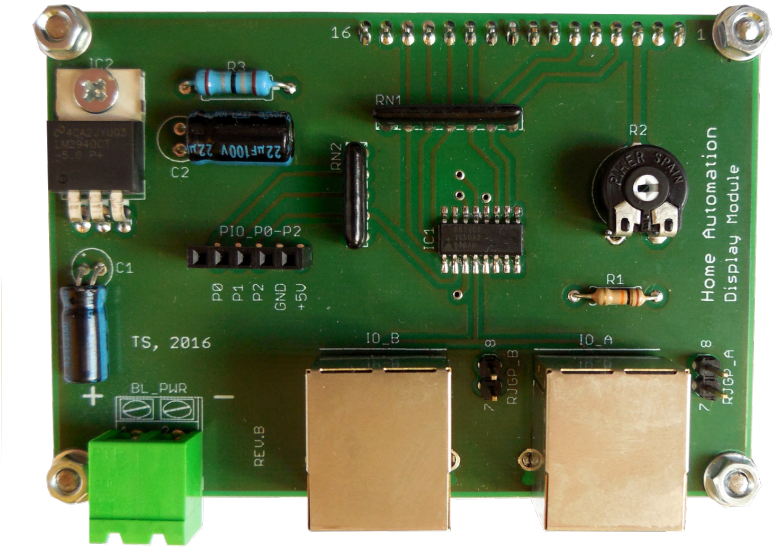
Obrázek A.5: Vyrobený senzorový modul

Modul pro lcd displej



(a) Vrchní strana se součástkami (b) Spodní strana

Obrázek A.6: Otisk desky navrženého modulu pro displej



Obrázek A.7: Vyrobený modul pro ovládání LCD displeje s řadičem

Příloha B

Manuál instalace projektu

V tomto manuálu bude krok po kroku popsána instalace vytvořeného systému na počítač Raspberry Pi. Na počítači se předpokládá nainstalovaný operační systém Raspbian Lite. Návod byl napsán na základě instalace na jmenovaném operačním systému s datem uvolnění 18. 3. 2016. Všechny soubory potřebné k instalaci lze nalézt na přiloženém CD. Při instalaci bude potřeba přístup k internetu. Postup instalace je následující:

1. Zprovoznit komunikaci s počítačem, ať už přes rozhraní RS232, které nabízí modul *bus-driver* nebo přes síťové prostředí pomocí ssh klienta. Poslední možností je připojit k počítači monitor s klávesnicí.

2. Jelikož pracujeme na čerstvě nainstalovaném systému, je vhodné provést nejprve pár základních nastavení: roztáhnout kapacitu disku přes celou paměťovou kartu, nastavit časovou zónu a upgradovat nainstalované balíky na nejnovější verzi.

   ```
   sudo raspi-config -> 1 Expand filesystem
   sudo raspi-config -> 4 Internationalisation Options -> Change Timezone
       -> Prague
   sudo apt-get update && sudo apt-get upgrade
   ```

   Spuštěním příkazu `raspi-config` se zobrazí grafický výběr možností, výše uvedené šipky znázorňují cestu. Po provedené těchto příkazů je vhodné restartovat počítač pro dokončení roztažení souborového systému přes celou kartu.

3. Instalace LAMP stacku. Nejprve nainstalujeme webový server Apache2 z repozitáře příkazem

   ```
   sudo apt-get install apache2
   ```

   Poté nainstalujeme MySQL server a provedeme poinstalační kroky. Při instalaci bude nutné zadat heslo uživatele *root* MySQL databáze.

   ```
   sudo apt-get install mysql-server php5-mysql
   sudo mysql_install_db
   sudo /usr/bin/mysql_secure_installation
   ```

Jelikož MySQL databáze neobsahuje tabulku časových zón, je nutné ji vložit ručně pomocí následujícího příkazu. Při výzvě zadejte heslo uživatele root pro MySQL databázi zvolené při instalaci.

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql -p
```

Dále nastavíme výchozí připojení k databázi v kódování UTF-8. Do souboru `/etc/mysql/my.cnf` na konec souboru do požadované sekce přidáme tyto řádky:

```
[mysqld]
collation-server = utf8_unicode_ci
init-connect='SET NAMES utf8'
character-set-server = utf8
```

Poté nainstalujeme PHP pomocí příkazů

```
sudo apt-get install php5 libapache2-mod-php5 php5-mcrypt
sudo apt-get install php5-sqlite
```

V souboru `/etc/apache2/mods-enabled/dir.conf` přesuneme *index.php* na první místo a dokončíme instalaci zapnutím modulu *rewrite* a restartem serveru:

```
sudo a2enmod rewrite
sudo service apache2 restart
```

Nakonec můžeme nainstalovat `PHPMyAdmin` pro přístup k databázi z webového rozhraní, pokud je vyžadován. Při instalování je důležité mezerníkem vybrat v nabídce možnost *Apache2*, jinak bude třeba dokončit instalaci ručně.

```
sudo apt-get install phpmyadmin
```

4. Instalace nejnovější verze překladače C++, Boost knihoven a MySQL C++ konektoru:

```
sudo apt-get install g++-4.9
sudo apt-get install libboost-all-dev
sudo apt-get install libmysqlcppconn-dev
```

5. Dalším krokem je instalace virtuálního souborového systému OWFS, který zpřístupní 1-Wire sběrnici. Před instalací je nejprve nutné povolit I^2C sběrnici na počítači příkazem:

```
sudo raspi-config -> Advanced Options -> I2C enable -> Yes
```

Poté je nutné vložit do souboru `/etc/modules` následující dva řádky:

```
i2c-bcm2708
i2c-dev
```

dále následující řádky na konec souboru `/boot/config.txt`:

```
dtparam=i2c1=on
dtparam=i2c_arm=on
```

Po povolení sběrnice I^2C je vhodné restartovat počítač a nechat zavést moduly jádra, které se o sběrnici starají. Před instalací je vhodné nainstalovat prerekvizitní balíky:

```
sudo apt-get install automake autoconf autotools-dev libavahi-client-dev
libtool libusb-dev libusb-1.0-0-dev libfuse-dev swig python2.7-dev
tcl8.5-dev php5-dev i2c-tools
```

Instalace virtuálního souborového systému spočívá ve stažení archivu s nejnovější verzí, jeho rozbalení, spuštění konfigurace, překladu a instalace:

```
cd /usr/src
sudo wget  -O owfs-latest.tgz
    http://sourceforge.net/projects/owfs/files/latest/download
sudo tar xzvf owfs-latest.tgz
cd owfs-x.y-pz
sudo ./configure
sudo make -j 4
sudo make install
```

Po instalaci souborového systému je potřeba nainstalovat balík `fuse`, aby bylo možné přistupovat k adresáři `1wire` i bez administrátorských práv.

```
sudo apt-get install fuse
```

poté je nutné v souboru `/etc/fuse.conf` odstranit komentář z řádku

```
user_allow_other
```

Dalším krokem je vytvoření přípojného bodu, kam bude filesystém namountován:

```
sudo mkdir /mnt/1wire
```

Po vytvoření přípojného bodu se přesuneme do adresáře systémové inicializace, nakopírujeme soubor potřebný pro automatický start a uděláme jej spustitelným. Poté jej zaregistrujeme do správce služeb, aby se spouštěl automaticky po startu systému

```
cd /etc/init.d
sudo cp CD/instalace/start1wire.sh start1wire.sh
sudo chmod +x start1wire.sh
sudo update-rc.d start1wire.sh defaults
```

6. V dalším kroku vytvoříme uživatele, pod kterým aplikace poběží, a nasměrujeme webový server do správného adresáře. Uživatele *automation* vytvoříme příkazem

   ```
   sudo adduser --home /automation automation
   ```

   Ve vytvořeném domovském adresáři vytvoříme podadresáře pro řídicí a ovládací aplikace:

   ```
   sudo mkdir /automation/app
   sudo mkdir /automation/www
   ```

   Poté zkopírujeme připravený soubor `000-default.conf` do adresáře `/etc/apache2/-sites-available/000-default.conf`. Tímto jsme webový server nasměrovali do adresáře `/automation/www`. Dále změníme skupinu, pod kterou webový server běží, aby mohl přistupovat ke společnému soketu. V souboru `/etc/apache2/envvars` změníme řádek se skupinou za následující a restartujeme server.

   ```
   export APACHE_RUN_GROUP=automation
   sudo service apache2 restart
   ```

7. Předposledním krokem je vytvoření datových struktur v databázi potřebných pro chod aplikací. K tomuto účelu byl vytvořen skript `createDB.sh`, který se nachází v podadresáři `mysql` adresáře s instalačnimi soubory. Po spuštění je nutné zadat heslo uživatele *root* zvolené při instalaci MySQL databáze.

   ```
   sudo ./createDB.sh
   ```

8. Posledním krokem je nakopírování zdrojových souborů obou aplikací a jejich instalace. K této činnosti je opět předpřipravený skript `install.sh`, který se nachází v adresáři s instalačními soubory spolu s archivem se zdrojovými soubory `home_-automation.tgz`.

   ```
   cp CD/instalace/home_automation.tgz home_automation.tgz
   cp CD/instalace/install.sh install.sh
   sudo ./install.sh
   ```

   Instalační skript zkopíruje soubory do správných adresářů, nastaví přístupová práva a přeloží řídicí aplikaci. Tento skript lze ovládat dvěma parametry, které ovlivňují jaká z aplikací se bude instalovat. Bez parametru je provedena instalace obou. Pro zobrazení možných variant spusťte skript s přepínačem `-help`.

   Nakonec nakopírujeme spouštěcí skript `automation` připravený v instalačním adresáři do adresáře správce služeb a zaregistrujeme jej:

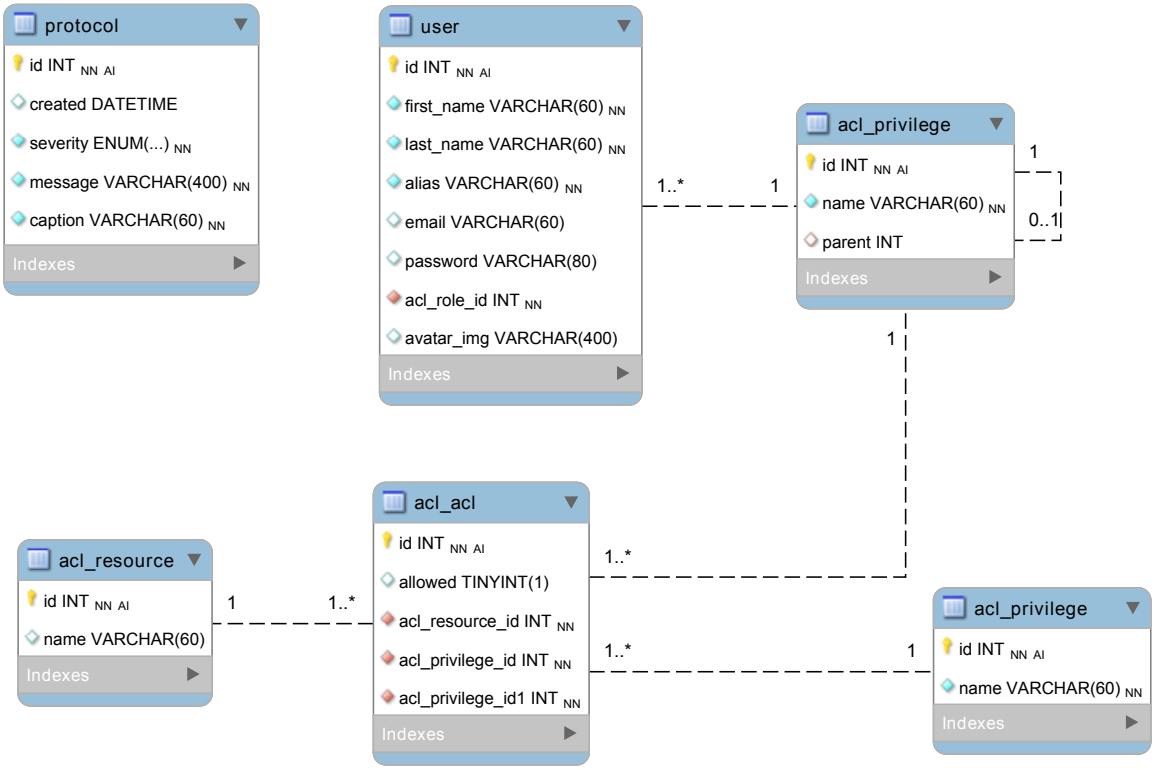
   ```
   sudo cp CD/instalace/automation /etc/init.d/automation
   sudo update-rc.d automation defaults
   ```

   Nyní již stačí restartovat počítač a řídicí aplikace by se měla sama spustit, což je indikováno zelenou led diodou na modulu `bus-driver`. Po nastavení síťového prostředí lze v prohlížeči otevřít ovládací aplikaci. Přístupové jméno výchozího uživatele je `admin` a heslo `123456`. Důrazně doporučujeme heslo výchozího administrátora změnit!

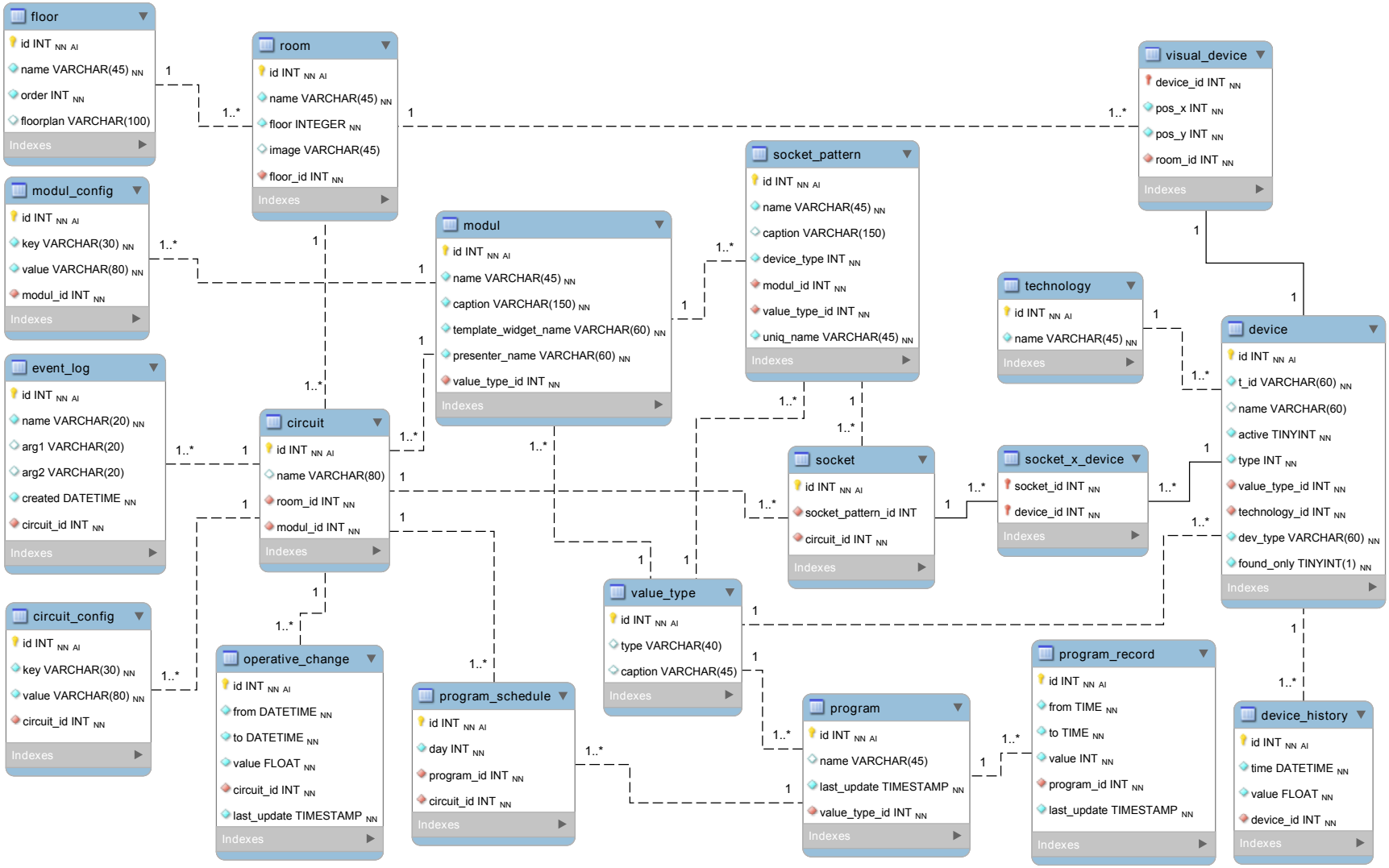
Příloha C

ER diagram databáze

E-R diagram je rozdělený na dvě části. Na hlavní části diagramu jsou zobrazeny hlavní entitní množiny, které pokrývají jádro aplikace. Na doplňkové části je zobrazena entitní množina pro protokol, která je využívána pro ukládání běhových i chybových hlášek. Ostatní entitní množiny slouží ke správě uživatelů a akcí, které mohou vykonávat v rámci ovládací aplikace.



Obrázek C.1: Doplňková část ER diagramu



Obrázek C.2: Hlavní část ER diagramu

Příloha D

Diagram tříd řídicí služby

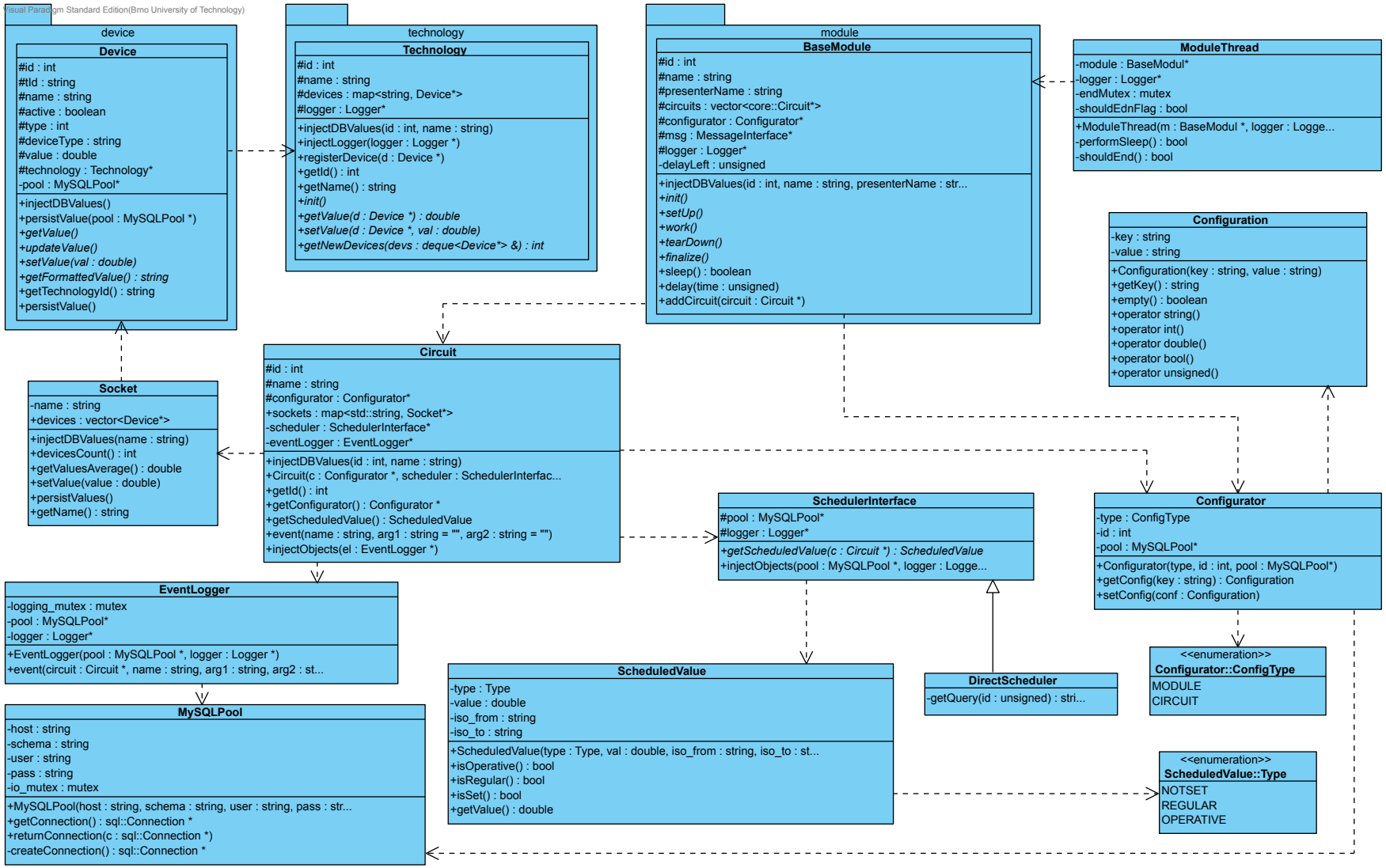
Diagram tříd řídicí aplikace je bohužel příliš obsáhlý, aby jej bylo možné zobrazit na jedné straně. Z tohoto důvodu byl rozdělen na čtyři související části. Na rozdělených diagramech jsou zobrazeny všechny třídy použité v aplikaci, bohužel z důvodu rozsáhlosti nebylo možné zobrazit kompletně všechny vazby mezi nimi. Všechny vztahy nebylo možné uvést hlavně u třídy `InstanceManager`, která je zodpovědná za vytvoření všech potřebných instancí tříd a tudíž závisí na mnoha z nich. Kompletní diagram tříd lze nalézt na přiloženém CD.

První část diagramu zobrazuje třídu okruhu a související třídy. Jde hlavně o abstraktní třídy `BaseModul`, `Technology` a `Device`. Při dynamickém sestavování podle stavu databáze jsou tyto abstraktní třídy nahrazeny instancemi tříd, které je implementují. Zařízení jsou k okruhu připojená skrz soket, což je obálka, která dovoluje manipulovat se všemi zařízeními současně nebo zpřístupnit každé zařízení zvlášť. Sokety jsou uloženy v asociativním kontejneru, kde klíčem je stejný řetězec, který je uveden v databázové tabulce `socket_pattern`. Důležitými třídami jsou také třídy konfigurace a plánovače, díky kterým lze dynamicky za běhu měnit nastavení okruhu.

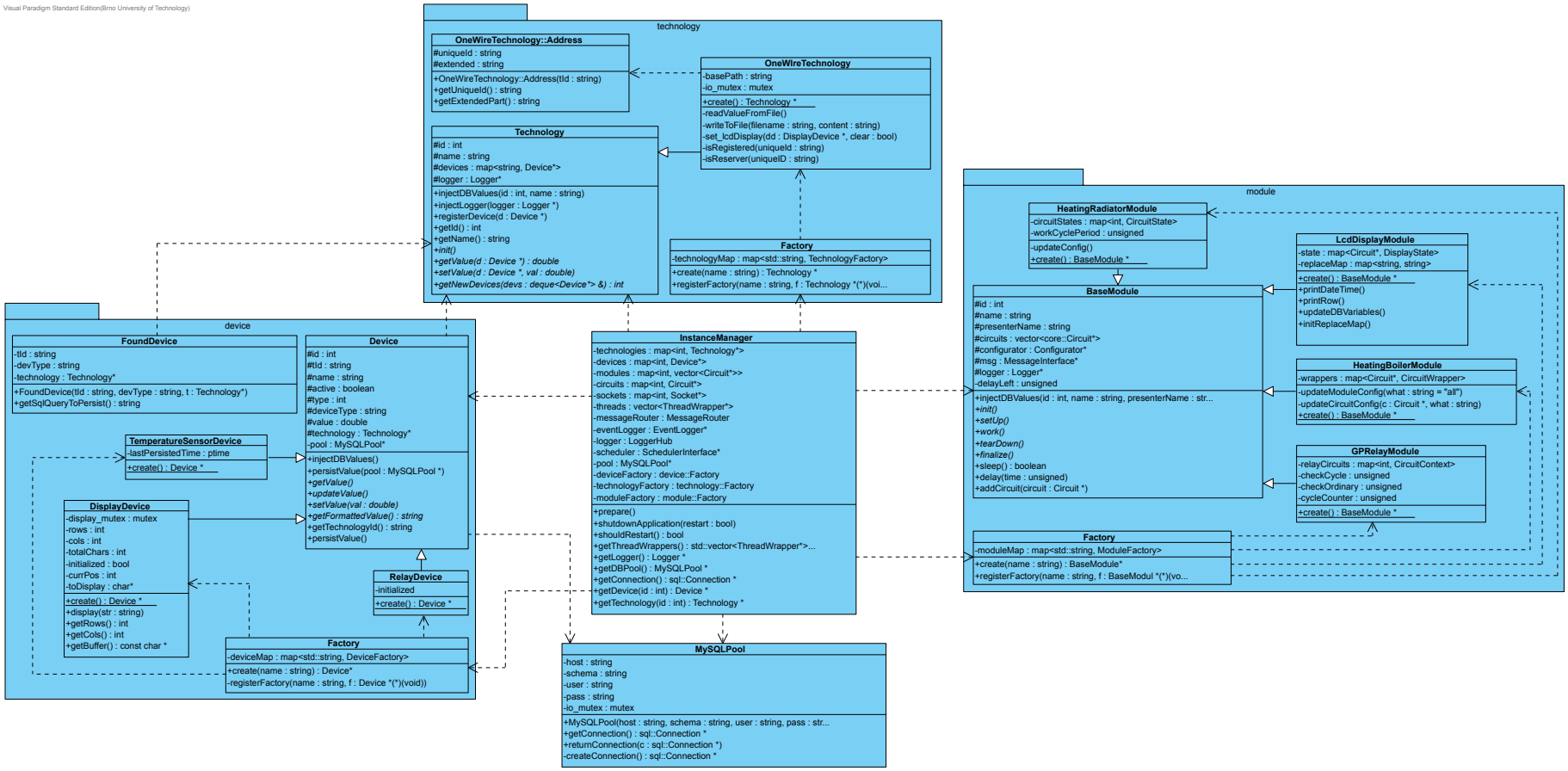
V druhé části se diagram zabývá vytvářením rozšiřitelných částí pří startu aplikace podle stavu databáze. Na diagramu jsou patrné rozšiřující třídy, které byly implementovány. V rámci každé rozšiřitelné části je implementována vlastní továrna, která má za úkol vytvořit instanci příslušné třídy podle unikátního jména, které je uvedené v databázi. Výroba instancí tříd se děje v rámci volání metody `prepare` nad objektem třídy `InstanceManager`.

Třetí diagram zobrazuje třídy, které dědí od abstraktní třídy `ThreadWrapper`. Jedná se o obálky, které obsahují výkonný kód, jež je spuštěn ve vlákně. Tyto třídy také musí implementovat metodu `stop`, která je volána asynchronně k běžící metodě `run` ve vlákně a jejímž cílem je nastavit příznak ukončení tak, aby běžící metoda `run` co nejdříve skončila. Na diagramu je zobrazeno i rozhraní pro práci s hláškami `Logger` a třídy implementující toto rozhraní. Za povšimnutí stojí třída `LedStatusThread`, která řídí LED diody na modulu `bus-driver` a implementuje jak abstraktní třídu pro spuštění ve vlákně, tak rozhraní pro práci s hláškami, na základě kterých spouští kontrolní LED diody.

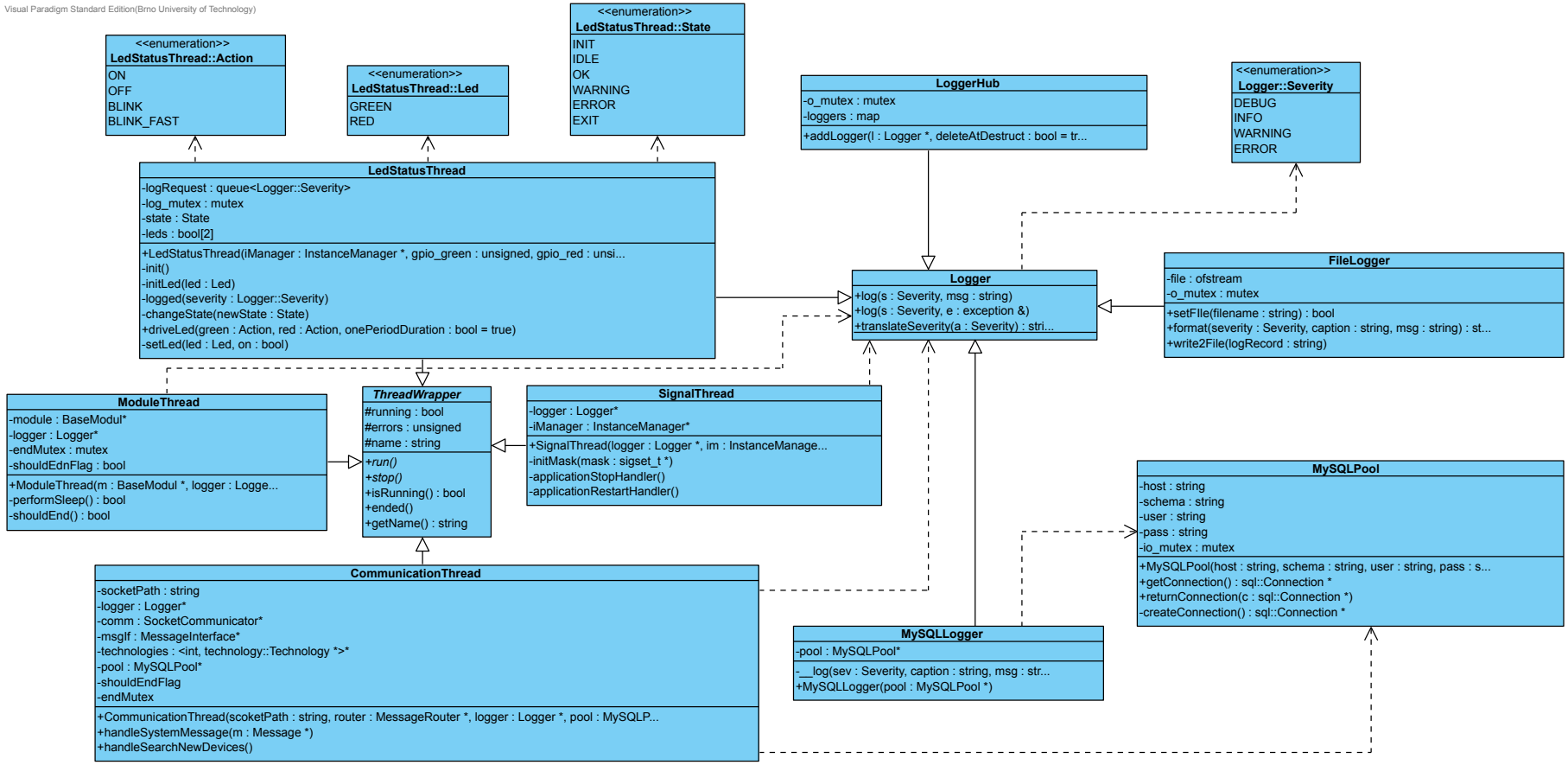
Poslední diagram zobrazuje komunikaci mezi ovladači pomocí třídy `MessageInterface`, které běží v samostatných vláknech a komunikaci aplikace s jinými procesy skrz Unixový soket. Za zmínku stojí především třída `CommunicationThread`, která je spuštěna v samostatném vláknu a jejímž úkolem je překládat komunikaci s jinými procesy na vnitřní komunikaci a přijatou zprávu od jiného procesu poslat příslušnému adresátovi, kterým může být okruh nebo ovladač. V jiném případě se může jednat o systémovou zprávu. Takovou zprávu dále neposílá a snaží se na ni reagovat.



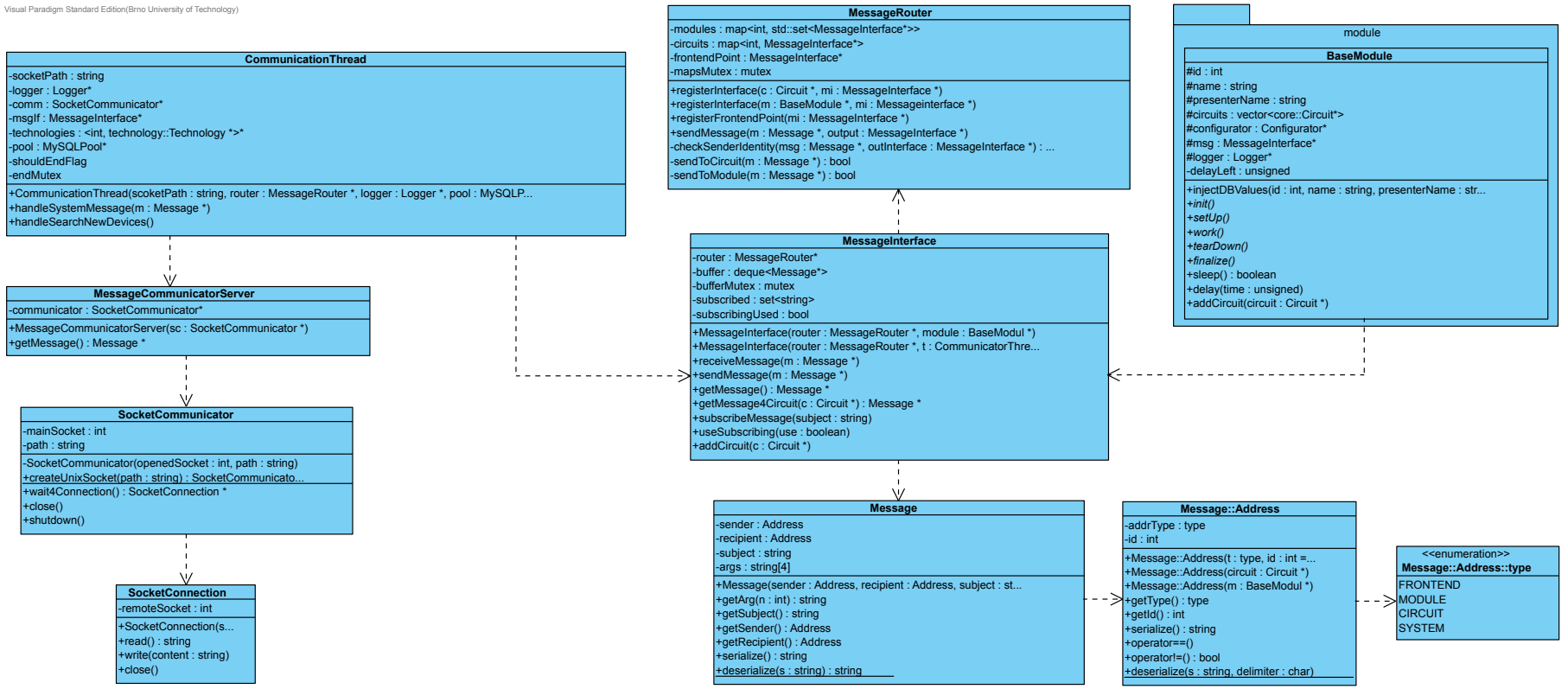
Obrázek D.1: 1. část – okruh a závislosti na rozšiřitelných částech



Obrázek D.2: 2. část – vytváření rozšiřitelných částí InstanceManagerem



Obrázek D.3: 3. část – druhy vláken a rozhraní pro logování

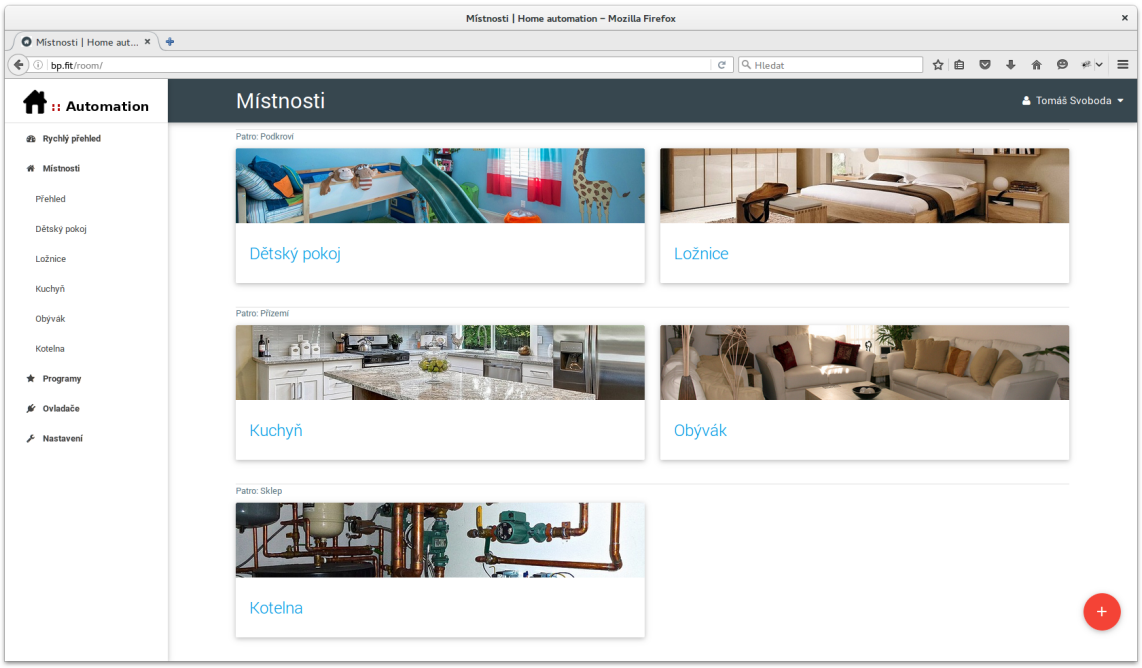


Obrázek D.4: 4. část – rozhraní pro komunikaci mezi vlákny a komunikaci mezi procesy

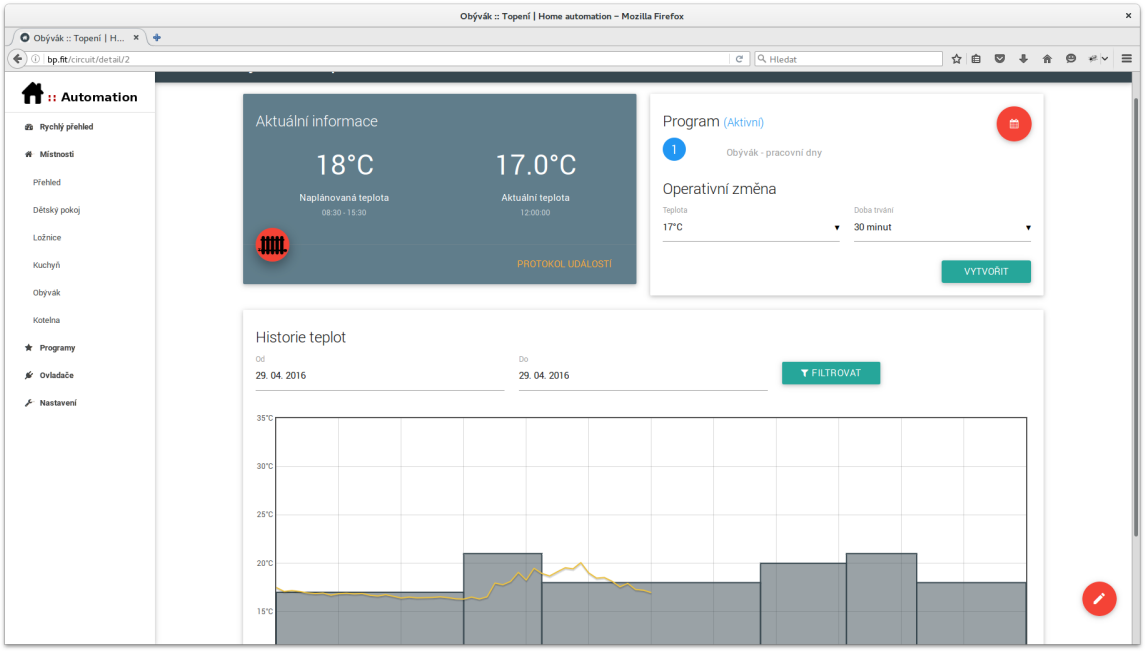
Příloha E

Snímky obrazovek ovládací aplikace

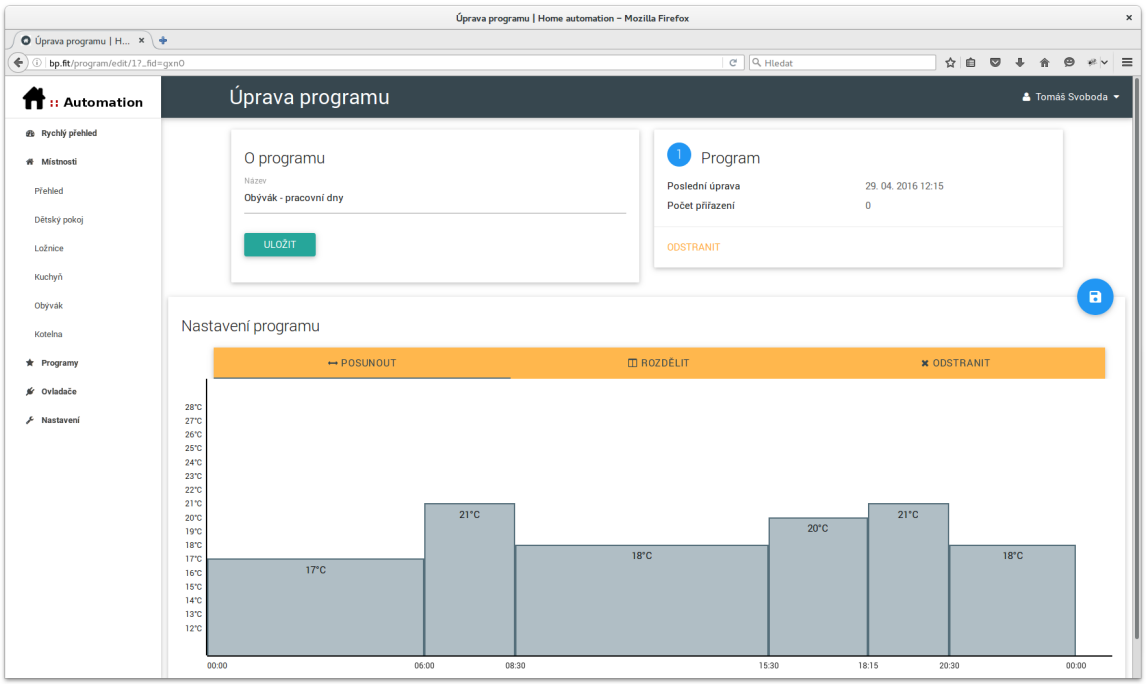
V této příloze bude uvedeno několik snímků obrazovek z ovládací aplikace. Více snímků obrazovek lze nalézt na přiloženém CD.



Obrázek E.1: Přehled všech místností seřazených podle patra



Obrázek E.2: Detail okruhu, který řídí vytápění



Obrázek E.3: Tvorba programu teploty

Příloha F

Obsah CD

```
.
├── src ..... zdrojové kódy
│   ├── automation ..... řídicí služba
│   └── www ..... webová ovládací aplikace
├── instalace ..... instalační soubory
│   ├── mysql
│   │   ├── createDB.sh
│   │   └── ...
│   ├── install.sh
│   ├── home_automation.tgz
│   ├── start1wire.sh
│   └── 000-default.conf
├── doc ..... generovaná programová dokumentace řídicí služby
│   ├── html
│   │   ├── index.html
│   │   └── ...
│   └── latex
│       ├── refman.pdf
│       └── ...
├── pcb ..... zdrojové soubory navržených modulů
│   ├── 1W_minimalistic_bus_driver
│   ├── 8_relay_board
│   ├── display_board
│   ├── sensor_board
│   ├── 1W_bus_driver
│   ├── main_board
│   └── photo ..... fotografie vyrobených modulů
│       └── ...
├── uml
│   ├── class_diagram ..... kompletní diagram tříd řídicí aplikace
│   └── ...
├── printscreens ..... snímky obrazovek ovládací aplikace
│   └── ...
├── document ..... zdrojové soubory technické zprávy v LaTeXu
└── document.pdf ..... technická zpráva
```