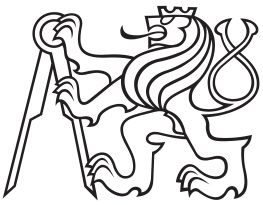
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Chytrá domácnost na míru

Vojtěch Lukeš

Studijní program: Kybernetika a robotika
Vedoucí práce: Ing. Petr Novák Ph.D.
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Lukeš** Jméno: **Vojtěch** Osobní číslo: **466346**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Chytrá domácnost na míru

Název bakalářské práce anglicky:

Custom Smart-Home

Pokyny pro vypracování:

Stále více se rozmáhají tzv. Chytré domácí systémy. S větší dostupností a klesající cenou potřebného vybavení se teprve v současné době znatelně posouvají z původní „profi“ oblasti až do zcela „amatérských“ řešení. Avšak tvorba zcela vlastních nejen senzorů / aktuátorů, ale i řídicí části systému přináší znatelně odlišné požadavky na propojení, strukturu, konfiguraci, ovládání, činnost a samozřejmě i celkovou cenu systému. Cílem práce je návrh struktury / částí „Chytrého domácího systému“ jako univerzálního základu pro zcela individuální, až amatérské, použití.
1) Prostudujte typy (chytrých) domácích systémů umožňující připojení zcela vlastních senzorů / aktuátorů, tvorby vlastních protokolů a zejména možnosti konfigurace i na libovolně nízké úrovni.
2) Zhodnoťte výhody běžných komunikačního kanálu / protokolů (LAN, CAN, RS485, WiFi, …) a pro vybrané vytvořte příklady komunikace pro některé často využívané HW platformy (STM32, Arduino, …).
3) Navrhněte centrální jednotku (použitý HW a SW) s ohledem na cenu, jednoduchost, univerzálnost, snadnost komunikace a schopnou vykonávat základní činnost systému.
4) Navrhněte možnosti několika úrovňové konfigurace systému z pohledu různých typů uživatelů (od grafické pro běžného uživatele až po programovou pro odborníka / nadšence).
5) Navrhněte vizualizaci stavu a ovládání systému (přes tablet, mobil, …) a to případně i bez nutnosti instalace fyzických ovládacích prvků (úspora za vedení, vypínače, snadná aktualizace, efekt, …, nižší cena).
6) Na několika vzorových příkladech demonstrujte vhodnost vytvořeného návrhu / systému.

Seznam doporučené literatury:

[1] WWW stránky nalezených / obdobných projektů, manuály a další relevantní informace
[2] Price Mark, C# 8.0 and .NET Core 3.0 - Modern Cross-Platform Development, Packt, 2019
[3] Noviello Carmine, Mastering STM32, LeanPub, 2017
[4] Chin Robert, A DIY SmartHome Guide (Arduino, ESP8266, Android), McGraw-Hill, 2020
Další potřebné materiály poskytne vedoucí práce.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Novák, Ph.D., Analýza a interpretace biomedicínských dat FEL

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

Datum zadání bakalářské práce: **06.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Petr Novák, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Děkuji svému vedoucímu práce Ing. Petru Novákovi Ph.D. za konzultace, nápady a veškerou ostatní pomoc při řešení bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 20. května 2021

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 20, 2021

Podpis/Signature

Abstrakt

Práce se zabývá návrhem a tvorbou systému domácí (dohledové) automatizace s důrazem na snadné přidávání vlastních experimentálních senzorových a akčních koncových modulů založených na mikroprocesorových programovacích platformách Arduino a ARM-STM32. Centrální jednotka umožňující sběr dat ze senzoru, vykonávání řídícího algoritmu a zápis dat do akčních členů je implementována jako přenositelný program pomocí .NET/C#. Pro komunikaci centrální jednotky s periferiemi byly vybrány běžně dostupné komunikační technologie UART, Ethernet, WiFi, Bluetooth Classic a Bluetooth Low Energy. Součástí práce je rovněž uživatelská vizualizace a konfigurace celého systému.

Klíčová slova: Chytrá domácnost, Domácí automatizace, Dohledové systémy, Arduino, STM32, .NET

Abstract

This thesis describes design and implementation of home (supervision) automation system. The main goal is to enable simple adding custom experimental sensor- and actuator- end modules. These are meant to be created using microcontroller programming platforms Arduino and ARM-STM32. The central unit is implemented as multiplatform application using .NET/C#. Its purpose is to collect data from sensors, to execute control algorithm and to write data back to actuators. Various commonly used communication technologies (UART, Ethernet, WiFi, Bluetooth Classic, Bluetooth Low Energy) were used to connect the central unit to peripherals. Integral parts of the thesis are also user visualisation and configuration applications.

Keywords: Smart home, Home automation, Supervision systems, Arduino, STM32, .NET

Title translation: Custom Smart-Home

Obsah

Kapitola 1

Úvod

V současné době se označení „Chytrá domácnost“ používá skutečně ve velkém rozsahu a téměř na cokoliv. Pod tímto pojmem si lze představit téměř vše od pouhého rozsvícení světla pohybovým senzorem až po sofistikovaný projekt, kde zásahy uživatele jsou omezeny na minimum a objekt je řízen zcela autonomně umělou inteligencí.

Jelikož se chytrá domácnost stává jakýmsi fenoménem dnešní doby, vzniká v této oblasti stále více projektů a nových řešení. Uživatelé neznalí elektroniky volí nejčastěji řešení od renomovaných průmyslových firem s dodáním „na klíč“, naopak ti pokročilejší sáhnou po různých open-source variantách, případně se pustí do realizace vlastního amatérského řešení, využívajícího jak komerčně dostupné komponenty, tak i zcela vlastní experimentální senzory a akční členy.

Právě ty posledně zmíněné domácí systémy dávají vzniknout zcela nové oblasti představující spíše dohledové systémy pro starší, osamocené osoby, jenž v současné době znatelně přibývá. I v tomto případě lze právem hovořit o chytré domácnosti, schopné například připomenout nedovřené dveře, upozornit na dlouho tekoucí vodu nebo i automaticky vypnout zapomenutý spuštěný vařič. Takový systém může dokonce sledovat i do jisté míry zdravotní stav dané osoby, například tělesnou aktivitu, tepovou či dechovou frekvenci. V tomto případě je těžiště funkce systému nikoliv v přizpůsobování okolí (zdravému) uživateli, ale v kontrole (i částečně indisponovaného) uživatele.

V případě méně standardních aplikací včetně právě dohledových systémů se stávající průmyslová a open-source řešení ukazují být z různých důvodů nepříliš vhodná, nebo dokonce přímo nepoužitelná.

Tato skutečnost motivuje vznik jednoduchého experimentálního systému, který není ucelený ani uzavřený, ale spíše se jedná o jakousi kostru, připravenou pro vlastní doplnění o požadované funkce a dokončení podle cílového použití.

Kapitola 2

Stávající řešení, motivace

V této části bude podrobněji zanalyzovaná aktuální situace na trhu v oblasti chytré domácnosti, budou představeny přístupy nejen jednotlivých velkých komerčních společností, ale rovněž i menších komunit. Dále bude posouzena vhodnost těchto přístupů pro specifické amatérské a experimentální použití, zejména pak s ohledem na domácí již zmíněný dohled.

Hlavní sledovaná kritéria budou tedy zejména:

- Otevřenost, tedy možnost přidávání vlastních senzorů a akčních členů.
- Možnost konfigurace na úrovni signálů, tedy tvorby vlastních funkčních bloků.
- Přímočarost a intuitivnost konfigurace pro méně znalé uživatele.
- Možnosti prezentace dat uživateli, vzdálený přístup a přívětivost.

2.1 Průmyslová automatizace

Společnosti, zabývající se primárně průmyslovou automatizací, se rovněž snaží proniknout do oblasti automatizace domácí. Jejich řešení, odvozená často od průmyslových programovatelných automatů PLC, mají výhodu v maximální možné spolehlivosti a robustnosti. Tato robustnost je však pro účely běžné chytré domácnosti až zbytečná, navíc je často vykoupena velmi vysokou cenou. Ta činí tyto systémy hůře dostupné i pro velké developerské projekty a až zcela nedostupné pro experimentální a amatérské využití. Uplatnění tak mohou nacházet spíše v komerčních a průmyslových budovách, kde cena nehraje takovou roli.

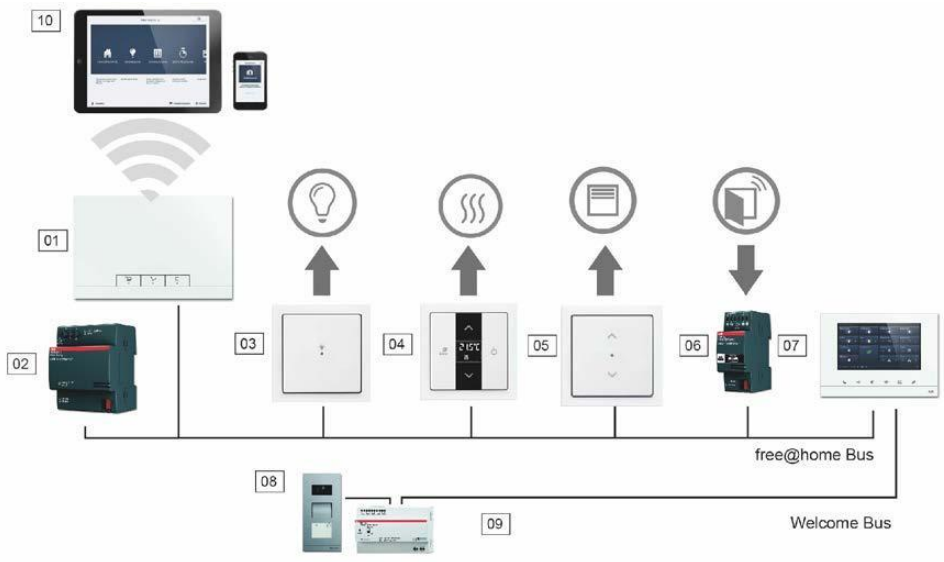
Hardwarové vybavení bývá obvykle na vysoké úrovni, ovšem systémy jsou v podstatě zcela uzavřené a s připojením koncových zařízení vlastní výroby se nepočítá. Rovněž programování a konfigurace těchto zařízení není pro neodborníky vždy jednoduchá záležitost a může vyžadovat specializované softwarové nástroje.

Příklady společností:

ABB [1], Siemens [2], TECO [3]

Typické výhody a nevýhody:

- \+ Spolehlivost, robustnost, dobrá/dlouhodobá podpora
- \+ Velký výběr kvalitních komponent
- \+ Možnost programování na úrovni signálů
- \- Velmi vysoká cena
- \- Uzavřenost na vlastní komponenty/hardware
- \- Konfigurace často vyžaduje specializovaný software



Obrázek 2.1: Topologie instalace free@home od ABB [4]

2.2 Globální IT společnosti

Velcí nadnárodní softwaroví giganti naopak cílí na běžného koncového uživatele, vyvíjí tedy zejména uživatelské mobilní, případně webové aplikace a investují nemalé prostředky do výzkumu v oblasti umělé inteligence. Hardware se obvykle omezuje na různé hlasové asistenty s větší či menší podporou komerčních výrobků třetích stran. Systémy jsou uzavřené a nepočítá se zde s připojováním vlastních zařízení a již vůbec ne s programováním na úrovni signálů a proměnných.

Tato možnost je vhodná spíše na efektní demonstraci problematiky, případně pro ovládání několika málo běžných domácích zařízení (chytrá žárovka, hlavice radiátoru…).

Příklady projektů:

Apple HomeKit, Amazon Alexa, Google Nest

Typické výhody a nevýhody:

+ Líbivé grafické rozhraní
+ Uživatelská přívětivost
+ Ovládání hlasem, umělá inteligence
- Pouze omezená podpora komerčních zařízení
- Nemožnost konfigurace na nízké úrovni

Obrázek 2.2: Hlasový asistent Amazon Alexa [5]

2.3 Nově vzniklé společnosti

Společnosti, zabývající se výhradně problematikou domácí automatizace, vyvíjí jak hardwarové komponenty, tak specializovaný software pro vizualizaci a konfiguraci. Komponenty mohou být levnější, neboť na ně nejsou kladeny takové požadavky jako v případě průmyslového nasazení, pro amatérské využití však mohou být stále nedosažitelné.

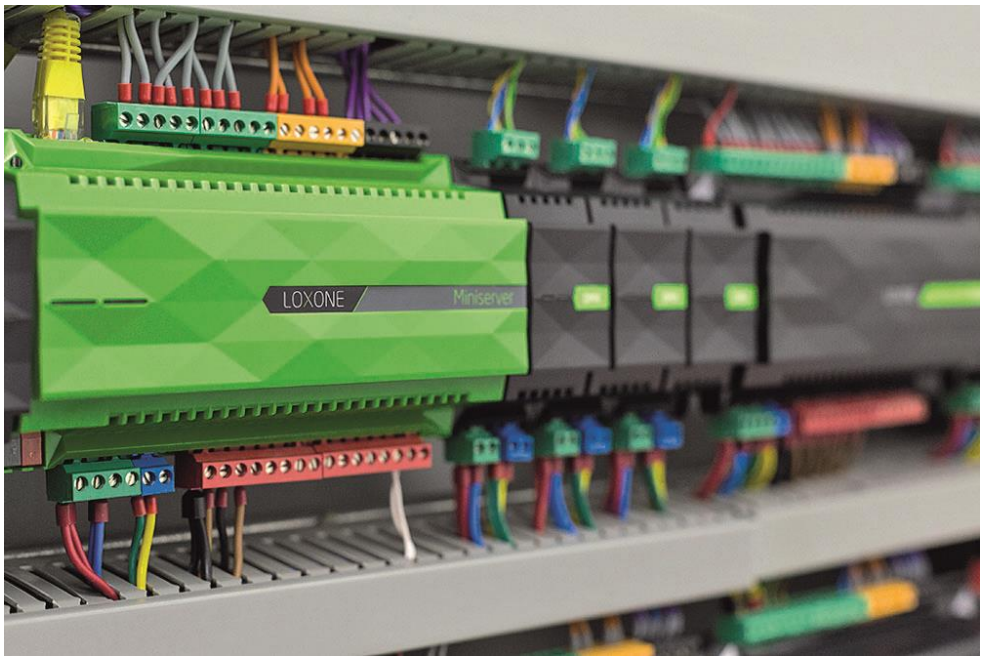
Integrace zařízení třetích stran bývá možná, ovšem vytvořená rozhraní nemusí být dostatečně univerzální. Stejně tak jsou omezené možnosti konfigurace.

Příklady společností:

Loxone [6], Evon [7], HomeSeer [8]

Typické výhody a nevýhody:

+ HW i SW optimalizované pro domácí automatizaci
+ Intuitivní konfigurace (zpravidla grafická)
+ Příznivější cena než u průmyslových systémů
- Stále vysoká cena pro široké amatérské použití
- Omezená podpora vlastních zařízení
- Nemožnost zcela vlastního programování/konfigurace



Obrázek 2.3: Loxone Miniserver s přídavnými moduly [9]

2.4 Open-source řešení

Síla převážně softwarových projektů, vyvíjených a spravovaných jednotlivci či komunitami nadšenců, spočívá v jejich otevřenosti a dostupnosti v podstatě komukoli. Podporu jednotlivých komponent třetích stran do systému přidávají jednotliví přispěvatelé. Výhodou je tak možnost takřka neomezené rozšiřitelnosti a modifikace.

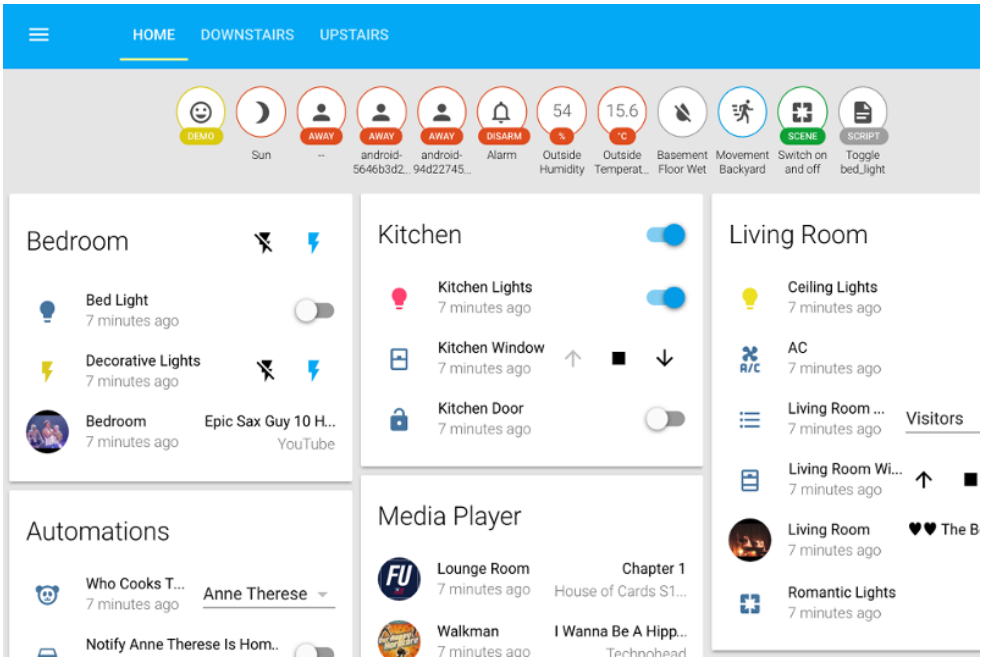
Bohužel snaha o vyrovnání se komerčním projektům a poskytnutí podpory co nejširšího spektra vyráběných zařízení vede k jisté pře-kombinovanosti a zbytečné složitosti. Také konfigurace těchto systémů nebývá tak přímočará jako v případě komerčních produktů. Častěji se zde setkáváme s konfiguračními skripty, jejichž tvorba vyžaduje dostatečné pochopení a přijetí celé architektury systému. Integrace nestandardních způsobů využití může být poměrně obtížná. Jelikož jde o nekomerční systémy, není zde snaha o produkci zisku a tím rovněž není hlavním cílem ani snaha o masové rozšíření.

Příklady projektů:

HomeAssistant [10], OpenHAB [11], Domoticz [12], FHEM [13]

Typické výhody a nevýhody:

+ Otevřenost, rozšiřitelnost
+ Mnoho funkcí, navíc stále rozšiřované
+ Podpora širokého spektra zařízení jiných výrobců
- Zbytečná složitost pro jednoduché aplikace
- Méně intuitivní konfigurace



Obrázek 2.4: Ukázka vizualizace v Home Assistant [14]

Kapitola 3

Cíl práce

Jak je z předchozího přehledu zřejmé, tak možnosti pro méně standardní, výzkumné, zcela amatérské, nebo dokonce experimentální využití nejsou nikterak velké. Komerční systémy jsou drahé a uzavřené, open-source projekty jsou často překombinované a tím pádem složitě použitelné pro jednoduché a nestandardní nasazení.

Mezi hlavní cíle této práce pro tvorbu domácího chytrého systému určeného zejména pro dohled bude tedy patřit:

- Vytipovat některé z běžných komunikačních kanálů a vytvořit příklady komunikace pro často využívané HW platformy. Výstupem bude komunikační knihovna, využitá budoucím tvůrcem zařízení pro snadnou komunikaci s centrálou.
- Navrhnout a vytvořit centrální jednotku s ohledem na cenu, jednoduchost, univerzálnost, snadnost komunikace a schopnost vykonávat základní činnosti systému.
- Navrhnout konfigurační aplikaci, umožňující konfiguraci jak graficky na vysoké úrovni pro méně znalé uživatele, tak naopak i na nízké úrovni pro experty.
- Implementovat uživatelskou vizualizaci pro web, tablet, mobil s možností běžného ovládání systému.
- Fyzicky realizovat prototypy některých zařízení a vytvořit několik vzorových příkladů pro použití navrženého systému.

Naopak se práce nebude zabývat těmito oblastmi:

- Zabezpečení komunikací proti hackerským útokům.
- Diagnostikou sítí.
- Odolnosti vůči výpadkům.
- Ochraně osobních údajů.
- Tvorbě plné dokumentace a koncovému nasazení.

Výsledný systém musí být co možná nejjednodušší a nejpřímočařejší, využitelný právě pro amatérská a výzkumná nasazení se zaměřením speciálně pro domácí dohled. S ohledem na charakter použití se nebude jednat o hotový uzavřený systém, připravený k okamžité aplikaci, ale spíše o jakousi kostru, určenou k doplnění o vlastní funkce potřebné v konkrétním nasazení.

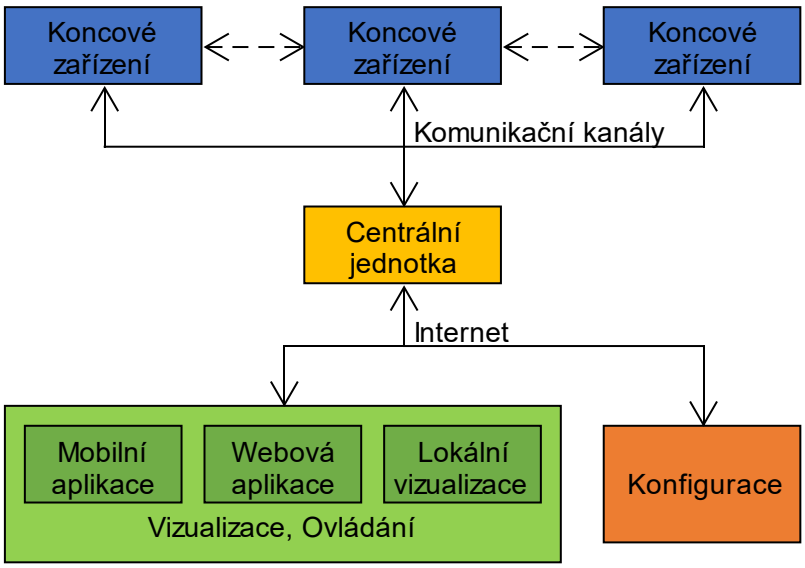
Kapitola 4

Obecný návrh, výběr technologií

I když je zde zaměření spíše na dohledový než na řídicí systém, tak jeho prezentovaný návrh nemůže být zásadně odlišný od těch již existujících. Vždy se jedná o nějaká koncová zařízení komunikující s centrální jednotou po různých sběrnicích neboli komunikačních kanálech. Centrální jednotka je vždy realizována nějakým typem počítače, případně řídicím automatem tvořícím jádro celého systému. Ovšem HW část není vše, požadovanou činnost zařízení tvoří hlavně obsažený SW. Centrální jednotka tedy musí obsahovat nějaký řídicí program, ten musí být možno konfigurovat a vytvořit tak celkové chování systému. Rovněž musí být přítomno nějaké vizualizační rozhraní pro interakci s uživatelem (kontrolu činnosti a ovládání systému). Od této struktury se tedy nijak neliší ani zde navrhovaný domácí systém.

Obecná struktura našeho domácího řídicího / dohledového systému, pracovně nazvaného DotHome, je schematicky znázorněna na obrázku 2.1. Systém se skládá z těchto hlavních částí:

- Koncová zařízení – Snímací a akční jednotky (HW) v systému rozmístěné do domácnosti a připojené pomocí nějakého komunikačního kanálu/sběrnice.
- Komunikační kanál – Vedení/sběrnice zajišťující přenos dat mezi koncovými zařízeními a centrální jednotkou (v obou směrech).
- Centrální jednotka – Mozek, tedy hlavní neboli řídicí část systému. Zpracovává vstupy z koncových zařízení, vykonává zadaný řídící algoritmus a následně zapisuje své výstupy zpět do koncových zařízení.
- Vizualizace/Ovládání – Prezentace stavu/dat ze systému uživateli a současně i možnost (základního) ovládání stavu systému.
- Konfigurace/nastavení – Nástroj/pomůcka pro editaci chování centrální jednotky. Umožňuje vytvoření požadovaného chování systému.



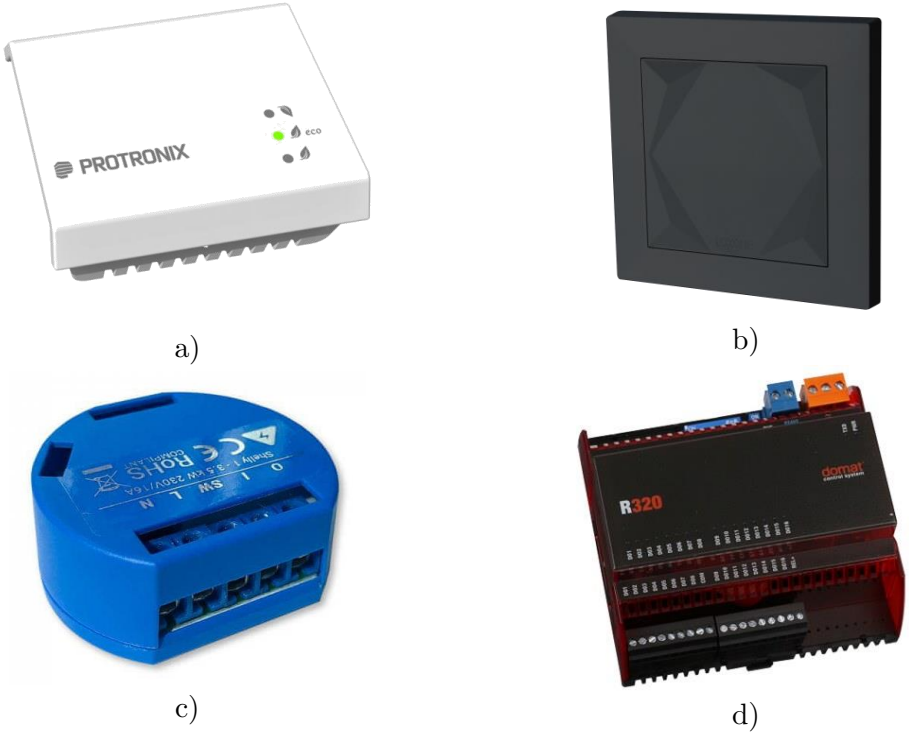
Obrázek 4.1: Schéma řídicího systému navržené chytré domácnosti

Jednotlivé zmíněné součásti budou podrobně popsány dále v samostatných částech.

4.1 Koncová zařízení

Koncovým zařízením je rozuměno (nejčastěji) fyzické zařízení, obsahující skutečné vstupy – senzory (např. teploty, vlhkosti, CO_2, pohybu, průtokoměr, tlačítko vypínače) a skutečné výstupy – akční členy (např. světlo, relé, ventil topení, pohon žaluzií). S centrální jednotkou se vyměňují data pomocí různých komunikačních kanálů. Koncové zařízení může obsahovat i několik senzorů současně, příkladem může být kombinovaný senzor teploty, vlhkosti a CO_2 s komunikačním rozhraním Modbus.

Tato práce se však zaměřuje na tvorbu zejména vlastních (experimentálních) koncových zařízení s použitím různých běžně používaných programovatelných mikrokontrolerů (AVR – ATMEGA328, ATMEGA2560; ARM – STM32xxxx, ESP8266/ESP32; …). Cílem práce je poskytnout pouze vhodné komunikační rozhraní v podobě programové knihovny v jazyce C pro co největší univerzálnost. Zejména v experimentálních systémech jsou měřené signály zcela závislé na konkrétním cílovém projektu. Avšak přenos takto získaných dat neboli komunikace je obdobná. Implementace skutečných vstupů a výstupů koncového zařízení, tedy reálných signálů, bude proto zcela ponechána na budoucím uživateli. Tato skutečnost je stěžejní myšlenkou zde vytvářeného domácího systému.



a) b) c) d)

Obrázek 4.2: Příklady některých komerčních koncových zařízení:

a) Čidlo teploty, vlhkosti a CO_2 Protronix [15]
b) Nástěnný vypínač Loxone [6]
c) Reléový výstup Shelly s WiFi čipem ESP8266 [16]
d) Univerzální vstupně-výstupní modul Domat [17]

V uvažovaných koncových zařízeních se počítá primárně s využitím dvou nejrozšířenějších procesorových produktů, což jsou:

- Platforma Arduino (oblíbená zejména mezi ne zcela programátorsky zdatnými uživateli), umožňující snadné a univerzální programování nepřeberného množství mikrokontrolerů z různých rodin a od různých společností a to v jazyce C++.
- Mikrokontrolery STM32 od společnosti ST Microelectonics (oblíbené naopak mezi již zdatnějšími vývojáři), pro jejichž programování budou použity některé snadno dostupné vývojové desky a programové knihovny HAL v jazyce C.

4.2 Komunikační kanály

Komunikační kanál je prostředek pro přenos dat mezi koncovými zařízeními a centrální jednotkou. Pokud je to účelné, mohou koncová zařízení komunikovat i přímo mezi sebou bez prostředníka, tedy i bez centrální jednotky. Například vypínač může přímo rozsvítit žárovku. Výhodou tohoto přístupu je vyšší robustnost a odolnost vůči výpadku centrální jednotky, naopak nevýhodou je mnohem složitější konfigurace a nutnost správné synchronizace dat. Právě z těchto důvodů důvodu nebude tato možnost dále uvažována.

Pod pojmem komunikační kanál je třeba rozumět vždy ucelený balík technologií, umožňující komunikaci mezi jednotlivými koncovými uzly/účastníky, nikoli pouze fyzický vodič/drát. Tedy všechny relevantní vrstvy OSI modelu od fyzického média až po reprezentaci dat aplikační vrstvou. Příkladem takového kanálu může být protokol MQTT [18], využívající transportní vrstvu TCP a fyzickou vrstvu Ethernet. Nebo protokol Modbus RTU [19], provozovaný na fyzické lince RS485.

Komerční projekty využívají především průmyslové (velmi spolehlivé) komunikační sběrnice od těch skutečně sofistikovaných jako je KNX, přes celkem rozšířené jako například CAN, až po velmi obecné a skutečně základní jako je RS485. Tato práce se však zaměřuje naopak na tvorbu vlastních zařízení s možností připojení pomocí komunikačních technologií běžně dostupných v obecné výpočetní technice. Podpora průmyslových sběrnic může být samozřejmě přidána kdykoliv později v případě potřeby, ale není hlavním cílem práce.

Rovněž je potřeba vzít v úvahu následující skutečnost. Pokud se vytváří řídicí domovní systém v nově stavěném objektu, jsou zde vhodné podmínky pro vytvoření fyzického vedení (kabely). Naopak pokud se vytváří dohledový systém ve stávajícím objektu, je často nutné použít bezdrátový přenos.

Pro implementaci byly vybrány následující komunikační technologie (kanály):

- Seriová linka a její různé modifikace (UART s případným USB převodníkem, RS485, RS232, …) – Vhodné zejména pro lokálně připojená zařízení (čtečky NFC, ovládací tlačítka, různé signalizace, …).
- Ethernet, WiFi (UDP, TCP) – Pro převážně vzdálená zařízení, i v jiných místnostech/patrech (vypínače, senzory, výstupní moduly, …).
- Bluetooth (Classic), Bluetooth Low Energy – Zejména pro bateriově napájená zařízení ve vzdálenosti na jednu maximálně dvě místnosti (senzory tepu, dechu, …).

Pokud lze na komunikační médium nahlížet jako na proud dat s dostatečně vysokou rychlostí, je vhodné, aby použitý protokol byl ve všech případech stejný (z důvodu vzájemné kompatibility a samozřejmě zjednodušení vývoje) a to ideálně textový (z důvodu snadnějšího ladění). Pokud bude použit přenosový kanál s opravdu nízkou přenosovou rychlostí, tak je vhodné vytvořit rovněž binární alternativu.

4.3 Centrální jednotka

Centrální jednotka představuje mozek celého systému. Přijímá data z koncových zařízení (nejčastěji senzorů), vykonává řídící algoritmus a zapisuje vytvořená data zpět do koncových zařízení (akčních členů). Zpracovávaná data jsou rovněž poskytována uživateli

ve formě vhodné vizualizace, nejčastěji formou nějakých obecných grafů, bar-grafů nebo stavových signalizací.

Centrální jednotku tvoří jednak HW (počítač), avšak důležitější je SW v ní obsažený. Ten zajišťuje požadovanou činnost celého systému. Může se jednat o program běžící na běžném (i domácím) počítači nebo na samostatném fyzickém zařízení (průmyslový počítač, Raspberry PI). Centrální jednotka může rovněž obsahovat lokálně připojené periferie (vstupy/výstupy – rozumíme např. GPIO Raspberry PI).

Vzhledem k očekávanému nasazení v libovolných, tedy i nepřipravených domácnostech je potřeba, aby se jednalo o program, spustitelný i na běžných domácích počítačích a obdobných zařízeních s různými operačními systémy (například i tablet).

Pro splnění těchto požadavků byl jako vývojová platforma zvolen objektový programovací jazyk C# a běhové prostředí .NET 5.0 (.NET Core). Jedná se o moderní, zcela multiplatformní framework, obsahující užitečné nástroje pro komunikaci (SignalR, Web API), vizualizaci (Blazor), ukládání dat (Json) i pro zajištění běhu vlastního programu (Dependency Injection, Configuration, Reflection).

Vývoj a demonstrace budou vytvořeny na běžném PC s OS MS/Windows 10 (v experimentech zřejmě nejčastější použití). Počítá se však i s nasazením na jednodeskové počítače typu Raspberry PI (Linux, MS/Windows IoT).

4.4 Vizualizace

Pokud by domácí systém nemusel nikterak komunikovat s uživatelem a vše by zajistil zcela sám, jednalo by se o ideální stav. K této podobě jsme však stále ještě daleko. Z toho důvodu je potřeba vytvořit rozhraní, pomocí něhož bude uživatel informován nejen o tom, co se v systému děje, tedy jaký je jeho aktuální stav, ale současně bude moci chování systému nějakým způsobem usměrňovat/upravovat. Za tímto účelem je potřeba vytvořit signalizační (směrem ze systému k uživateli) a ovládací (směrem od uživatele do systému) prvky.

Takového uživatelské rozhraní může být lokální (tablet zabudovaný ve zdi) nebo vzdálené (mobilní/webová aplikace). Uživateli poskytuje grafické rozhraní pro sledování a ovládání jednotlivých procesů v domě/systému. Typickými prvky tohoto rozhraní budou tlačítka, přepínače, upozornění, termostaty nebo grafy některých měřených veličin.

Spojení takovýchto vizualizačních komponent je s centrální jednotkou zajištěno přes internetovou síť, a to buď přímo nebo zprostředkovaně přes cloudové úložiště. Lokální vizualizace (již zmíněný tablet zabudovaný ve zdi) by teoreticky mohla být připojena i jiným komunikačním kanálem jako je například Bluetooth nebo USB. V praxi se však

z důvodu unifikace používá stejné rozhraní jako pro vzdálenou vizualizaci. Stejný přístup bude použit i ve zde uvedeném návrhu.

V práci bude uvažována primárně vizualizace pomocí webové stránky nevyžadující instalaci na cílovém zařízení. V případě mobilní aplikace pro systémy Google/Android nebo Apple/iOS je možné s výhodou použít moderní technologii Progresivní webové aplikace (PWA) umožňující instalaci webové aplikace přímo na cílovém zařízení, a tedy přístup k některým funkcím telefonu/tabletu. Pro chytrou domácnost jsou velmi užitečnou takovou funkcí tzv. Push notifikace. Jsou to notifikace ze serveru (centrální jednotky) informující uživatele o události i při vypnuté vizualizační aplikaci.

Pro tvorbu takovéto webové aplikace se výborně hodí nová technologie Blazor. Jde o součást frameworku ASP.NET Core umožňující kombinovat webové stránky (HTML, CSS) s výkonným kódem (C#), a to jak na straně serveru, tak i ve webovém prohlížeči uživatele.

Po vzoru komerčních a open-source projektů by se prvky vizualizace měly zobrazovat automaticky na základě vytvořené konfigurace systému.

4.5 Konfigurace

Každý jakkoli obecný systém je nutno vždy nějak nakonfigurovat, tedy přizpůsobit pro cílové použití. Konfigurace přestavuje proces nastavující/vytvářející celkovou činnost systému podle požadavků uživatele. Taková konfigurace může mít několik úrovní a skládat se z několika (i oddělených) částí.

Konfigurace je tedy v podstatě další uživatelské rozhraní, avšak typicky určené pouze pro administrátora, dostatečně znalého struktury daného systému. Týká se typicky pouze centrální jednotky a musí zahrnovat následující:

- Jaká koncová zařízení a jakým způsobem (komunikačním kanálem) jsou k centrální jednotce připojena.
- Jaké skutečné vstupy/výstupy koncových zařízení jsou definovány pro použití v centrální jednotce. Případně jakou mají datovou reprezentaci.
- Řídící algoritmus, tedy pravidla, podle nichž se vytváří chování celého systému. Tato pravidla mohou být zapsána různým způsobem (rovnice, skript, program, graf, …).
- Jaká data a jakým způsobem budou prezentována uživateli ve vizualizaci. Ne všechny hodnoty je potřeba zobrazovat, a ne všechny hodnoty je povoleno nastavovat. Úroveň vizualizace může být rovněž volena podle odborné znalosti uživatele.

Pod pojem řídící algoritmus rozumíme například soubor pravidel, kdy v případě detekování trvalého průtoku vody je po deseti minutách zobrazeno upozornění ve vizualizaci. Po dalších deseti minutách se spustí hlasitý alarm, a pokud nedojde k potřebné reakci uživatele, zavře se automaticky uzávěr vody.

Přístupy ke konfiguraci mohou být různé. Od jednoduchého grafického programování (intuitivní, avšak omezené) přes ruční psaní skriptů (již pro zasvěcené) až po programování skutečně nativního kódu (pro odborníky).

Konfigurace na vysoké úrovni by měla být dostatečně přímočará a intuitivní i pro méně znalé uživatele. Z toho důvodu bude použit přístup jako v průmyslových a komerčních projektech, tedy grafická konfigurace pomocí funkčních bloků. Díky otevřenosti systému bude ponechána možnost libovolně editovat existující bloky nebo si i naprogramovat bloky zcela vlastní.

Konfiguraci lze považovat za jednu z nejsložitějších činností při nasazování (nového) systému. Z tohoto důvodu konfigurační aplikace nebude implementována jako obecná webová aplikace, ale jako oddělený jednoúčelový desktopový program. Již dříve zmíněná platforma .NET 5.0 nabízí pro tyto účely pokročilý grafický subsystém nazvaný WPF, využívající výhod značkovacího jazyka XAML (sémantický popis) a Data Bindingů (snadné spojení GUI prvků s programovým kódem). Nevýhodou je v současnosti podpora pouze pro OS MS/Windows. Verze .NET 6.0 by však již měla podporovat multiplatformní variantu nazvanou MAUI. V aktuálně vytvářeném návrhu bude použita velmi podobná knihovna třetí strany s názvem AvaloniaUI (disponující téměř stejnou architekturou a možnostmi).

Kapitola 5

Koncová zařízení a komunikace

Jak již bylo řečeno, jedná se zejména o malá fyzická zařízení s připojenými senzory/aktuátory, interagující s reálným světem a komunikující s centrální jednotkou. Takovéto typické koncové zařízení se skládá z následujících částí:

- Vstupní a výstupní (analogové/digitální) obvody pro skutečné měření a řízení okolí. Ty jsou zcela závislé na cílovém použití, a proto nejsou součástí této práce.
- Čtení a zpracování dat z těchto vstupů/výstupů. Rovněž velmi závislé na cílovém použití. V práci jsou definovány pouze jejich datové typy pro přenos do centrální jednotky.
- Komunikace (oboustranná) s centrální jednotkou. Velmi často zcela nezávislá na cílovém projektu, a proto je cílem této práce.

Tato koncová zařízení budou typicky realizována konkrétním uživatelem pomocí programovatelných mikrokontrolerů s použitím nejčastěji platforem Arduino a STM32. V rámci této práce budou tedy pouze definovány nutné datové typy, komunikační protokoly a vytvořeny příslušné programové knihovny pro zmíněné cílové procesorové platformy.

Poznámka: Vstupy (data koncového zařízení) budeme označovat jako *RValues* (hodnoty čtené ze senzorů a zasílané do centrální jednotky) a výstupy jako *WValues* (hodnoty přijímané z centrální jednotky a zapisované na výstupy). Později v centrální jednotce se vstup *RValue* koncového zařízení stane výstupem a výstup *WValue* vstupem funkčního bloku představujícího příslušné koncové zařízení. Tedy:

- RValue(s) – Data čtená z koncového zařízení (hodnoty ze senzorů).
- WValue(s) – Data zapisovaná do koncového zařízení (hodnoty pro aktuátory).

5.1 Použité datové typy

V oblasti jednoduchých HW procesorů (AVR, ARM-CortexM0, …) není vždy snadné pracovat s mnoha různými datovými typy jako v případě vysokoúrovňových programo-

vacích jazyků. Rovněž nemusí být snadné zcela libovolná data přenášet jakýmkoli komunikačním kanálem (rychlost/objem dat). Různé HW systémy používají různé kódování/přesnosti čísel.

Formát dat skutečných vstupů a výstupů koncových zařízení není možné, ovlivnit, je dán použitým HW (senzory, aktuátory). Tato různorodost dat však není vhodná pro přenos různými komunikačními kanály. Z tohoto důvodu bylo přistoupeno k výběru poněkud omezeného množství základních datových typů vhodných pro většinu i malých procesorů. V tabulce 5.1 jsou vybrány zřejmě nejčastěji používané typy hodnot, včetně velikosti, rozsahu a příkladu použití. Tedy takové datové typy *RValues* a *WValues*, na které lze převést naprostou většinu reálných veličin ze senzorů/do aktuátorů a tím znatelně zjednodušit jejich přenos do/z centrální jednotky. Další typy je v případě potřeby samozřejmě možné přidat.

Seznam použitých datových typů byl vytvořen s ohledem na minimální množství přenášených dat při komunikaci (textová/binární) a jejich nejjednodušší zpracování/uložení v jednoduchých procesorech.

N	Název	Implementace		Bitů	Rozsah	Příklad použití
		MCU	centrála			
1	**Pulse**	`bool`	`bool`	1	`true/false`	stisk tlačítka
2	**Bool**	`bool`	`bool`	1	`true/false`	světlo
3	**Uint8**	`uint8_t`	`uint`	8	$\{0,\ldots,2^8-1\}$	stupeň vytápění
4	**Uint16**	`uint16_t`	`uint`	16	$\{0,\ldots,2^{16}-1\}$	CO_2 ppm
5	**Uint32**	`uint32_t`	`uint`	32	$\{0,\ldots,2^{32}-1\}$	pulzy vodoměru
6	**Int8**	`int8_t`	`int`	8	$\{-2^{-7},\ldots,2^7-1\}$	vnější teplota
7	**Int16**	`int16_t`	`int`	16	$\{-2^{-15},\ldots,2^{15}-1\}$	
8	**Int32**	`int32_t`	`int`	32	$\{-2^{-31},\ldots,2^{31}-1\}$	
9	**Float2**	`int16_t /100`	`double`	16	$\langle -327.67,327.68)$	teplota
10	**Float4**	`int16_t /10000`	`double`	16	$\langle -3.2767,3.2768)$	úhel v rad
11	**Float**	`float IEE754`	`double`	32	R	
12	**String**	`uint8_t+char[]`	`string`	8+?	2^8-1 ASCII znaků	chybová hláška
13	**Binary**	`uint8_t+uint8_t[]`	`byte[]`	8+?	2^8-1 bytů	RFID kód

Tabulka 5.1: Datové typy vstupů a výstupů koncových zařízení

Pulse je speciální datový typ použitý pro signalizaci události. Nutnost jeho zařazení se ukázala až v průběhu realizace projektu. Dvě bezprostředně po sobě následující změny datového typu *Bool* (aktivní/neaktivní), odpovídající například rychlému stisknutí a uvolnění tlačítka, nemusí být totiž vždy správně a tedy včas centrálou zpracovány.

Datové typy *Float2* a *Float4* jsou celá čísla, posunutá o 2, respektive 4 desetinná místa doleva, Tedy dělená 100 nebo 10000. Jejich použití je motivováno úsporou paměti v malých procesorech a při pomalé binární komunikaci proti plnohodnotnému typu *Float*.

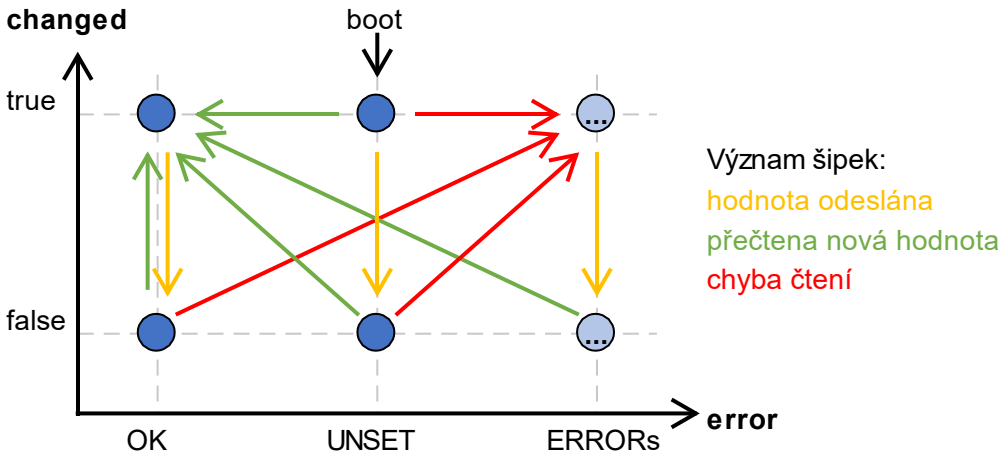
5.2 Stavy vstupů a výstupů

V koncovém zařízení a v centrále (částečně i při přenosu komunikačním kanálem) obsahují *RValues* a *WValues* kromě svého typu a aktuální hodnoty ještě další informace o svém stavu. Tento stav se skládá ze dvou částí:

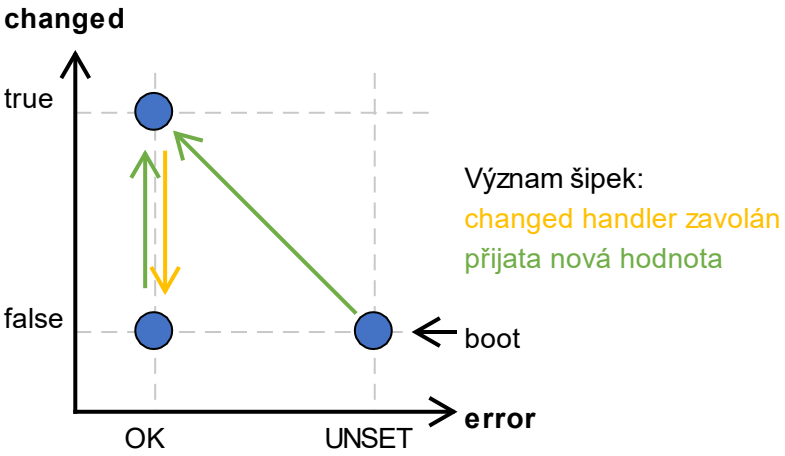
- `bool changed` – Indikace změny hodnoty. Nová hodnota byla koncovým zařízením přečtena ze senzoru, ale dosud neodeslána do centrály. Případně byla hodnota přijata z centrály a dosud nezapsána koncovým zařízením na akční člen.
- `dh_error_t error` – Výčtový typ určující platnost hodnot:
  - `DH_OK` – hodnota je platná
  - `DH_UNSET` – hodnota nebyla dosud přijata/přečtena
  - `DH_ERROR_GENERIC` – v průběhu čtení došlo k neznámé chybě
  - `DH_ERROR_CONNECTION` – v průběhu čtení došlo k chybě spojení
  - `DH_ERROR_TIMEOUT` – v průběhu čtení vypršel čas

  Dle potřeby lze přidávat další specifické chyby. Chybové stavy *DH_ERROR_X* mají význam pouze v případě *RValues*. *WValue* chybnou přijatou hodnotu ignoruje.

Význam stavových proměnných je nejlépe zřejmý z následujících stavových diagramů:



Obrázek 5.1: Přechody mezi stavy u *RValues*



Obrázek 5.2: Přechody mezi stavy u *WValues*

5.3 Požadavky na protokol

Přenos dat se může zdát velmi snadnou záležitostí. Ve skutečnosti se však často jedná o jednu z nejsložitějších částí projektu. V praxi se setkáváme s různými protokoly, běžícími na různých přenosových kanálech. Je to dáno jednak historickým vývojem a jednak optimálním využitím specifických vlastností jednotlivých kanálů.

V práci však bude uvažován univerzální protokol, schopný provozu na různých komunikačních kanálech. Rozlišené budou pouze následující dva případy:

- Textový protokol pro relativně rychlé kanály/přenosy (WiFi, LAN – desítky KB/s až jednotky MB/s), případně i pro pomalejší (sériový port, Bluetooth – jednotky až desítky KB/s), pokud se neočekává vysoká frekvence přenosu dat. Značnou výhodou je čitelnost, a tedy snadné ladění.
- Binární protokol pro velmi pomalé kanály/přenosy (CAN s 8 byty na paket), případně pro aplikace s vysokou frekvencí přenosu dat. V tom případě je potřeba data přenášet maximálně úsporně, bez zbytečných režijních informací.

Ať již bude protokol realizován textově či binárně, je nutné, aby obsluhující knihovna implementovala následující funkce:

- *Ping* – Odpověď na dotaz, zda cílové zařízení existuje.
- *Details* – Vypsání názvu zařízení a informací o obsažených *RValues* a *WValues*
- *Read* – Čtení hodnot *RValues* (z koncového zařízení).
- *Write* – Zapsání hodnot *WValues* (do koncového zařízení).

V případě kanálů, umožňujících inicializaci komunikace všemi uzly (účastníky komunikace) pak navíc:

- *ChangedInfo* – Autonomní notifikace při změně hodnoty vstupu nebo při vypršení časového intervalu, aniž by bylo nutné aktivně číst každý koncový uzel z centrální jednotky.

Pro jednoduchost komunikačního protokolu uvažujme pouze čtení všech *RValues* a zápis všech *WValues* najednou. Toto zjednodušení si můžeme dovolit, neboť předpokládáme použití spíše „malých“ zařízení s několika vstupy a výstupy. Samozřejmě pokud nebudou určitá data/hodnoty v přenosu obsažena, tak na ně nebude cílové zařízení (koncové zařízení nebo centrála) reagovat. Takto lze v podstatě uskutečnit i přenos pouze vybraných hodnot, tedy nikoli vždy všech najednou.

5.4 Textový protokol

Vzhledem ke zvyšující se přenosové rychlosti současných komunikačních kanálů (i MB/s) lze stále více využívat textový přenos dat. Tyto typy přenosů jsou velmi dobře čitelné a lze je tedy velmi snadno ladit v případě potíží/chyb. Velmi zřídka se však přenášejí surová textová dat, většinou jsou vložená do nějakého vhodného formátu/rámce.

V tomto projektu je využit známý formát s názvem JSON. Mezi jeho výhody patří jednoduchost, čitelnost, úspornost a implementace jak v .NET Core tak i existence nenáročných knihoven pro různé mikrokontrolery. Zvolena byla jeho úsporná varianta, kde *RValues* a *WValues* jsou uloženy jako pole obsahující stav platnosti a přenášenou hodnotu. Následující tabulka obsahuje příklady jednotlivých zpráv komunikace s hypotetickým zařízením „SenzorKuchyne“ obsahujícího kromě vstupů měření teploty a relativní vlhkosti také výstup světelnou signalizaci.

Příkaz centrála → zařízení	Odpověď zařízení → centrála
`Ping`	`PingResponse`

`Details`	`DetailsResponse {`
	`  "Name": "SenzorKuchyne",`
	`  "RValues": {`
	`  "Teplota": "Float2",`
	`  "Vlhkost": "Float2",`
	`  },`
	`  "WValues": {`
	`  "Svetlo": Bool"`
	`  }`
	`}`

`Read`	`ReadResponse {`
	`  "Teplota": [ "Unset", 0.00 ],`
	`  "Vlhkost": [ "OK", 38.65 ]`
	`}`

`Write {`
`  "Svetlo": true`
`}`
	`WriteResponse`

-	`ChangedInfo {`
	`  "Teplota": [ "OK", 21.50 ],`
	`  "Vlhkost": [ "OK", 38.65 ]`
	`}`

Tabulka 5.2: Příklady zpráv v textovém protokolu

Datový typ „Binary“ je přenášen jako hexadecimální textový řetězec (AsciHex číslice) bez prefixu 0x.

V případě kanálu neudržujícího spojení, tedy umožňujícího zasílání paketů (například UDP), odpovídají pakety jednotlivým přenášeným zprávám. Pokud kanál pracuje s permanentně otevřeným / udržovaným spojením (UART, TCP, Bluetooth), je použit jako oddělovač zpráv znak nového řádku \n. Kódování předpokládáme ASCII (případně UTF-8), bez podpory speciálních znaků.

5.5 Binární protokol

Pro případ budoucího použití kanálu s velmi omezenou propustností (CAN s 8B na zprávu, BlueTooth LowEnergy s 20B na zprávu, …) byla navrhnuta rovněž binární alternativa komunikačního protokolu. Dvojice zpráv *Details*, *DetailsResponse* je určena k jednorázovému přenosu pouze v průběhu konfigurace zařízení/systému, proto v tomto případě není délka nikterak kritická. Můžeme tedy zachovat textový popis *RValues* a *WValues*.

Obecný popis jednotlivých bajtů binárního protokolu je uveden v tabulce 5.3. V tabulce 5.4 jsou ekvivalenty zpráv z tabulky 5.2. Detaily jsou následující:

- Pořadí bytů v hodnotách je little endian. Nejprve nižší, pak vyšší byte (naprostá většina procesorových systémů).
- První bajt je kód paketu. 0x01 – 0x7F odpovídá směru z centrály do zařízení, 0x81 – 0xFF naopak ze zařízení do centrály.
- Textové popisy v *DetailsResponse* jsou ukončené znakem \0.
- Typ hodnoty odpovídá sloupci *N* v tabulce 5.1.
- Pokud je hodnota v chybovém stavu (ne *DH_OK*), pak bajt *typ/chyba* paketů *ReadResopnse* a *ChangedInfo* má nejvyšší bit 1 a spodních 7 bitů odpovídá kódu chyby.
- Hodnota textového datového typu je ukončena znakem \0. U Binárního datového typu je první bajt hodnoty vždy délka.

Paket	B1	B2	B3		B4...					
Ping	0x01	X								
Ping Response	0x81	X								
Details	0x02	X								
Details Response	0x82	počet R	počet W	název zařízení	název R1	typ R1	…	název W1	typ W2	…
Read	0x03	X								
Read Response	0x83	počet R	typ/chyba 1	hodnota 1	typ/chyba 2	hodnota 2	…			
Write	0x04	počet W	typ 1	hodnota 1	typ 2	hodnota 2	…			
Write Response	0x84	X								
Changed Info	0x85	počet R	typ/chyba 1	hodnota 1	typ/chyba 2	hodnota 2	…			

Tabulka 5.3: Obecný popis binárního protokolu

Paket	B1	B2	B3	B4	B5	B6	B7...B17	B18...B25	B26	B27...B34	B35	B36...B42	B43
Ping	0x01	X											
Ping Response	0x81	X											
Details	0x02	X											
Details Response	0x82	2	1	"SenzorKuchyne"				"Teplota"	0x09	"Vlhkost"	0x09	"Svetlo"	0x02
Read	0x03	X											
Read Response	0x83	2	0x81	0x09	15	25	X						
Write	0x04	1	0x02	1	X								
Write Response	0x84	X											
Changed Info	0x85	2	0x81	0x09	15	25	X						

Tabulka 5.4: Příklady zpráv v binárním protokolu

Binární pakety neobsahují žádné unikátní značky začátku a konce. Jejich oddělení je dosaženo odmlkou v průběhu komunikace / přenosu. Avšak pro zajištění správné integrity (správnosti obsahu) jsou doplněny kontrolním součtem typu CRC16.

5.6 Specifika použitých kanálů

Na přenosové kanály vybrané k implementaci budeme pohlížet jako na proud dat, kde lze použít textový protokol z části 5.4 (mají dostatečnou propustnost dat). Je však třeba definovat způsob adresace (rozlišení) jednotlivých zařízení a některé další skutečnosti závislé na konkrétním přenosovém kanále.

5.6.1 Sériová linka

Použití sériové linky (UART, dnes nejčastěji s USB převodníkem do PC) je velice přímočaré. UART je základní vestavěné rozhraní ve všech použitých mikrokontrolerech. Jedná se o obousměrný proud dat s nastavitelnou přenosovou rychlostí. Samotný UART používá dva vodiče (pro každý směr přenosu dat zvlášť) s napěťovými úrovněmi definovanými použitou technologií (TTL, HCMOS), případně je možné použít budiče napěťových úrovní pro větší dosah, tedy standardy RS232 nebo RS485. Ačkoliv některé komunikační kanály jako například RS458 umožňují na jednu sběrnici připojit až 32 zařízení, v práci bude předpokládáno spojení vždy pouze dvou uzlů (jedna sériová linka z centrální jednotky do jednoho koncového zařízení) s adresací názvem sériového portu v OS (COM1, /dev/ttyUSB0). V základním nastavení předpokládáme přenosovou rychlost $115200bd \cdot s^{-1}$, s jedním stop bitem a bez parity. To odpovídá reálné přenosové rychlosti zhruba 10KB/s

5.6.2 Ethernet, WiFi (UDP, TCP)

Přenosové kanály Ethernet (LAN) a WIFI se řadí mezi vysokorychlostní přenosové kanály (i několik MB/s). Pomocí nich se data přenášejí využitím následujících protokolů.

UDP (User Datagram Protocol) je protokol Transportní vrstvy OSI modelu. Přenos zpráv probíhá pomocí jasně ohraničených paketů, spojení mezi uzly není vytvářeno/udržováno. Zpráva je odeslána a předpokládá se její doručení (v současné době v podstatě jisté). To dává možnost zasílání zpráv, jejichž příjemci mohou být všechny uzly v dané síti (Broadcast). Tento způsob komunikace je v naší aplikaci vhodný například pro vyhledávání nových koncových zařízení. V aplikaci je použit vždy port 8000.

Protokol UDP neposkytuje záruku doručení paketu. Spolehlivost nižších vrstev zásobníku OSI bývá však v případě malých pevných domácích instalací natolik dobrá, že lze tento typ protokolu v řadě případů s výhodou použít i bez dodatečného ověřování. Je jednoduchý a nenáročný, tudíž vhodný pro použití v nekritických zařízeních, kde ztráta paketu nepovede k závažnému selhání systému. Například nepřenesení stisku tlačítka a nerozsvícení světla s pravděpodobností 0,01% je zcela nepodstatné.

TCP (Transmission Control Protokol) je spojový protokol (vytvářející a udržující spojení) transportní vrstvy OSI modelu, garantující doručení zpráv. Jako takový je vhodný i pro použití v kombinaci s méně spolehlivými nižšími vrstvami (WiFi). Na TCP staví mnoho protokolů vyšších vrstev (http, MQTT, …), nám však postačí prostý datový kanál TCP. V aplikaci je použit vždy port 8001.

Implementace je však poněkud složitější než v případě UDP a pro obousměrnou komunikaci je nutné udržovat kanál stále otevřený (což však zajišťují patřičné programové knihovny). V režimu dotaz-odpověď je možné vždy otevřít spojení, provést komunikaci a spojení poté uzavřít. Připojení probíhá způsobem klient-server. Koncové zařízení může být jak klientem, tak serverem, obě implementace mají své výhody. V prvním případě centrála okamžitě vidí, jaká koncová zařízení jsou k dispozici (koncová zařízení se automaticky připojí na centrálu), ve druhém případě koncové zařízení nemusí znát adresu centrály (centrála se připojuje na koncové zařízení). Implementována bude první varianta.

V případě platformy Arduino je rozhraní protokolů TCP a UDP nezávislé na použitém HW (ESP8266, ESP32 nebo libovolný mikrokontroler s Ethernet čipy ENC28J60 nebo WIZ5500, připojenými přes sběrnici SPI). Přístup ke kanálu se děje pomocí bázových tříd *UDP*, *Client* a *Server*. V případě jednoduchých procesorů STM32 bude použit externí SPI čip ENC28J60 (složitější STM32 již řadič sběrnice obsahují).

Jednotlivá zařízení budou vždy adresována pomocí IP adresy.

5.6.3 Bluetooth (Classic)

Bluetooth Classic je standard pro bezdrátovou komunikaci ve stejném rádiovém pásmu jako WiFi (2,4GHz). Specifikace však zahrnuje i vyšší vrstvy – tzv. profily. Jedná se o dohodnuté způsoby využití kanálu (např. přenos souborů, přenos zvuku, …) Pro zakomponování do naší aplikace nejlépe vyhoví virtuální sériový port označovaný Serial Port Profile (SPP). Jedná se o emulaci sériového portu a implementují ho například velmi dostupné moduly s názvy HC-05/HC-06. Tyto moduly jednoduše převádějí lokální UART rozhraní (z procesoru) na zmíněný SPP (přes Bluetooth). Proto se implementace na straně koncového zařízení nebude nijak lišit od běžné sériové linky (UART). Adresace zařízení bude probíhat pomocí 48-bitové tzv. Bluetooth adresy (ekvivalent MAC adresy v sítích LAN).

Maximální reálná přenosová rychlost pro Bluetooth Classic je zhruba 10KB/s až 50KB/s (podle použitého modulu a zahlcení prostředí signálem).

5.6.4 Bluetooth Low Energy

Bluetooth Low energy (BLE) využívá stejné frekvenční pásmo jako Bluetooth Classic, ovšem technologie jsou obecně nekompatibilní a pro podporu obou současně musí být použito tzv. duální zařízení. Obě části však mohou sdílet jednu anténu a tedy pracovat současně.

Jedná se o moderní technologii optimalizovanou pro malý odběr energie, a tedy vhodnou pro bateriové aplikace. Po vzoru Bluetooth Classic obsahuje specifikace několik profilů podle způsobu využití konkrétního zařízení. Celková architektura je však složitější. Základní schéma je na obrázku 5.3, podrobně viz [20].

Veškerá komunikace je standardně založena na profilu GATT. Ten obsahuje služby (services) skládající se z charakteristik. Charakteristiky nesou data. Lze je číst a zapisovat z připojeného klienta (příznaky READ, WRITE, NOTIFY, …). Kromě toho mohou obsahovat tzv. deskriptory, poskytující dodatečné informace/popis. Bylo by tedy možné vytvořit ideální variantu komunikace s koncovým zařízením, kdy by *RValues* byly obsaženy jako charakteristiky pod jednou služnou (service) a *WValues* pod druhou. Názvy, datové typy a stavy by pak byly uloženy v deskriptorech příslušných charakteristik.

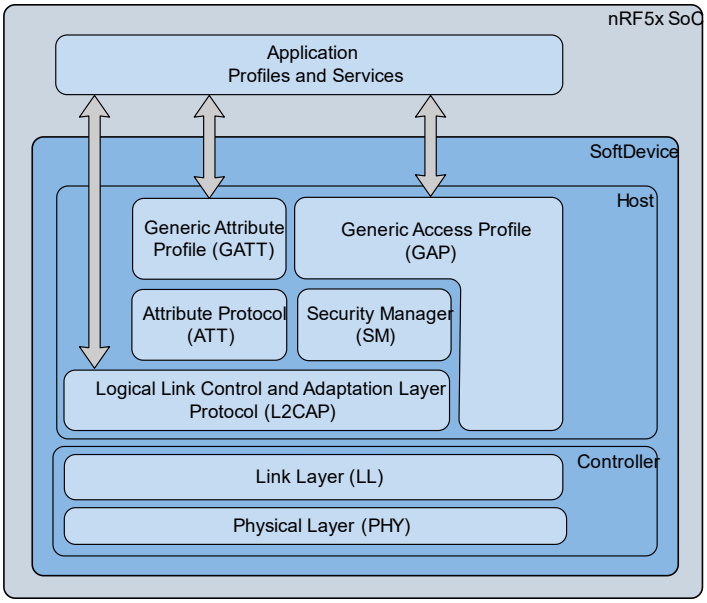
Bohužel však běžně dostupné a levné HW moduly poskytují pouze omezenou možnost konfigurace deskriptorů, případně mají striktně omezený maximální počet vytvářených charakteristik. Vzhledem k identifikaci charakteristik pomocí UUID se navíc charakteristiky spíše hodí pro sériově vyráběná zařízení, splňující určitý standard specifikace (měřič pulzu, tachometr, …). Pro účely práce bude proto opět použit přístup emulovaného sériového kanálu. Pro BLE sice standardní obdoba profilu SPP neexistuje, dá se však snadno vytvořit použitím jediné charakteristiky s příznaky WRITE a NOTIFY

sloužící jako buffer pro 20 bytů sériového kanálu (standardní limit velikosti dat pro přenos pomocí jedné charakteristiky).

Komunikační kanál Bluetoth LowEnergy je určen primárně k zasílání malých bloků o velikosti 20 bytes (některá rozšíření až 200 bytes) avšak pouze několikrát za vteřinu (10x až 20x). Reálná přenosová rychlost je tedy velmi nízká, a to kolem 1KB/s až maximálně 5KB/s (rozšířené varianty). Pokud je přenos ještě navíc potvrzovaný, tak reálná rychlost dále velmi klesá.

K adresaci bude opět použita 48-bitová Bluetooth adresa, ovšem je třeba použít adresu veřejnou (Public), nikoliv náhodnou (Random). Náhodná adresa se může měnit při spuštění čipu, proto pro adresaci není vhodná. V případě použití náhodné MAC adresy (není až tak výjimečné) je nutno zařízení identifikovat pomocí obsahu tzv. advertisement paketu, jenž bohužel ne vždy lze v levných modulech nastavit zcela podle potřeby.

Moduly, vhodné pro experimentální použití jsou například AT-09 nebo ST Microelectronic BlueNRG.



Obrázek 5.3: Zásobník protokolů BLE [21]

5.7 Knihovna pro koncové zařízení

Jak již bylo řečeno, vzhledem ke koncovému zařízení se tato práce zabývá pouze definováním datových typů a jejich přenosem do centrály. Za tímto účelem byla vytvořena programová knihovna v jazyce C jako podpora pro snadný vývoj koncového zařízení.

Vlastní knihovna pro MCU, implementující datové typy a navržený protokol je zcela nezávislá na konkrétním komunikačním kanálu. Je obsažena v souborech *dothome.h* a *dothome.c*. Rozhraní je definováno následujícími datovými typy:

```
// Datové typy R- a WValues
typedef enum {
    DH_PULSE,
    DH_BOOL,
    DH_UINT8,
    DH_UINT16,
    DH_UINT32,
    DH_INT8,
    DH_INT16,
    DH_INT32,
    DH_FLOAT2,
    DH_FLOAT4,
    DH_FLOAT,
    DH_STRING,
    DH_BINARY
} dh_type_t;

// Chybové stavy R- a WValues
typedef enum {
    DH_OK,
    DH_UNSET,
    DH_ERROR_GENERIC,
    DH_ERROR_CONNECTION,
    DH_ERROR_TIMEOUT
} dh_error_t;
```

```
// Uložená hodnota R- a WValues
typedef union {
    bool pulse_val;
    bool bool_val;
    uint8_t uint8_val;
    uint16_t uint16_val;
    uint32_t uint32_val;
    int8_t int8_val;
    int16_t int16_val;
    int32_t int32_val;
    int16_t float2_val;
    int16_t float4_val;
    float float_val;
    struct {
        uint8_t length;
        uint8_t max_length;
        char *value;
    } string_val;
    struct {
        uint8_t length;
        uint8_t max_length;
        uint8_t *value;
    } object_val;
} dh_union_t;
```

```
// RValue (vstup zařízení)
typedef struct dh_r_value {
    char* name;
    dh_type_t type;
    dh_error_t error;
    bool changed;
    dh_union_t value;
} dh_r_value_t;

// WValue (výstup zařízení)
typedef struct dh_w_value {
    char* name;
    dh_type_t type;
    dh_error_t error;
    bool changed;
    dh_union_t value;
    void (*changed_handler)
        (struct dh_w_value*);
    // Funkce, volaná knihovnou
        po zápisu nové hodnoty
} dh_w_value_t;
```

```
// Reprezentace celého
     koncového zařízení
typedef struct {
    char *name;
    uint16_t r_values_count;
    dh_r_value *r_values;
    uint16_t w_values_count;
    dh_w_value *w_values;
} dh_handle_t;
```

Dále jsou definovány funkce pro textovou komunikaci:

- `void dh_handle_text(dh_handle_t *dh_handle, char *text, void(*printer)(char*, bool))`
  - Musí být zavolána, byla-li komunikačním kanálem přijata textová zpráva (do koncového zařízení). Knihovna požadavek zpracuje a odpoví funkcí *printer*. U případných změněných *WValues* zavolá funkci *changed_handler*.
- `void dh_poll_text(dh_handle_t *dh_handle, void (*printer)(char*,bool))`
  - Musí být volána cyklicky v hlavní smyčce programu. Knihovna v případě změny některých *RValues* vyšle textovou *ChangedInfo* zprávu funkcí *printer*.

Analogicky existují i funkce pro binární komunikaci

- `void dh_handle_bin(…)`
- `void dh_poll_bin(…)`

Funkcemi pro editaci *RValues* jsou následující:

- `void dh_set_error(dh_r_value_t *value_handle, dh_error_t error)`
  - Nastavuje hodnotě *value_handle* chybový stav *error*.
- `void dh_trigger_pulse(dh_r_value_t *value_handle)`
  - Zapíše informaci o pulzu do *value_handle*. Na straně centrály bude po odeslání vygenerován pulz.
- `void dh_set_bool(dh_r_value_t *value_handle, bool value)`
  - Nastaví hodnotu *value_handle* typu *Bool*.
- `void dh_set_uint(dh_r_value_t *value_handle, uint32_t value)`
  - Nastaví hodnotu *value_handle* typu *Uint8, Uint16* nebo *Uint32*.
- `void dh_set_int(dh_r_value_t *value_handle, int32_t value)`
  - Nastaví hodnotu *value_handle* typu *Int8, Int16* nebo *Int32*.
- `void dh_set_float(dh_r_value_t *value_handle, float value)`
  - Nastaví hodnotu *value_handle* typu *Float2, Float4* nebo *Float*.
- `void dh_set_binary(dh_r_value_t* value_handle, uint8_t* val,uint8_t len)`
  - Nastaví hodnotu *value_handle* typu *Binary* na hodnotu *val* s délkou *len*.

Tyto funkce kromě samotné změny hodnoty ze senzoru rovněž zkontrolují, zda došlo k její změně od předešlé uložené hodnoty a případně nastaví příznak *changed*, aby při následujícím periodickém volání funkce *dh_poll_X* byla vyslána zpráva *ChangedInfo* do centrální jednotky. Zmíněné funkce tedy data do centrály přímo nevysílají, pouze je zapisují/ukládají v koncovém zařízení a teprve periodické volání funkce *dh_poll_X* vysílá data označená příznakem změny.

Funkce *printer* je vlastně vložená závislost, umožňující knihovnu zcela odstínit od implementace komunikačního kanálu. Skutečná funkce *printer* je tedy vytvořena podle aktuálně použitého komunikačního kanálu a je na ni předán odkaz do knihovny.

Následuje minimální příklad použití knihovny pro referenční zařízení „SenzorKuchyne“ z 5.4 na platformě Arduino, komunikující po výchozím sériovém portu.

```
#include <dothome.h>

// vytvoření přijímacího bufferu
#define BUFF_SIZE 100
char buffer[BUFF_SIZE];
void handler(dh_w_value* value);

float teplota, vlhkost;

dh_r_value r_values[2] = { // definice RValues
     {.name = "Teplota", .type = DH_FLOAT2, .error = DH_UNSET, },
     {.name = "Vlhkost", .type = DH_FLOAT2, .error = DH_UNSET, }
};

dh_w_value w_values[1] = { // definice WValues
     {.name = "Svetlo", .type = DH_BOOL,
     .error = DH_UNSET, .changed_handler = handler }
};

dh_handle_t dh = { // definice zařízení
     .name = "SenzorKuchyne",
     .r_values_count = 2,
     .r_values = r_values,
     .w_values_count = 1,
     .w_values = w_values
};

void handler(dh_w_value* value) {
     // Zpracování změny hodnoty WValue
}

// odesílací funkce
void printer(char* text, bool end) {
     Serial.print(text);
     if (end) Serial.print("\n"); // Konec zprávy odpovídá znaku '\n'
}
// inicializace/nastavení
void setup() {
     Serial.begin(115200);
}
// ...
```

```
// ...
// stále se opakující smyčka
void loop() {
     // pokud jsou data od přijmu
     if (Serial.available()) {
          // přečtení prijatých dat / zprávy / paketu
          int r = Serial.readBytesUntil('\n', buffer, BUFF_SIZE);
          if (r > 0 && r < BUFF_SIZE) {
               buffer[received] = 0;
               // textová zpráva byla přijata
               dh_handle_text(&dh, buffer, printer);
          }
     }
     // test na změnu hodnot a jejich vyslání
     dh_poll_text(&dh, printer); // Nutno volat periodicky

     // ... čtení teploty, vlhkosti v určitém intervalu
     // aktualizace hodnot
     dh_set_float(dh_w_values + 0, teplota);
     dh_set_float(dh_w_values + 1, vlhkost);
}
```

Kapitola 6

Centrální jednotka

V předchozí kapitole byl popsán princip a realizace koncového zařízení. Rovněž byly vysvětleny použité komunikační kanály a protokoly a jejich využití při přenosu dat/hodnot. Zbývá v podstatě nejdůležitější a také nejsložitější část a tou je centrální jednotka. I když se jedná pouze o jedno zařízení (jeden počítač), skládá se z několika částí.

Nejprve stručně popišme, jaké jsou činnosti (nikoli části) centrální jednotky:

- Vytvoření a spravování komunikace pro dříve vybrané komunikační kanály.
- Periodické čtení *RValues* z dostupných koncových zařízení, zápis výstupních *WValues* do koncových zařízení.
- Periodické vykonávání pravidel vytvářejících požadovanou činnost celého systému. Ty jsou uloženy v konfiguraci.
- Vizualizaci pro uživatele. Jde o tvorbu WWW stránek pro zobrazení v běžném prohlížeči s případným využitím výhod PWA. Stránky prezentují aktuální stav systému a poskytují uživateli možnost vstupu za účelem ovládání/řízení činnosti systému.
- Konfigurace. Aby se definovalo požadované chování centrální jednotky, je potřeba vytvořit její konfiguraci. Aplikace pro konfiguraci je sice samostatnou aplikací, ale centrální jednotka poskytuje rozhraní pro načtení aktuální konfigurace do této aplikace a rovněž příjem konfigurace vytvořené touto aplikací. Konfigurační aplikace může i nemusí být spouštěna přímo na centrální jednotce (podle její výkonosti).

Jak je z popisu zřejmé, centrální jednotka obsahuje skutečně mnoho funkcionalit. Aby byly všechny vhodně zajištěny a současně byla podporována určitá multi-platformnost, bylo využito softwarové prostředí .NET 5.0/C#. To je schopné zajistit jak lokální, tak i WWW aplikace na několika platformách, a přitom poskytnout velkou míru znovu použitelnosti programového kódu.

Nyní k programovým částem centrální jednotky. Řešení *DotHome* se skládá ze dvou spustitelných projektů:

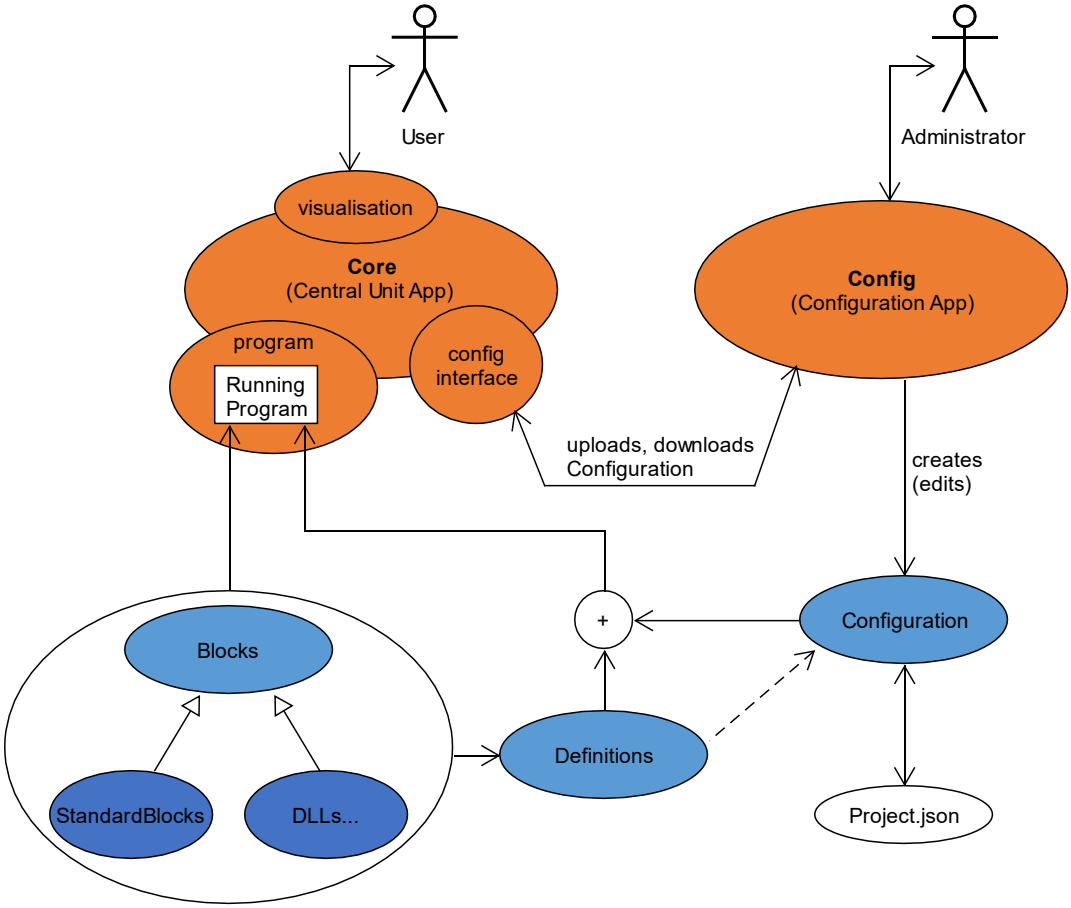
- *DotHome.Core* – Hlavní výkonný program centrální jednotky vytvořený pomocí frameworku ASP.NET Core. Obsahuje tři základní funkce:
  - Program vykonávající řídící algoritmus sestavený z bloků podle zadané konfigurace.
  - Webová aplikace poskytující automaticky generovanou vizualizaci požadovaných funkčních bloků.
  - Rozhraní pro komunikaci s konfigurační aplikací (získání/vložení konfigurace systému).
- *DotHome.Config* – Multiplatformní desktopová konfigurační aplikace vytvořená pomocí grafické knihovny AvaloniaUI. Umožňuje tvořit/upravovat konfiguraci pomocí grafických bloků.

Tyto dvě zmíněné aplikace mají několik společných činností. Například zpracování konfigurace je potřebné nejen v aplikaci pro tvorbu konfigurace, ale rovněž v hlavní aplikaci centrální jednotky pro vykonávání její zadané činnosti.

Dále jsou vytvořené tři knihovní (modelové) projekty:

- *Blocks* (v programovém kódu *DotHome.RunningModel*) – Část pro tvorbu funkčních bloků. V základu neobsahuje žádné hotové/reálné bloky, pouze bázovou abstraktní třídu *Block* a některé další nástroje pro jejich tvorbu. Jednotlivé skutečné bloky jsou tvořené v rámci samostatných projektů, importovaných jako knihovní/DLL soubory do *Config* a *Core*. Základní vestavěné/předdefinované bloky jsou implementovány v *DotHome.StandardBlocks*.
- *Definitions* (v programovém kódu *DotHome.Definitions*) – Pomocná část obsahující metadata o blocích z knihoven dostupných v *Blocks* (informace o vstupech, výstupech, parametrech). Vlastní *Definitions* vznikne načtením dostupných *Blocks* pomocí zabudovaného nástroje *DefinitionsCreator*.
- *Configuration* (v programovém kódu *DotHome.ProgrammingModel*) – Obsahuje informaci o propojení jednotlivých vstupů a výstupů u bloků, hodnoty parametrů a umístění na pracovní ploše. Obsah je editován v samostatné aplikaci *Config* a je ukládán jako projekt do souboru. Po nahrání do *Core* v centrále je převeden na *RunningProgram* a spuštěn.

Výhodou tohoto rozdělení na jednotlivé části je snadnější použití tzv. Data Bindingů v *Config* a celkové odstínění *Config* od implementace bloků. To mimo jiné ponechává do budoucna možnost například celou část *Core* implementovat v jiném jazyce (C++, Python, …).



Obrázek 6.1: Vztahy jednotlivých částí celku *DotHome*

V této kapitole bude dále stručně vysvětlena funkce a implementace jednotlivých výše zmíněných částí. Pro detailní pochopení je vhodné prostudovat dokumentaci v programovém kódu. Než však bude podán popis jednotlivých částí, je vhodné nastínit jejich celkovou činnost. Tu lze z pohledu aplikace *Core* na centrální jednotce stručně popsat následovně:

- Vyhledají se všechny dostupné typy funkčních bloků (*Blocks*). Tedy nejen ty předdefinované v programu, ale i ty dodané pomocí externích knihoven/DLL.
- Ze všech dostupných funkčních bloků se poté vytvoří informace, jaké komponenty (*Blocks*) mohou být obsaženy ve vytvořeném programu pro centrální jednotku. Tyto informace se uloží do části *Definitions*.
- Načte se aktuální konfigurace (*Configuration*). Ta obsahuje informace, jaké bloky jsou jak propojeny a nastaveny. Pomocí informací o dostupných blocích (*Definitions*) jsou vytvořeny instance skutečných bloků a ty jsou naplněny/konfigurovány podle aktuální konfigurace (*Configuration*).

Takto sestavný program nazvaný *RunningProgram* je uložen v centrální jednotce a je periodicky vykonáván. Vykonávání programu spočívá v opakovaném volání metod vytvořených *Blocks*. Těm se předávají hodnoty čtené z koncových zařízení, uvnitř bloků se zpracovávají, postupují do dalších bloků (podle propojení zadaných konfigurací) a nakonec se odesílají zpět do koncových zařízení jako výstupy.

Následuje podrobnější popis některých částí, a to hlavně z programového pohledu.

6.1 Blocks

Část *Blocks* obsahuje kostru pro tvorbu jakýchkoli funkčních bloků. Základní struktura je zřejmá z diagramu tříd na obrázku 6.2.

6.1.1 Blok

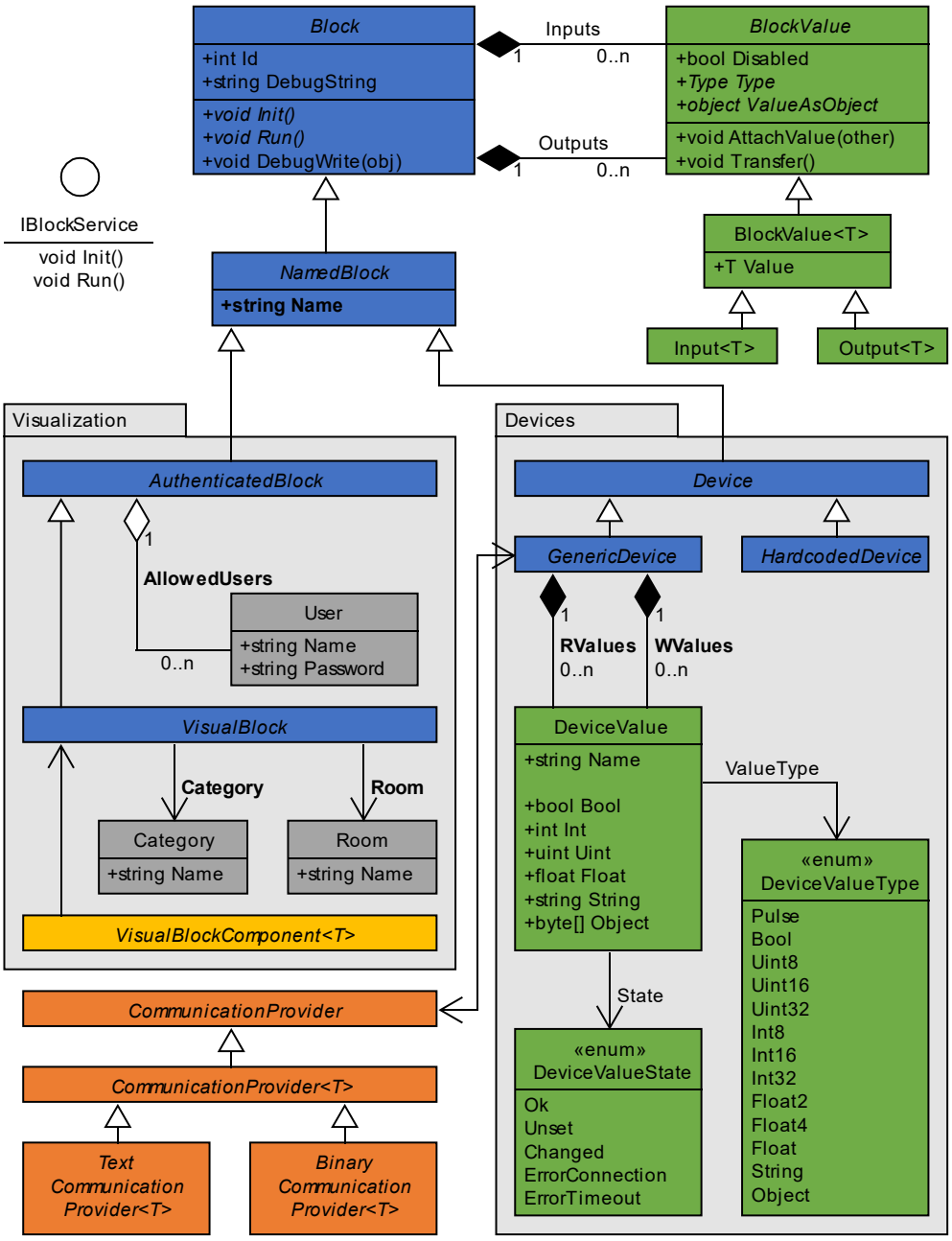
Organizační jednotkou vykonávaného algoritmu je funkční blok. Blokem je instance jakékoliv třídy, zděděné od abstraktní třídy *Block*. Každá taková třída musí implementovat dvě základní metody:

- `void Init()` – Slouží k inicializaci bloku, spouští se při startu centrální jednotky, nebo při změně algoritmu, tedy nové konfiguraci.
- `void Run()` – Obsahuje výkonný kód bloku, spouští se s danou periodou, zpracovává vstupy bloku a nastavuje hodnoty jeho výstupů.

Vstupy a výstupy bloku jsou veřejné vlastnosti typu *Input<T>*, respektive *Output<T>*, kde T musí být jeden z podporovaných datových typů, uvedených v tabulce 5.1 ve sloupci implementace v centrále. Třídy *Input<T>* a *Output<T>* jsou implementací shodné s obecnějším *BlockValue<T>* a existují čistě pro lepší odlišení vstupu a výstupu. Vstupy a výstupy jsou jediným způsobem předávání dat mezi jednotlivými bloky. Toto předávání se uskutečňuje pomocí metod *Attach* (připojí vstup k výstupu) a *Transfer* (přenese hodnotu na všechny připojené vstupy).

Kromě vstupů a výstupů jsou další charakteristikou bloku jeho parametry. Parametrem je každá veřejná vlastnost označená atributem *ParameterAttribute*. Tyto parametry jsou později zobrazené v *Config*, kde je lze editovat. Podporované jsou primitivní datové typy, textový řetězec a některé další zmíněné dále. V diagramu 6.2 jsou vlastnosti představující parametry vyznačeny tučně.

Pokud blok ke své funkci potřebuje externí závislosti (přístup do databáze, komunikační kanál, …), lze tuto závislost vložit v konstruktoru bloku. Jedná se o implementaci návrhového vzoru Dependency Injection. Závislost může být služba z nadřazeného kontejneru ASP.NET Core ServiceCollection, nebo zcela vlastní implementace. Vlastní služby mohou implementovat rozhraní *IBlockService*, poskytující stejné metody, jako blok – *Init* a *Run*. Tyto metody budou v tom případě spouštěny stejně jako v případě bloků.



Obrázek 6.2: UML diagram tříd pro *Blocks*

Následuje příklad implementace jednoduchého bloku sčítajícího až čtyři vstupní hodnoty:

```
[Category("Math"), Description("Output is sum of all inputs"),
    Color("#64b464")]
public class Sum : Block
{
    [Description("Input 1")]
    public Input<double> I1 { get; set; }

    [Description("Input 2")]
    public Input<double> I2 { get; set; }

    [Description("Input 3"), Disablable(true)]
    public Input<double> I3 { get; set; }

    [Description("Input 4"), Disablable(true)]
    public Input<double> I4 { get; set; }

    [Description("Output")]
    public Output<double> O { get; set; }

    public override void Init() { }

    public override void Run()
    {
        O.Value = I1.Value + I2.Value + I3.Value + I4.Value;
    }
}
```

6.1.2 Reprezentace koncového zařízení

Koncové zařízení bude reprezentováno speciálním druhem bloku a to třídou *Device*. *HardcodedDevice* je bázová třída, připravená pro implementaci zcela libovolného koncového zařízení s vlastním komunikačním kanálem, a to libovolným způsobem. Uživatelsky vytvářeným zařízením popsaným v kapitole 5 odpovídá třída *GenericDevice*.

Implementace *DeviceValue* je shodná s implementací dat v koncovém zařízení, odkud budou hodnoty v podstatě zrcadleny. Vstupy a výstupy v tomto případě nejsou přímo vlastnosti *Intput<T>, Output<T>*, ale jsou vytvářené genericky – *RValues* zařízení odpovídají výstupům (jsou výstupem koncového zařízení), *WValues* odpovídají vstupům (jsou vstupy do koncového zařízení).

Vlastní výměna dat s fyzickým zařízením je již implementována v metodě *Run* třídy *GenericDevice*. Zděděné třídy (*UDPDevice, SerialDevice, …*) implementují pouze parametry pro adresaci zařízení a v konstruktoru vybírají příslušnou implementaci *CommunicationProvider*.

CommunicationProvider<T> je bázová generická abstraktní třída, definující rozhraní pro skutečnou komunikaci s daným typem koncového zařízení. Toto rozhraní zahrnuje následující metody:

- `void RegisterDevice(T device)` – Slouží pro zaregistrování zařízení umožňující později přijímat zprávy.
- `void WriteDevice(T device)` – Zapíše do zařízení hodnoty *WValues*.
- `void ReadDevice(T device)` – Zašle zařízení dotaz na stav jeho *RWalues*.
- `List<T> SearchDevices()` – Vyhledá všechna dostupná zařízení na daném komunikačním kanálu a vrátí jejich seznam. Tato funkce bude typicky využívána při konfiguraci. Pokud podstata kanálu neumožňuje vyhledávání, vrátí NULL.

Podtřída *TextCommunicationProvider<T>* poskytuje implementaci textového protokolu popsaného v části 5.4. Po svých potomcích (pro konkrétní komunikační kanál) vyžaduje jen implementaci funkce *TextSend* pro odeslání a vyvolání události *TextReceived* při příchozí komunikaci. Podobně *BinaryCommunicationProvider<T>* implementuje *CommunicationProvider* pomocí *BytesSend* a *BytesReceived*.

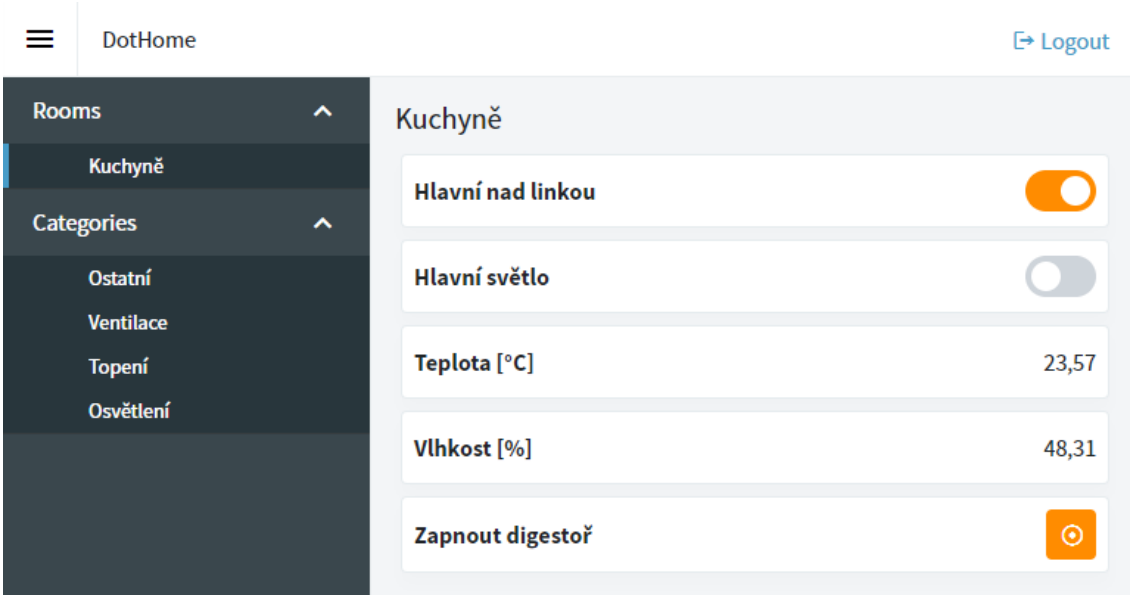
Způsob implementace záleží na konkrétním použitém komunikačním kanálu. Je však třeba si uvědomit skutečnost, kdy vlastní komunikace může nevhodně blokovat vlákno po nenulový čas. Z toho důvodu je ve všech implementacích použita odchozí fronta a průběžné odesílání probíhá ve skutečnosti na pozadí.

6.1.3 Vizualizační bloky

Bloky, odvozené od *AuthenticatedBlock* obsahují navíc jako parametr seznam uživatelů s oprávněním přístupu. Přímým potomkem je například blok *Alert*, realizující notifikaci uživatele ve vizualizaci.

VisualBlock je blok, jehož grafická reprezentace se ve vizualizaci pro uživatele přímo zobrazí. Parametry *Category* a *Room* slouží k filtraci. Vlastní vzhled bloku je definovaný ve *VisualBlockComponent<T>*, což je tzv. Blazor Component, tedy kombinace HTML a C# kódu definující vzhled a chování ovládacího prvku. Příslušný blok typu *T* je pak modelem této komponenty, což umožňuje přímé napojení na jednotlivé vlastnosti. Překreslení komponenty je možné vynutit zavoláním události *VisualStateChanged*.

Obrázek 6.3 uvádí příklad výsledné vizualizace se dvěma vypínači, dvěma bloky zobrazujícími hodnotu a jedním tlačítkem.



Obrázek 6.3: Ukázka vizualizace pro interakci s uživatelem

6.2 Definitions

Definitions (diagram tříd 6.3) je pomocná část vytvořená přečtením dostupných knihoven implementujících nějaký *Block* obsaženým nástrojem *DefinitionsCreator.* Tato část tedy slouží v podstatě jako informační knihovna aktuálně dostupných bloků pro konfiguraci. Na základě informací z *Definitons* a pozdější načtené *Configuration* vzniká *RunningProgram* jenž vykonává centrální jednotka.

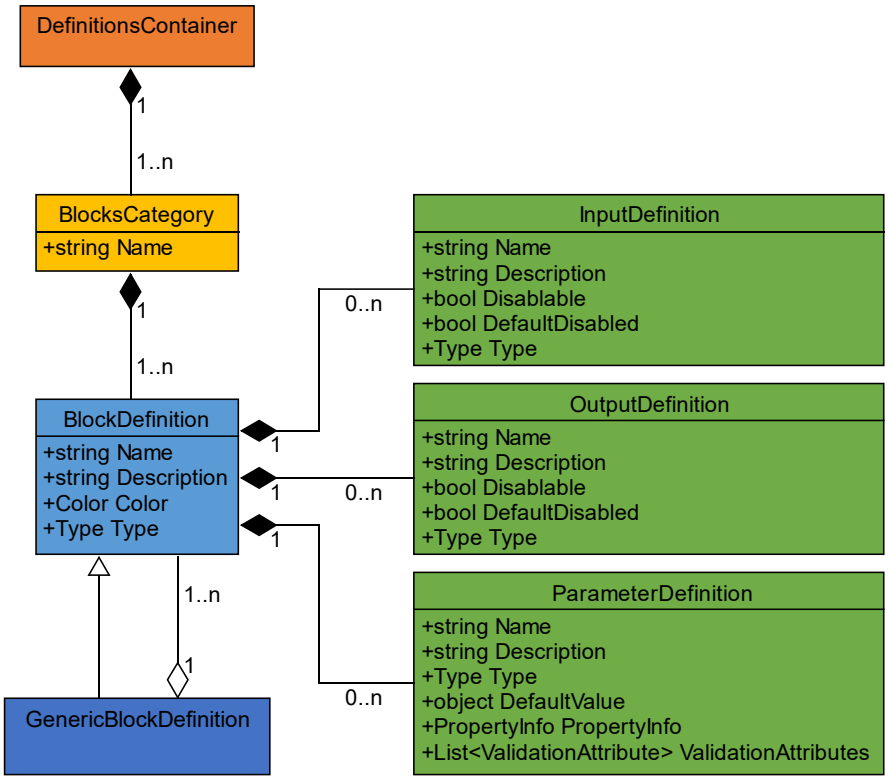
Základní třídou je zde *DefinitionsContainer*, obsahující několik kategorií *BlocksCategory* s definicemi bloků *BlockDefinition*. Bloky jsou do kategorií rozděleny podle atributu *System.ComponentModel.CategoryAttribute* aplikovaného na konkrétní třídu bloku.

Každé neabstraktní třídě, dědící od *Block,* odpovídá jedna instance *BlockDefinition*. V případě generické třídy bloku, je vytvořena instance pro každý ze seznamu podporovaných datových typů v tabulce 5.1 ve sloupci centrála. Tyto možnosti jsou později v konfiguračním GUI zagregovány do *GenericBlockDefinition*, čistě pro větší přehlednost knihovny dostupných bloků.

Vlastnost *Name,* je dána názvem třídy, *Description* atributem *System.ComponentModel.DescriptionAttribute, Color* atributem *DotHome.RunningModel.Attributes.ColorAttribute.* Tyto vlastnosti slouží pro zobrazení v *Config* GUI. *Type* obsahuje přímo typ třídy bloku, jejíž instance bude vytvořena později pro *RunningProgram* podle načteného *Configuration.*

Podobným způsobem odpovídají *InputDefinition*, *OutputDefinition* a *ParameterDefinition* příslušným vlastnostem třídy bloku. Vlastnosti *Disablable* a *DefaultDisabled* souvisejí s možností vypnutí jednotlivých vstupů/výstupů podle konfigurace. *Type* je zde

datový typ hodnoty. *ParameterDefinition* obsahuje rovněž informaci o výchozí hodnotě a validátorech. Validátory (*ValidationAttributes*) jsou standardním nástrojem .NET pro ověření správnosti uživatelem zadané hodnoty. Jedná se například o omezení na interval v případě čísla, nebo o regulární výraz pro textový řetězec. Kromě těchto vestavěných možností byl vytvořen další speciální atribut *UniqueAttribute*, vyjadřující požadavek, aby v celém projektu neexistovaly dva bloky se stejnou hodnotou daného parametru.



Obrázek 6.4: UML diagram tříd pro *Definitions*

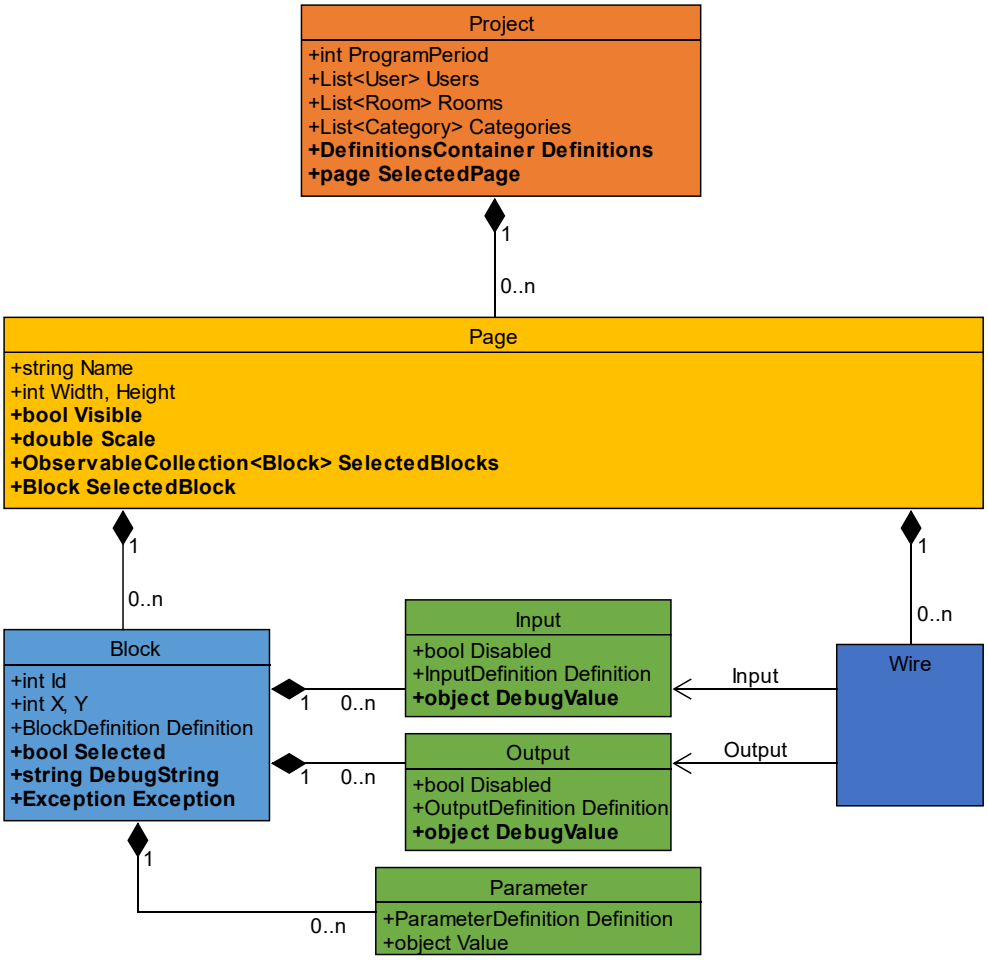
6.3 Configuration

V konfigurační aplikaci je vytvářena nebo upravována část *Configuration* (diagram tříd 6.4). Tučně jsou vyznačené neukládané vlastnosti, tedy ty, co existující pouze pro uchování aktuálního stavu, případně pro ladění.

Vlastnímu konfiguračnímu souboru, nahrávanému později do centrály, odpovídá třída *Project*. Obsahuje seznam uživatelů, místností a kategorií pro účely vizualizace. Celý *Project* je pro přehlednost rozdělen na několik stránek *Page*. Tyto stránky obsahují jednotlivé bloky.

Blok v *Configuration* obsahuje pouze informaci o umístění jednotlivých *BlockDefinitions* na příslušné stránce. Každému *InputDefiniton* odpovídá *Input*, každému *OutputDefinition* odpovídá *Output* a každému *ParameterDefinition* odpovídá *Parameter*. *Wire* jsou spoje určující propojení jednotlivých vstupů a výstupů mezi bloky.

Zde je velmi důležité si uvědomit, že *Configuration* dává smysl jedině v kontextu informací obsažených v *Definitions*. *Definitions* určují, co vše lze použít pro konfiguraci a *Configuration* obsahuje informace, jak je za pomoci dostupných *Definitions* aktuální konfigurace skutečně vytvořena (činnost bloků a jejich propojení).



Obrázek 6.5: UML diagram tříd pro *Configuration*

Součástí *Configuration* je modul *ModelSerializer*, umožňující zápis a čtení vytvořené konfigurace do JSON souboru. Tento formát byl zvolen čistě kvůli zabudované podpoře v .NET, a tedy snadnému použití. Důležitým požadavkem bylo zachování referencí mezi objekty. *Definitions* se do souboru neukládají, neboť jsou jednoznačně určeny aktuálně

dostupnými knihovnami bloků/DLL. Je tedy vždy uložen pouze název příslušného *BlockDefinition* a případně jeho parametry. Pokud dojde ke změně *Definitions* (například došlo k přidání bloků, změně implementace), je třeba při načítání soubor s konfigurací patřičně upravit.

6.4 Core

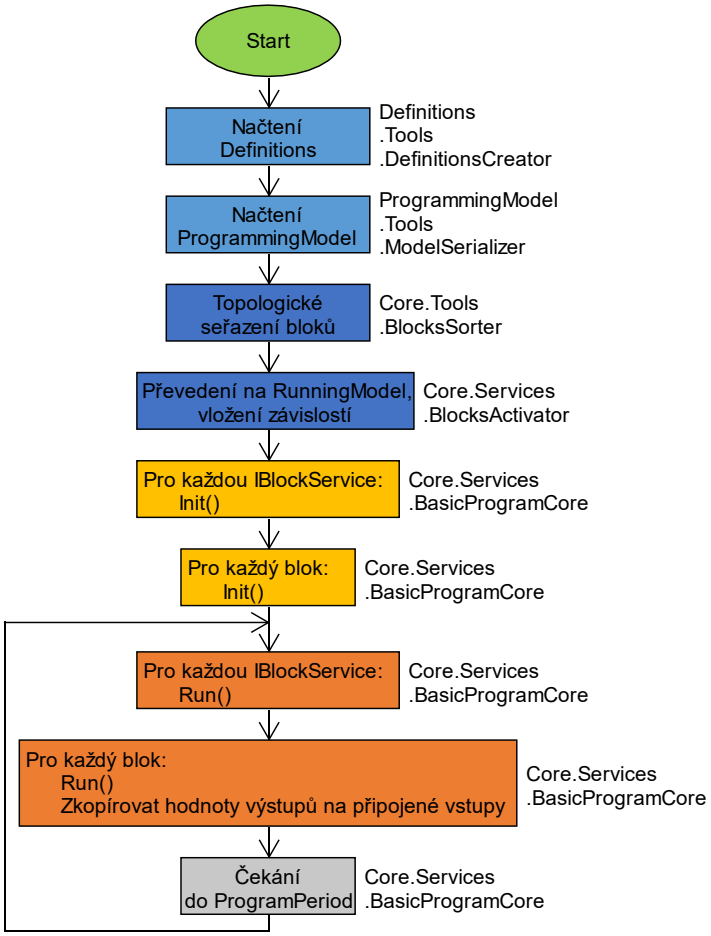
Core je nejdůležitější aplikace celého navrhovaného řešení. Běží v centrální jednotce. Jak již bylo zmíněno, staví na frameworku ASP.NET Core. To umožňuje patřičnou modularitu a škálovatelnost. Podrobný popis všech součástí není předmětem této práce, proto bude jen stručně představen způsob vykonávání řídícího algoritmu (*RunningProgram*) modulem *Services.BasicProgramCore*.

Jak bylo řečeno, řídící algoritmus je založen na činnosti bloků. Tyto bloky je nejprve potřeba nastavit podle informací načtených ze souboru Project.json a vytvořit z nich *RunningProgram*. Závislosti podle vzoru Dependency Injection jsou uchovávány a vkládány modulem *Services.BlocksActivator*.

Před vlastním vytvořením RunningProgram je však potřeba bloky topologicky správně seřadit [22]. Jedná se o řazení, při kterém pokud existuje spojení výstupu bloku A se vstupem bloku B, pak blok A musí být ve výsledném pořadí před blokem B. Tím je zajištěn správný tok dat, tedy směrem z výstupu jednoho bloku do vstupu následujícího bloku. Smyčky mezi bloky jsou pro jednoduchost zakázány. V opačném případě by bylo nutné algoritmus řazení doplnit o jejich detekci a přerušení.

Po seřazení a převedení potřených bloků na *RunningProgram* jsou zavolány metody *Init* jednotlivých bloků (jejich prvotní inicializace po vytvoření). Následně jsou v pravidelných časových intervalech, určených vlastností *Configuration.Project.ProgramPeriod,* volány metody *Run* (krok vykonávaného programu) jednotlivých bloků. Hodnoty jsou poté přenášeny z výstupů bloků na připojené vstupy jiným bloků.

Vzhledem ke zcela dostatečnému výkonu dnešních počítačů není periodické vykonávání žádný problémem. V případě potřeby úspory výpočetního výkonu by bylo třeba sledovat změny hodnot vstupů jednotlivých bloků a metodu *Run* volat skutečně pouze v případě detekované změny některého vstupu. Problém by však mohl nastat u bloků, jež jsou samy zdrojem změn (zařízení, časovače, uživatelské ovládací prvky, …). Celý algoritmus by pak musel být značně komplikovanější. V případě bloků se složitější metodou *Run* je vždy možné sledovat změny interně a výpočetně náročný algoritmus spustit jen v případě nutnosti.



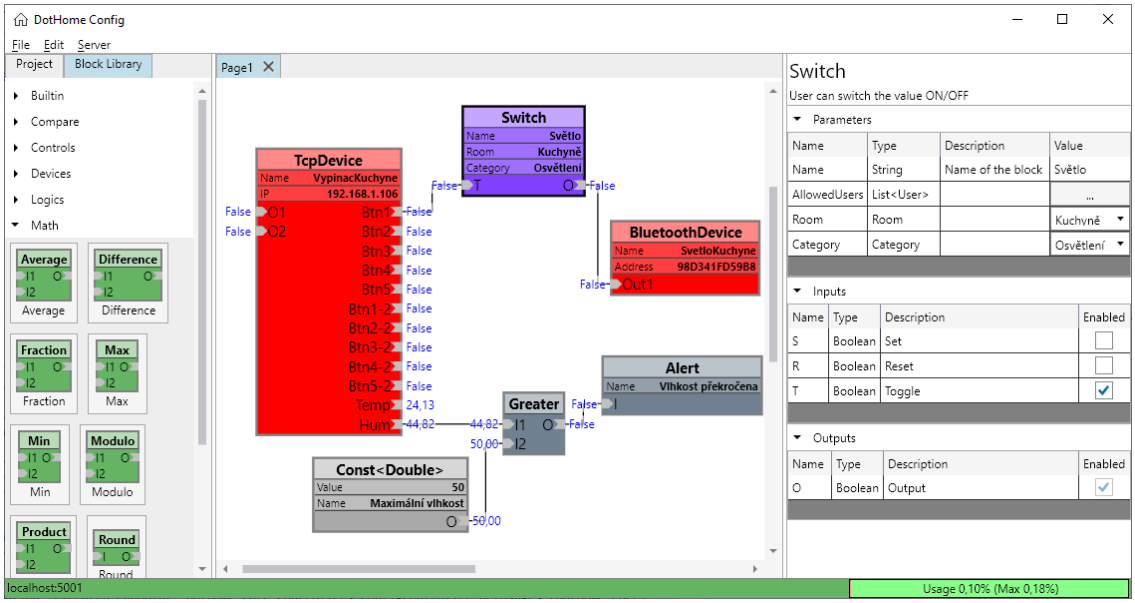
Obrázek 6.6: Vývojový diagram řídícího algoritmu (vpravo příslušný modul)

6.5 Config

Config je samostatná aplikace tvořící administrátorské rozhraní pro konfiguraci systému. Tedy pro snadné spojování jednotlivých bloků do funkčních celků intuitivním grafickým způsobem pomocí metody drag&drop. Jedná se o desktopovou aplikaci, vytvořenou s využitím grafického frameworku AvaloniaUI (obdoba WPF). Popis implementace by byl opět nad rámec této práce. Rozhraní aplikace je vytvořeno technikou Data Binding, přičemž data jsou získávána z výše popsaného *Configuration*. Komunikace s centrální jednotkou se uskutečňuje přes Web API s použitím knihovny SignalR pro přenos hodnot v reálném čase, což je vhodné zejména v režimu ladění (Implementace v modulu *Server*).

Rozhraní je vidět na obrázku 6.5. Skládá se ze tří základních částí:

- Dvě záložky vlevo:
  - Záložka Project umožňuje přidávat a spravovat stránky, uživatele, místnosti a kategorie. Stránky představují samostatné pracovní plochy pro tvorbu konfigurace systému.
  - Záložka Block Library zobrazuje všechny dostupné bloky (tedy ty, co byly systémem načteny jako dostupné). Administrátor je odtud přenese na pracovní plochu pouhým přetažením.
- Pracovní plocha uprostřed zobrazuje vybranou stránku projektu. Umožňuje přesouvat, spojovat a vybírat jednotlivé bloky.
- Okno vlastností vpravo poskytuje detaily o právě vybraném bloku. Je zde jeho detailní popis, seznam parametrů s možností jejich editace a rovněž seznamy jeho vstupů/výstupů s možností vypnutí.



Obrázek 6.7: Uživatelské rozhraní konfigurační aplikace

Spojení s centrální jednotkou, nahrávání a stahování projektu (konfigurace) probíhá přes nabídku *Server*. Tamtéž je možné zapnout režim ladění, ve kterém se u jednotlivých vstupů/výstupů bloků současně zobrazují jejich aktuální hodnoty. Rovněž lze vykonávání programu podle potřeby pozastavit a krokovat.

Připojení konfigurační aplikace na centrální jednotku je indikováno zeleným pruhem v dolní části. Linkový graf v pravém dolním rohu by měl orientačně informovat o využití procesorového času v centrální jednotce.

Celkově byl při vývoji kladen důraz na maximální možnou intuitivnost, jednoduchost a podobnost s jinými obdobnými programy tak, aby nebylo potřeba zdlouhavého školení.

Kapitola 7

Referenční příklady

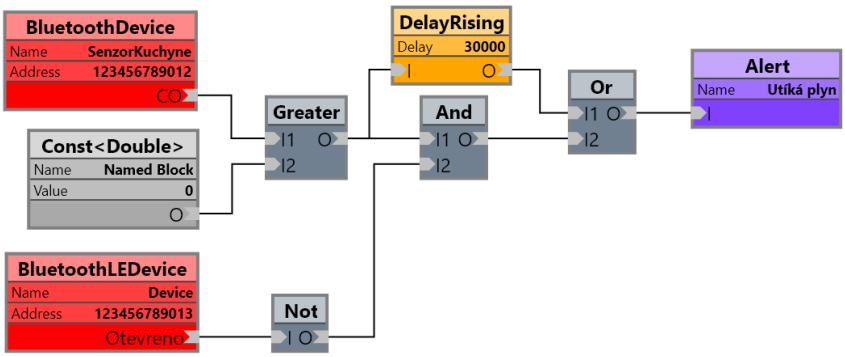
Jelikož je primárním cílem vytvářeného systému možnost dohledu, tak byly vytvořeny některé demonstrační příklady zaměřené právě na tuto oblast. Poskytují dostatečné ověření vhodnosti použitého přístupu pro vytvoření potřebné konfigurace systému. Dále tedy bude představeno několik vzorových příkladů pro použití navrženého domácího systému pro dohled nad zdravotně indisponovanou osobou.

7.1 Utíkající plyn

První příklad je typický ve starších zástavbách, kde je pro vaření stále ještě velmi často používán plynový sporák. V tomto případě jsou použita dvě koncová zařízení:

- SenzorKuchyne (senzor) – Měří koncentraci plynu CO pomocí připojeného chemického senzoru MQ-9.
- OkenniSenzorKuchyne (senzor) – Poskytuje informaci, zda je okno v kuchyni otevřené/zavřené.

Činnost je následující: Pokud koncentrace CO přesáhne nastavenou hodnotu (Maximalni CO = 12) a zároveň není otevřené okno, zobrazí se notifikace „Utíká plyn“ okamžitě. Naopak, je-li okno otevřené, spustí se $30s$ časovač pro vytvoření zpoždění, a teprve po jeho uplynutí bude zobrazena notifikace „Utíká plyn“. Uživatel již mohl únik plynu detekovat například čichem a okno okamžitě otevřít (proto uvedená prodleva).



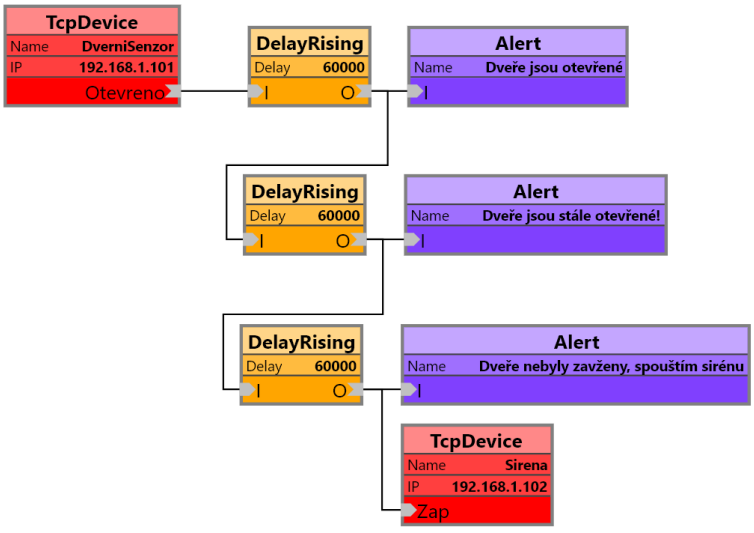
Obrázek 7.1: Konfigurace vzorového příkladu „Utíkající plyn“

7.2 Otevřené dveře

Další příklad je rovněž častý pro starší osoby. Ty si občas již neuvědomují, zda skutečně dostatečně dobře zavřeli vchodové dveře (například po návštěvě, nebo po obdržení pošty). Zde jsou taktéž použita dvě koncová zařízení:

- DverniSenzor (senzor) – Poskytuje informaci, zda jsou vchodové dveře otevřené.
- Sirena (aktuátor) – Při aktivaci se spustí dobře slyšitelný poplach.

Činnost je následující: Pokud jsou dveře otevřené již $60s$, zobrazí se první upozornění a po $120s$ druhé. Po $180s$ se zobrazí poslední upozornění a spustí se dobře slyšitelná siréna. Ta má uživatele upozornit na opomenutí zavření dveří.



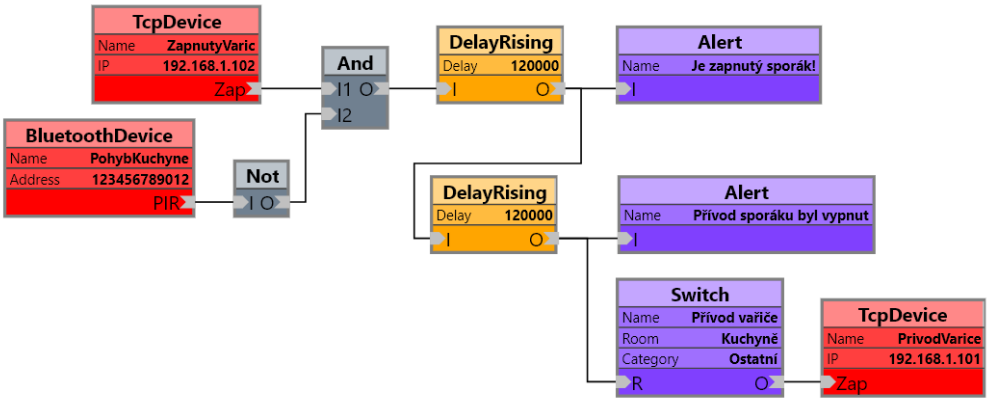
Obrázek 7.2: Konfigurace vzorového příkladu „Otevřené dveře“

7.3 Zapnutý vařič

Třetí zde popisovaná situace bývá také častá u starších osob a to zapomenutí vypnutí vařiče (klasického elektrického). Jsou použita tři koncová zařízení:

- PohybKuchyne (senzor) – Dává na výstupu puls, pokud připojený PIR (pohybový) senzor zaregistruje pohyb v místnosti.
- ZapnutyVaric (senzor) – Obsahuje proudový senzor, detekující zapnutý vařič.
- PrivodVarice (aktuátor) - Připojen na stykač umožňující odpojit přívod energie do vařiče.

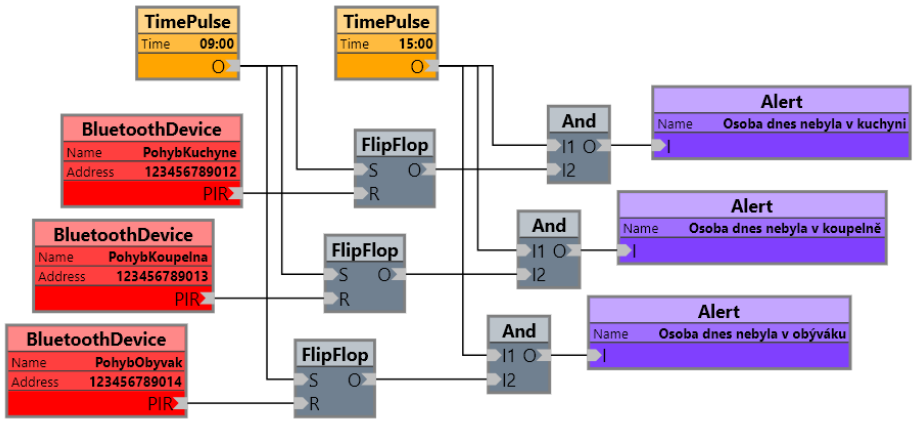
Činnost je následující: Přívod sporáku je ovládán vypínačem ve vizualizaci. Jedině tam je možné ho zapnout. Pokud je vařič zapnutý a po dobu $120s$ nebyl v kuchyni zaregistrován žádný pohyb, zobrazí se notifikace. Po dalších $120s$ se zobrazí druhá notifikace a deaktivuje se vypínač, čímž dojde k vypnutí přívodu energie pro vařič. Pro opětovné zapnutí musí uživatel použít vizualizaci. Pokud je kdykoliv v průběhu čekání detekován pohyb, výstup bloku And se přepne na *false*, sestupná hrana projde blokem DelayRising okamžitě a počítání času se nuluje, a tedy vrací na začátek.



Obrázek 7.3: Konfigurace vzorového příkladu „Zapnutý vařič“

7.4 Přítomnost v místnostech

V tomto posledním uvedeném případě jde o poněkud odlišný scénář, avšak v dohledových systémech vcelku často používaný. Vychází z předpokladu, kdy sledovaná osoba, pokud je v pořádku, navštíví každý den v čase od 9:00 do 15:00 alespoň jednou všechny místnosti v bytě. Proto do každé z nich umístíme PIR (pohybový) senzor, jehož výstup připojíme na resetovací vstup klopného obvodu. V 9:00 jsou všechny klopné obvody aktivovány. V 15:00 se zkontrolují výstupy klopných obvodů a detekuje se, zda některý zůstal ve stavu *true*. Pokud ano, tak osoba některou ze sledovaných místností v uvedeném čase vůbec nenavštívila, a zobrazí se tedy příslušné upozornění.



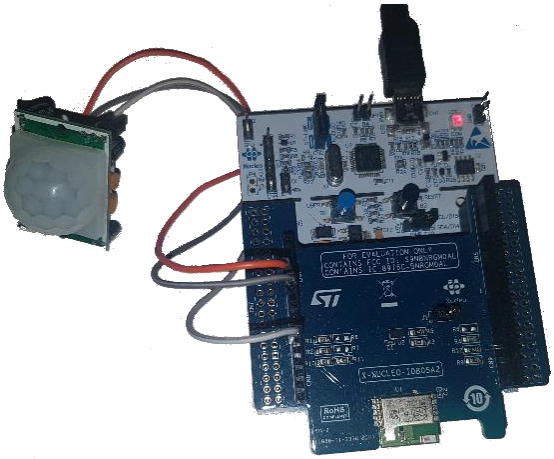
Obrázek 7.4: Vzorový příklad „Přítomnost v místnostech“

Zde uvedené příklady jsou samozřejmě pouze demonstrační (senzory, časování, výstupy). Přesto však vhodně prezentují schopnosti navrženého systému pro jeho primárně zamýšlené použití. Jsou zde uvedeny jako samostatné/oddělené příklady použití, ale ve skutečnosti mohou být samozřejmě v konfiguraci vytvořeny všechny současně. Je tedy zřejmé jak pomocí jednotlivých bloků (předdefinovaných, případně i vlastních), lze sestavit i značně sofistikované konfigurace systému. V reálném prostředí by samozřejmě mohly být zde uvedené „notifikace“ nahrazené jinými výstupy, pro cílovou osobu, například zvuky, světla, hlasy a atd. Případně se mohou notifikace zobrazovat nikoliv sledované osobě, ale osobě sledující (příbuznému indisponovaného člověka).

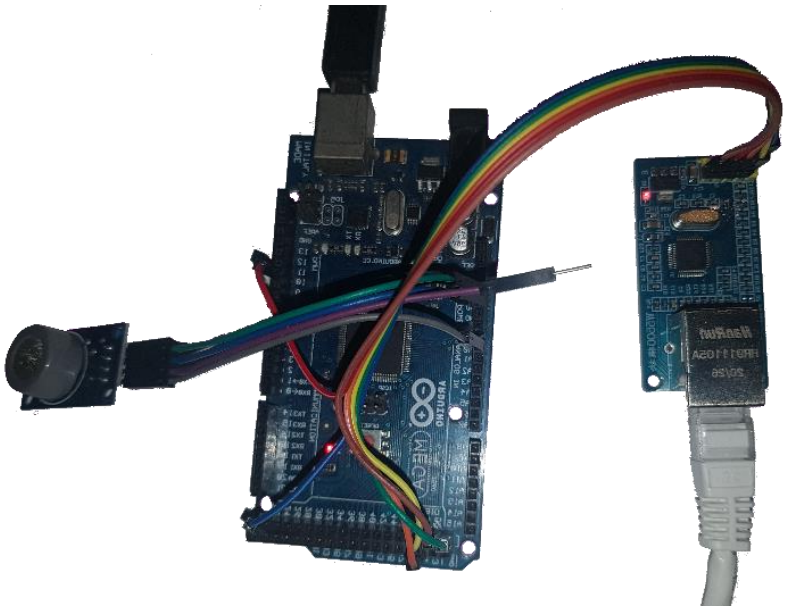
7.5 Realizace koncových zařízení

S vytvořenou knihovnou je tvorba skutečných koncových zařízení velmi zjednodušena. Pro účely vývoje lze například výstup stykače simulovat LED diodou, dveřní kontakt přepínačem. Jako ilustrační příklady jsou na obrázku 7.5 uvedena tři reálně vytvořená zařízení:

a) Pohybový PIR senzor (HC-SR501) s komunikačním kanálem Bluetooth Low Energy. Použitá vývojová platforma je Nucleo 64 s procesorem STM32F303RE. Komunikaci zajišťuje rozšiřující modul BlueNRG.
b) Senzor koncentrace CO MQ-9 s komunikačním kanálem Ethernet. Použitá vývojová platforma je Arduino Mega s procesorem ATMEGA2560. Komunikaci zajišťuje modul s čipem WIZ5500.
c) Nástěnný dotykový vypínač, převzatý z jiného projektu. Použitý procesor je ESP32, komunikační kanál je zde tedy WiFi. Kromě pěti dotykových plošek obsahuje senzor teploty a relativní vlhkosti SHTC3.



a)



b)

c)

Obrázek 7.5: Příklady koncových zařízení

Kapitola 8

Závěr

Cíl práce spočíval v návrhu a částečné demonstrační tvorbě domácího chytrého systému vhodného zejména pro dohledové použití. Ještě přesněji řečeno pro experimentální/školní využití při tvorbě dohledových systémů za účelem ověření správnosti vytvořených senzorů, vhodnosti získávaných dat a rovněž jejich následného zpracování.

Nejprve byl uveden přehled již existujících „chytrých domácích systémů“ avšak určených převážně pouze k řízení/ovládání domácího prostředí. Jak velké komerční, tak i tzv. open-source řešení jsou zaměřeny na připojení zejména komerčních, a tedy již hotových komponent (koncová zařízení a centrální jednotky). Začlenění zcela vlastních amatérských až experimentálních koncových zařízení je do těchto systémů velmi složité až nereálné.

Z tohoto důvodu byl vytvořen návrh zde popisovaného systému s názvem DotHome zaměřeného na poněkud jiné priority. Jde o možnost připojení libovolných vlastních/experimentálních koncových zařízení, dále o využitelnost libovolných běžných (nikoliv jen průmyslových) komunikačních kanálů. Způsob konfigurace byl volen co možná nejvíc přímočarý, tedy grafický. V poslední řadě se jedná o možnost spuštění celého systému na téměř libovolném počítači, tedy i například na běžném domácím PC s operačním systémem MS/Windows.

Pro částečnou realizaci, a tedy pro ověření správné činnosti návrhu systému byly vytvořeny a naprogramovány některé stěžejní části. Systém se jako v každém případě tohoto typu skládá z koncových zařízení, komunikačních kanálů a centrální jednotky.

Pro vzorovou tvorbu koncových zařízení byly vybrány dvě velmi rozšířené procesorové platformy Arduino (pro méně znalé) a STM32 (pro znalejší uživatele). Pro ně byla vytvořena programová knihovna definující datové typy pro ukládání dat ze senzorů a jejich přenos pomocí vybraných komunikačních kanálů do centrální jednotky.

Komunikační kanály byly zvoleny skutečně ty nejdostupnější v běžné výpočetní technice (WIFI, Ethernet, Bluetooth, Sériový port). Nicméně princip vytvořeného komunikačního

protokolu umožňuje jeho použití zcela nezávisle na konkrétním typu komunikačního kanálu. Byly navrženy dvě varianty protokolu, a to textový pro rychlejší přenosové kanály a binární pro pomalejší varianty přenosu.

Stěžejní částí práce je centrální jednotka. Ta vytváří celkovou činnost systému, a obsahuje proto nejvíce programového kódu/vybavení. Její hlavní činností je periodicky číst data z koncových zařízení, vykonávat zadaný program a výsledky opět zapisovat do koncových zařízení. Toto je umožněno pomocí programu vytvořeného z tzv. funkčních bloků a jejich vhodnou konfigurací a propojením.

Další nedílnou součástí je vizualizace pro interakci s uživatelem, implementovaná jako progresivní webová aplikace za použití technologie Blazor/ASP.NET. Core.

Uvedenou činnost centrální jednotky je potřeba nějak nakonfigurovat, tedy vytvořit její program/funkční bloky. Za tímto účelem byla vytvořena samostatná multiplatformní desktopová aplikace, umožňující sestavovat funkční celky z bloků intuitivní metodou drag&drop.

Navržený systém přejímá množství poznatků a činností z těch již existujících (jiná řešení často ani nejsou). Avšak klade důraz na zcela odlišné priority. Je určen zejména pro školní/grantové/experimentální použití. Snaží se definovat a vytvořit pouze potřebný základ pro další doplnění podle aktuální potřeby. Jeho výhodou je právě tímto dosažená univerzálnost.

8.1 Možná rozšíření a vylepšení

Vzhledem k velkému rozsahu problematiky a omezenému (zejména časovému) prostoru není výsledkem systém připravený pro okamžité nasazení. Stěžejní věci byly naimplementovány, avšak pro seriózní nasazení by si systém žádal delší dobu testování, stejně jako odladění případných nedostatků.

Potřeba tvorby vlastních koncových zařízení prezentovaným způsobem je dána zadáním/hlavní myšlenkou. Kromě toho se zde však otevírá celá řada dalších možných vylepšení:

- Využití dalších komunikačních kanálů. Architektura umožňuje rozšíření o libovolné, i stávající průmyslové sběrnice i bez použití prezentovaného protokolu.
- Tvorba funkčních bloků, ať už obecných, vizualizačních, nebo představujících koncová zařízení. Nyní existuje pouze neúplná knihovna standardních bloků, potřebných k základnímu ověření architektury.
- Vylepšení základního běhového algoritmu (*DotHome.Core*). Jedná se především o vyřešení problematiky vytvoření smyček, jež mohou některé specifické případy

vyžadovat. Dále je zde prostor pro optimalizaci hlavní programové smyčky vytvořeného programu, kde namísto periodického vykonávání je možné využít běh řízený událostmi.

- Vylepšení grafického rozhraní vizualizace. V současném stavu jsou vizualizační bloky zobrazené jako boxy ve webovém rozhraní s poněkud jednoduchou grafikou. Použitý framework umožňuje vytvořit daleko graficky sofistikovanější výsledek s použitím technologií HTML, CSS, JavaScript, ovšem vyžaduje určitou zkušenost s tímto typem programování.
- Vylepšení konfigurační aplikace. Implementovány byly pouze základní funkce pro editaci bloků, komunikaci se serverem a základní ladění. Pro větší produktivitu by bylo vhodné například zavést historii změn (CTRL+Z, CTRL+Y) nebo možnost vytvoření vlastního bloku složeného z jiných bloků (po vzoru Subsystem v prostředí Matlab/Simulink).

Příloha A

Literatura a zdroje

[1] "ABB free@home," [Online]. Available: https://new.abb.com/low-voltage/cs/nizke-napeti/produkty/automatizace-bytu-a-budov/produktove-rady/abb-free@home. [Accessed 12. 4. 2021].

[2] "Siemens home automation," [Online]. Available: https://new.siemens.com/global/en/company/stories/industry/factory-automation/an-automated-home-quickly-and-easily.html. [Accessed 12. 4. 2021].

[3] "Teco smart home," [Online]. Available: https://www.tecomat.com/reference/smart-home/. [Accessed 26. 4. 2021].

[4] "Obrázek - Nízké napětí," [Online]. Available: https://nizke-napeti.cz.abb.com/files/document/5875/files/Manual-ABB_FreeHome_systemova-prirucka.pdf. [Accessed 26. 4. 2021].

[5] "Obrázek - Smart Home," [Online]. Available: https://www.smarthome.com/collections/amazon. [Accessed 21. 4. 2021].

[6] "Loxone," [Online]. Available: https://www.loxone.com/cscz/. [Accessed 12. 4. 2021].

[7] "Evon smart home," [Online]. Available: https://evon-smarthome.com/. [Accessed 21. 4. 2021].

[8] "HomeSeer," [Online]. Available: https://homeseer.com/. [Accessed 26. 4. 2021].

[9] "Obrázek - Das intelligente Haus," [Online]. Available: https://www.das-intelligente-haus.de/schlaue-haeuser/Intelligentes-Musterhaus-mit-hybrider-Loxone-Steuerung. [Accessed 12. 4. 2021].

[10] "Home Assistant," [Online]. Available: https://www.home-assistant.io/. [Accessed 12. 4. 2021].

[11] "OpenHAB," [Online]. Available: https://www.openhab.org/. [Accessed 27. 4. 2021].

[12] "Domoticz," [Online]. Available: https://www.domoticz.com/. [Accessed 12. 4. 2021].

[13] "FHEM," [Online]. Available: https://fhem.de/fhem.html. [Accessed 27. 4. 2021].

[14] "Obrázek - Berlinger," [Online]. Available: https://www.berlinger.cz/blog/raspberry-pi-server-pro-senzory/. [Accessed 12. 4. 2021].

[15] "Obrázek - Enectiva," [Online]. Available: https://www.enectiva.cz/img/hw/NLII-CO2-RH-600x496.jpg. [Accessed 29. 4. 2021].

[16] "Obrázek - Shelly," [Online]. Available: https://shop.shelly.cloud/shelly-1-wifi-smart-home-automation#50. [Accessed 12. 5. 2021].

[17] "Obrázek - Domat," [Online]. Available: https://www.domat-int.com/cs/i-o-moduly-a-prevodniky. [Accessed 12. 5. 2021].

[18] "MQTT Version 5.0," [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf. [Accessed 11. 4. 2021].

[19] "Simply Modbus," [Online]. Available: https://simplymodbus.ca/FAQ.htm#Modbus. [Accessed 11. 4. 2021].

[20] K. Townsend, Getting Started with Bluetooth Low Energy, O'Reilly, 2014.

[21] "Obrázek - Nordic Semiconductors," [Online]. Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsds_s140%2FSDS%2Fs1xx%2Fble_protocol_stack%2Fble_protocol_stack.html. [Accessed 8. 5. 2021].

[22] "Algoritmy.net," [Online]. Available: https://www.algoritmy.net/article/1381/Topologicke-usporadani. [Accessed 13. 5. 2021].

[23] "ABB free@home Systémová příručka," [Online]. Available: https://nizke-napeti.cz.abb.com/files/document/5875/files/Manual-ABB_FreeHome_systemova-prirucka.pdf. [Accessed 12. 4. 2021].

[24] C. Robert, A DIY SmartHome Guide, McGraw-Hill, 2020.

[25] M. Price, C# 8.0 and .NET Core 3.0 - Modern Cross-Platform Development, Packt, 2019.

[26] N. Carmine, Mastering STM32, LeanPub, 2017.

Příloha B

Pojmy, zkratky a technologie

Chytrá domácnost: Široký pojem, zahrnující vše od slaboproudé elektroinstalace přes ovládání domu na dálku, automatickou regulaci topení, větrání, až po zcela autonomní přizpůsobování se domu potřebám obyvatele na základě umělé inteligence.

Domácí automatizace: Lze nahlížet jako na synonymum chytré domácnosti, klade však důraz spíše na nižší úrovně (propojení zařízení, způsob ovládání, ne tolik autonomní chování)

LAN: Lokální síť, v kontextu práce rozumíme síť uvnitř domu, typicky za NAT jednotkou. Implementace je WiFi nebo Ethernet.

Internet: Počítačová síť vně domu.

UDP: User Datagram Protocol – nespojový protokol transportní vrstvy modelu OSI. Neposkytuje záruku doručení paketu.

TCP: Transmission Control Protocol – spolehlivý spojový protokol transportní vrstvy modelu OSI.

MCU: Microcontroller unit – mikroporcesor

Arduino: Open source platforma, umožňující programování mikrokontrolerů různých výrobců pomocí jednotného rozhraní. Jedná se spíše o výukový/amatérský projekt. (https://www.arduino.cc/)

ARM: Architektura procesorů - Advanced Risc Machine (https://www.arm.com/why-arm/architecture/cpu)

STM32: Rodina 32-bitových mikrokontrolerů architektury ARM od společnosti ST Microelectronics (https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html)

HAL: Hardware abstracion layer – Soubor knihoven pro čipy STM32, umožňující jejich programování bez nutnosti použití jednotlivých registrů. Tato vrstva sjednocuje kód pro čipy, jejichž hardwarová implementace se může lišit. (https://deepbluembedded.com/stm32-hal-library-tutorial-examples/)

C#: Vysokoúrovňový, objektově orientovaný programovací jazyk, vyvinutý společností Microsoft, kompilovaný do Common Language Infrastructure (https://docs.microsoft.com/en-us/dotnet/csharp/)

.NET Core: Multiplatformní framework, virtuální stroj, umožňující spustit CLI aplikace. Předchůdcem je .NET Framework (https://docs.microsoft.com/en-us/dotnet/fundamentals/)

ASP.NET Core: Součást .NET Core, modulární framework, určený zejména pro tvorbu webových aplikací. (https://docs.microsoft.com/en-us/aspnet/core/)

SignalR: Komunikační knihovna ASP.NET Core, umožňující real-time komunikaci mezi uzly. (https://docs.microsoft.com/en-us/aspnet/core/signalr/)

Blazor: Součást ASP.NET Core, umožňuje tvorbu dynamických webových aplikací pomocí kombinace jazyků HTML, CSS a C# a to jak na serveru, tak ve webovém prohlížeči pomocí WebAssembly. (https://docs.microsoft.com/en-us/aspnet/core/blazor/)

PWA: Progresivní webová aplikace, umožňuje tvorbu klasických webových stránek, ovšem s možností instalace na uživatelská zařízení (PC, telefon). To umožňuje využívat některé funkce, běžně dostupné jen nativním aplikacím (https://web.dev/progressive-web-apps/)

WPF: Windows Presentation Foundation – Grafická knihovna pro tvorbu desktopových aplikací pro Windows v .NET Core. Využívá princip Data-Bindingů a XAML pro definici uživatelského rozhraní. (https://docs.microsoft.com/en-us/dotnet/desktop/wpf/)

AvaloniaUI: Multiplatformní, open source varianta WPF (https://avaloniaui.net/)

XAML: Značkovací jazyk pro tvorbu uživatelských rozhraní (https://docs.microsoft.com/en-us/dotnet/desktop/wpf/xaml/)

Data Binding: Technologie propojení a automatické synchronizace vlastností objektů modelu s prvky uživatelského rozhraní. (https://docs.microsoft.com/en-us/dotnet/desktop/wpf/data/)

Dependency Injection: Návrhový vzor pro tvorbu závislostí mezi objekty univerzálním, automatickým způsobem (https://www.tutorialsteacher.com/ioc/dependency-injection)

UML: Unified modelling language – nástroj pro grafické znázornění softwarových systémů (https://cs.wikipedia.org/wiki/Unified_Modeling_Language)

GUI: Graphical User Interface – Uživatelské rozhraní programu.