VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DISTRIBUOVANÝ ŘÍDICÍ SYSTÉM S DYNAMICKY MODIFIKOVATELNÝMI UZLY

DISTRIBUTED CONTROL SYSTEM WITH DYNAMICALLY EVOLVABLE NODES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. RADIM KŘEK**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Křek Radim, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Distribuovaný řídicí systém s dynamicky modifikovatelnými uzly**
Distributed Control System with Dynamically Evolvable Nodes
Kategorie: Operační systémy
Zadání:

1. Prostudujte problematikou distribuovaných řídicích systémů (DCS), internetu věcí (IoT) a systémů dispečerského řízení a sběru dat (SCADA).
2. Seznamte se s existující pokusnou implementací operačního systému pro rekonfigurovatelný IoT uzel v jazyce MicroPython na platformě ESP8266/ESP32, komunikující protokolem MQTT.
3. Navrhněte celkovou architekturu DCS s možností dynamicky modifikovat software na uzlech DCS prostřednictvím MQTT. Zaměřte se na operační systém uzlů, databázi, uživatelské rozhrani (HMI) a prostředky pro monitorování a správu systému.
4. Navržený systém realizujte s využitím existujících i vámi nově implementovaných komponent, proveďte testování za pomoci vhodné aplikace a vyhodnoťte dosažené výsledky.

Literatura:

- Drahovský, P.: Rekonfigurovatelný IoT uzel na bázi ESP8266/ESP32. Bakalářská práce FIT VUT v Brně. URL: https://wis.fit.vutbr.cz/FIT/db/dir.php/rp/2017/BP/20280.pdf

Při obhajobě semestrální části projektu je požadováno:

- Body 1, 2 a část návrhu.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Janoušek Vladimír, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Tato práce se zabývá vytvořením dynamicky modifikovatelného uzlu, který poté může spolupracovat s dalšími uzly a dohromady tak vytvořit distribuovaný řídicí systém. Jednotlivé uzly pak spolu komunikují pomocí protokolu MQTT. Pro vytvoření byly po hardwarové stránce použity desky ESP8266 a ESP32. Operační systém je vytvořen v jazyce MicroPython a podporuje nahrávání uživatelských aplikací, které jsou napsány v tomtéž jazyku. Dále je popsáno vytvoření monitorování na platformě Raspberry Pi, které kontroluje chování sítě uzlů. Celý systém tak lze využít například pro řízení inteligentních domů.

Abstract

This thesis describes creation of dynamically evolvable node, which will cooperate with other nodes. Group of these nodes will then create a distributed control system. The MQTT protocol is used for communications purposes between individual nodes. As hardware platform is used ESP32 and ESP8266. Whole operating system is written in MicroPython and supports a live uploading of user applications written in the same language. Later in thesis is decribed creation of monitoring node on Raspberry Pi, which control network. Complete system can be then used to control a intelligent house.

Klíčová slova

ESP, ESP8266, ESP32, Micropython, Distribuovaný řídicí systém, SCADA, Dynamicky modifikovatelný uzel, Asyncio, Domoticz

Keywords

ESP, ESP8266, ESP32, Micropython, SCADA, Distributed Control System, Dynamically evolvable node, Asyncio, Domoticz

Citace

KŘEK, Radim. *Distribuovaný řídicí systém s dynamicky modifikovatelnými uzly.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Vladimír Janoušek, Ph.D.

Distribuovaný řídicí systém s dynamicky modifikovatelnými uzly

Prohlášení

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

....
Radim Křek
20. května 2019

Obsah

1 Úvod **2**

2 Systémy pro řízení a Internet věcí **3**
2.1 Distribuované systémy 3
2.2 SCADA systémy 4
2.3 Internet věcí 5

3 Hardwarové platformy **8**
3.1 Arduino 8
3.2 PyBoard 10
3.3 ESP 11

4 Návrh systému **14**
4.1 Ukázkový problém k řešení 14
4.2 Analýza 15
4.3 Použítý hardware a software 16
4.4 Uživatelské aplikace 16
4.5 Komunikační protokal 17

5 Implementace **20**
5.1 Knihovny jazyka Micropython 20
5.2 Struktura systému 21
5.3 Uživatelské aplikace 27
5.4 Ukládání hodnot a Domoticz 31

6 Nástroje pro správu systému **32**
6.1 Remote control 32
6.2 Monitoring 33

7 Demo příklad **34**
7.1 Příprava ESP 34
7.2 Nastavení uzlů 36

8 Závěr **40**

Literatura **42**

Kapitola 1

Úvod

Internet věcí se stává v dnešní době čím dál více populární. Proto se není třeba divit že jeho principy nalezneme skoro v každém řídicím systému. S tím se také pojí obrovské množství různých hardwarových platforem určených pro vývoj právě těchto systémů, které jsou dnes k dispozici. Velmi tomu také napomáhá skutečnost, že cena drobné elektroniky jde čím dál více dolů a více lidí si ji tak může dovolit. Ruku v ruce s vývojem různých modulů a čipů jde vývoj softwaru. A to jak profesionálních uzavřených řešení tak komunitních. Jedním takovým systémem je experimentální implementace operačního systému pro ESP8266/ESP32 určená pro distribuovaný řídicí systém schopný rekonfigurace jednotlivých uzlů za běhu nazvanáý MPOS. Ta vznikla jako bakalářská práce Petera Drahovského na Fakultě informačních technologií Vysokého učení technického v Brně. [13]

Cílem této práce je na zmíněnou experimentální implementaci navázat a zmíněný operační systém rozšířit a zdokonalit. Hlavním cílem pak bude zdokonalení souběžného běhu uživatelských aplikací, jelikož moduly ESP nedisponují podporou více vláken. V rámci komunikačního protokolu pak dojde k úpravě, aby bylo možno plně využít schopností MQTT protokolu. Bude ale dbáno na zpětnou kompatibilitu s MPOS.

První část textu se bude zabývat obecnou teorií řídicích systému a internetu věcí společně s popisem nejpopulárnějších hardwarových platforem využívaných v tomto odvětví. Dále bude rozebrána problematika vytvářeného systému společně s jeho návrhem včetně návrhu komunikačního protokolu. Poslední částí textu bude samotná implementace operačního systému.

Kapitola 2

Systémy pro řízení a Internet věcí

Kdykoliv chceme vytvořit systém, který bude složen z více částí, je vždy potřeba myslet na to jak bude řízen. Velmi často se využívá Centrální řízení, tedy že jedno zařízení je povýšeno na centrální uzel a ten řídí a koordinuje všechny ostatní. Jedná se o nejjednodušší přístup, který je také velmi jednoduchý na správu, jelikož je zde pouze jeden uzel, který je náchylný na poruchy. Avšak pokud na tomto uzlu nastane chybový stav, celý systém je paralyzován. [17]

Proto se často, zvláště pak v místech kde je potřeba vysoká stabilita systému, přistupuje k jiným typům řízení, které nejsou závislé na centrálním bodě. Právě těmito systémy se bude zaobírat následující kapitola. K těmto typům řízení patří například distribuované systémy nebo decentralizované systémy. Decentralizované systémy jsou pouze rozšířením centralizovaného systému, viz obrázek 2.1, a ty tedy již dále rozebírány nebudou.

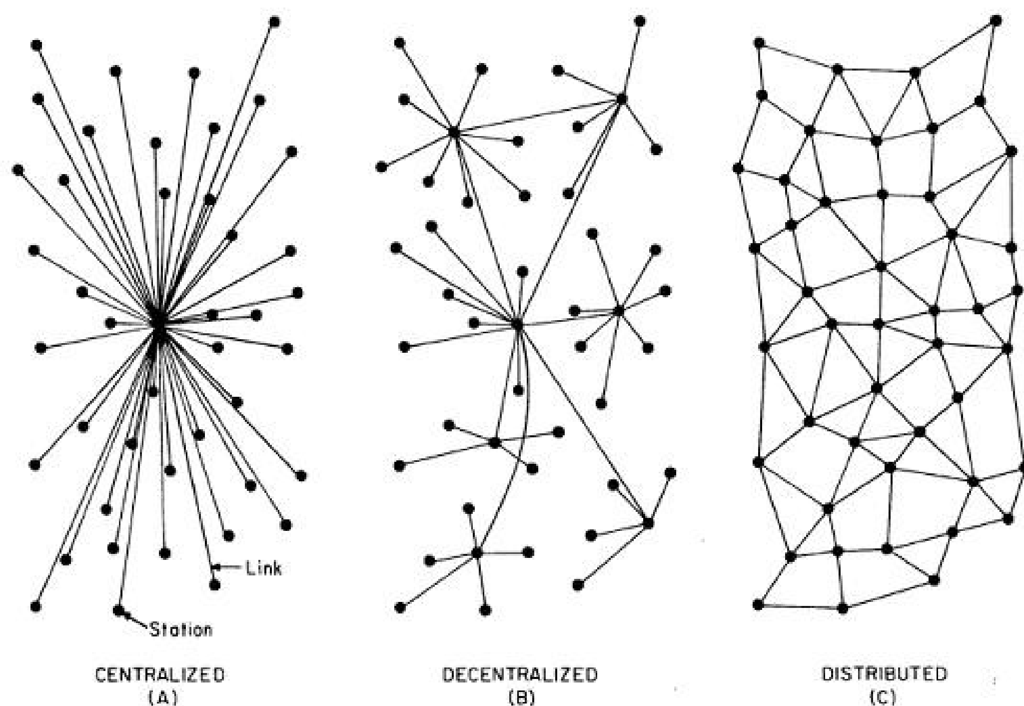
2.1 Distribuované systémy

S pojmem Distribuované systémy, se velmi často používá ještě pojem Programovatelné logické automaty[1]. I když jsou tyto pojmy velmi stěžejní v rámci automatizace a řízení a často se prolínají, tak jsou ale velké rozdíly v jejich funkci a použití. Dále se tedy bude text zaobírat jen distribuovanými systémy. [8]

Tyto systémy se skládají z velkého množství různých senzorů a řídicích členů. Mezi těmito členy není žádný, který by figuroval jako nadřízený a všechny tedy pracují nezávisle a pouze si mezi sebou předávají informace. V systému se pohybuje velké množství zpráv a právě z tohoto důvodu se Distribuované systémy, dále jen DCS, začali rozšiřovat ve chvíli, kdy komunikační sběrnice dosáhly potřebné rychlosti a spolehlivosti. Výhoda struktury, kdy nejsou uzly závislé na centrálním prvku, se nejvíce pozná v případě poruchy. Pokud nastane chybový stav jednoho členu, pak zbytek sítě může pracovat dále, i když třeba jen v omezeném režimu. DCS se také vyznačuje velkou škálovatelností, kdy stačí nový člen pouze připojit k síti. [8, 18, 3]

S Distribuovanými systémy se nejčastěji setkáme ve velkých továrnách nebo podnicích, kde je potřeba řízení velkého množství zařízení a je potřeba vysoké spolehlivosti. Avšak v dnešní době kdy se velké množství různých technologií dostává do každodenního života, se tak setkáváme s DCS velmi často i mimo továrny.

[1] https://controlstation.com/dick-morley-story-plc/



Obrázek 2.1: Porovnání schemat různých systémů řízení.

Převzato z:
`https://medium.com/delta-exchange/centralized-vs-decentralized-vs-distributed-41d92d463868`

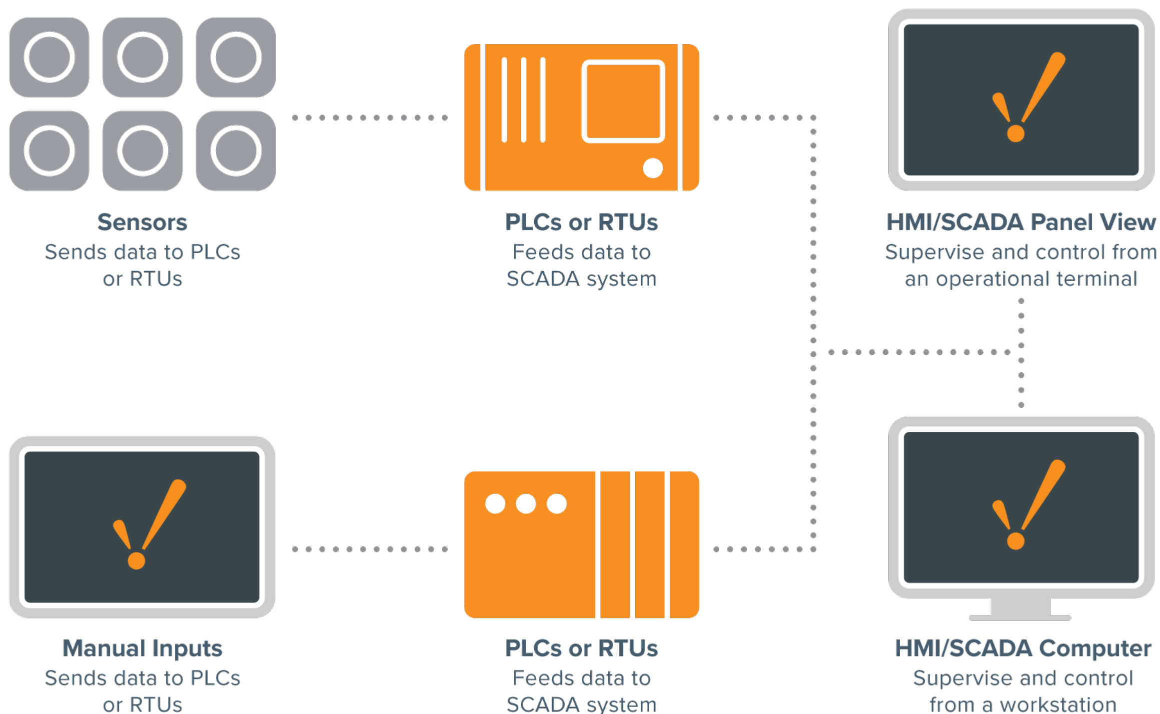
2.2 SCADA systémy

SCADA, neboli Supervisory Control And Data Acquisition, je vzdálený monitorovací a řídicí systém. V dnešní době se se systémy typu SCADA můžeme setkat kdekoliv, ale původně vznikly pro použití v industriálních podnicích. Samotná SCADA však není plnohodnotný řídicí systém. Jedná se spíše o jakýsi nadřazený prvek, který monitoruje co se děje např. v DCS, které ovládá továrnu. [1, 6]

Schéma systému je tedy takové, že senzory jsou připojeny na PLA nebo RTU,[2] které převádí data ze senzorů na digitální data. Dále je v systému SCADA server, který ukládá všechny hodnoty, a to i historické, a stará se o jejich zpracování. RTU nebo PLA pak se serverem komunikují nejčastěji pomocí LAN nebo WIFI, ale můžou být použity i alternativní sběrnice. Posledním členem je HMI[3], což je nejčastěji display, který zobrazuje data o systému v reálném čase. Celý systém pak může vypadat například tak, jak je zobrazeno na obrázku 2.2.[10][7]

[2] Remote Terminal Unit
[3] Human-Machine Interface

Při zobrazování dat se dbá na jejich grafickou podobu. A to tak aby byli pro uživatele co nejsnadněji pochopitelné. Obvykle se tak na obrazovce zobrazuje nákres řízené části společně s užitečnými daty. Takovýto nákres můžeme vidět na obrázku 2.3, který zároveň ukazuje že HMI nemusí být umístěn stacionárně v rámci budovy ve které je SCADA použita, ale může se jednat o mobilní zařízení, které k datům přistupuje vzdáleně. [10, 20]



Obrázek 2.2: Základní schéma SCADA systému

Převzato z: https://inductiveautomation.com/resources/article/what-is-scada

2.3 Internet věcí

Internet věcí, zkracovaný na IoT, se považuje za přímého následníka SCADA systémů. Avšak definovat přesně tento pojem je komplikované, jelikož se objevuje několik různých definic. Nejčastěji se ale setkáme s definicí, která jej popisuje následujícím způsobem. Internet věcí je systém, obsahující zařízení s unikátní identifikátorem, které jsou schopny zasílat data prostřednictvím sítě bez potřeby zásahu uživatele. Samotné zařízení, nebo chceme-li „Věc“, pak může být cokoliv od senzoru pro měření teploty po implantát v lidském těle nebo jakýkoliv jiný člověkem vytvořený objekt, kterému je možno přiřadit unikátní identifikátor, nejčastěji IP adresu, a je schopen zasílat data. [21, 12]

Obrázek 2.3: Zobrazení dat SCADA systému.

Převzato z: `https://inductiveautomation.com/resources/article/what-is-scada`

Samotný pojem Internet věci poprvé použil Kevin Ashton v roce 1999 v prezentaci pro Procter & Gamble. Avšak první nápady na přidání inteligence k senzorům se objevují již v dřívějších letech. Za první zařízení které můžeme zařadit do kategorie IoT, se považuje automat na nápoje ze začátku osmdesátých let umístěný na Carnegie-Mellonově univerzitě, kde pomocí webového rozhraní bylo možno zjistit, zda je automat prázdný nebo ne. Největšího rozmachu se ale IoT dostává až v posledních letech, kdy se objevuje velké množství cenově dostupného hardwaru. Další skutečností která pomohla k rozšíření, bylo nasazení IPv6 adres, které poskytují velké množství adres, které můžou být jednotlivým zařízením přiřazena. Dnes se ve světě setkáme s velkým množstvím různých využití IoT, od malých uživatelských projektů přes firemní až po velké industriální systémy. Ve zmiňovaném uživatelském segmentu se pak nejčastěji jedná o chytrou domácnost. [12, 21, 22]

Samotná komunikace jednotlivých zařízení, probíhá na různých médiích a to v závislosti na konkrétním zaměření IoT systému. Jednotlivá média se pak nejvíce liší v rychlosti komunikace. Existují také sítě, které se hodí tehdy, kdy nemá zařízení stálé připojení k napájení a potřebuje tedy co nejvíce šetřit energii. Nejznámější takové sítě jsou Lora[4] a Sigfox,[5] které mají pokrytí i v Česku. Tyto sítě jsou velmi pomalé a dokáží přenést pouze malé množství dat, ale oproti klasickému internetovému připojení, které je v IoT nejpoužívanější, mají velmi malou energetickou náročnost. Podobnou sítí, která ale není tak rozšířená je NB-IoT.[6]

[4] https://lora-alliance.org/
[5] https://www.sigfox.com
[6] https://www.iot-portal.cz/mapa-pokryti/

Kdykoliv připojujeme jakékoliv zařízení k internetu je potřeba řešit i jeho zabezpečení. V případě Internetu Věcí se připojují tisíce různých zařízení. S tím tedy vzniká velké množství nezabezpečených uzlů. Největším a nejznámějším případem, který poukázal na problém zabezpečení IoT byl malware Mirai, který napadl velké množství IoT zařízení, které zapojil do BotNetu. Ten byl poté použit k DDoS útokům na velké množství webových služeb. Jelikož v roce 2017 se odhadoval počet aktivních IoT zařízení na 8,3 milionů a prognózy do další let jsou takové, že zařízení bude čím dál více, je zabezpečení IoT nutností. [22, 16, 9]

Kapitola 3

Hardwarové platformy

V následující kapitole jsou rozebrány aktuální hardwarové platformy, které jsou dostupné a vhodné k použití pro distribuovaný řídicí systém řešený v této práci. U každé platformy bude její základní popis a bude zde uvedeno proč není použita pro tuto práci. Platforma ESP pak bude popsána více podrobně a to včetně možností jak se dá tato platforma programovat.

Platforma Raspberry Pi nebude popisována vůbec i když bude využita pro běh podpůrných nástrojů a služeb. Tato platforma je v dnešní době velmi rozšířená a má velkou komunitu, proto není třeba ji nijak představovat.

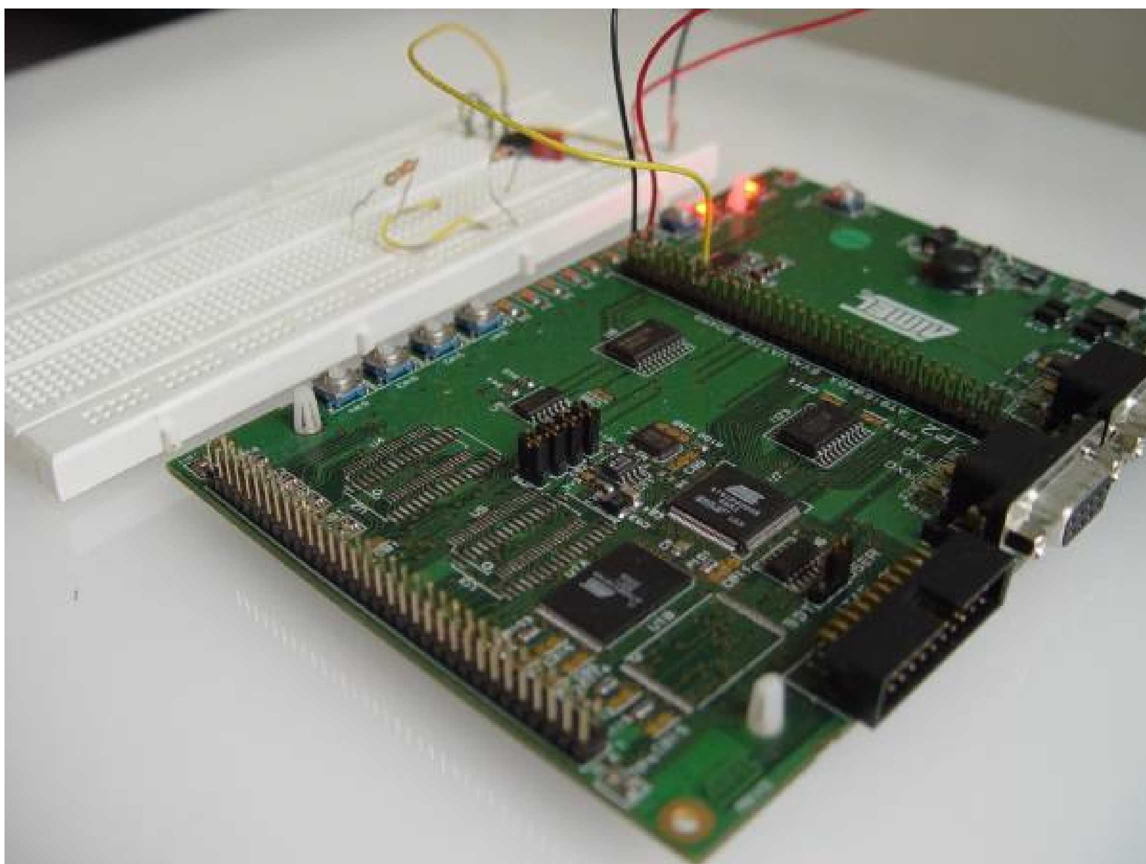
3.1 Arduino

Pravděpodobně nejznámější hardwarovou platformou, která se využívá pro IoT je právě Arduino. Začátky této platformy jsou z roku 2003, kdy Hernando Barragán studující na univerzitě IDII[1] v Itálii, psal diplomovou práci, jejímž cílem bylo zjednodušit práci s elektronikou a zpřístupnit ji více lidem. Výsledkem se stala deska Wiring, která na začátku používala mikrokontroler Paralax,[2] ale později přešla na mikrokontroler Atmel. Prototyp desky s mikrokontrolerem Atmel je zobrazen na obrázku 3.1. Právě tato deska se pak stala základem pro budoucí Arduino. V roce 2005 další studenti univerzity IDII, Massimo Banzi a David Mellis, přidali k desce Wiring kompatibilitu s mikrokontrolerem ATmega8 a o něco později vytvořili vlastní kopii desky právě s tímto kontrolerem a nazvali ji Arduino. [11]

Od roku 2006, kdy byla oficiálně vydána první deska Arduino Serial, bylo vydáno nezměrné množství různých desek Arduino. Od základních prototypovacích desek jako je Arduino UNO, přes IoT řešení s Arduino Nano až po profesionální desku Arduino Industrial 101. Celkový přehled vydaných desek pak můžeme nalézt například na webové stránce Core Electronics.[3] Nejrozšířenější desky se pak dočkaly levných Čínských kopií se stejnými vlastnostmi, proto se s Arduinem lze dnes setkat prakticky kdekoliv. Velkou výhodou Arduina je také velká komunita, která se vytvořila okolo této platformy. [4]

[1]Interaction Design Institute Ivrea
[2]https://www.parallax.com/
[3]https://core-electronics.com.au/tutorials/history-of-arduino.html

Pro DCS řešený v této práci jsou vhodné modely Nano, Micro případně jakékoliv modely z kategorie IoT (Arduino Uno wifi, rodina MKR). První dva zmíněné modely jsou vhodnými kandidáty z hlediska velikosti a hlavně díky cenové dostupnosti. Bohužel ani jeden z nich neobsahuje žádný hardware pro připojení do sítě (Lan, wifi), což je při budování IoT velký problém. Tento problém řeší modely z rodiny IoT, které většinou obsahují wifi modul. Bohužel problémem je cena jednotlivých modelů, která se pohybuje od 700 korun nahoru. Výše bylo sice zmíněno že existují levnější Čínské klony, bohužel ty jsou ale pouze pro desky, které nemají vestavěný síťový modul.



Obrázek 3.1: Deska Wiring s mikrokontrolerem Atmel AT91R40008

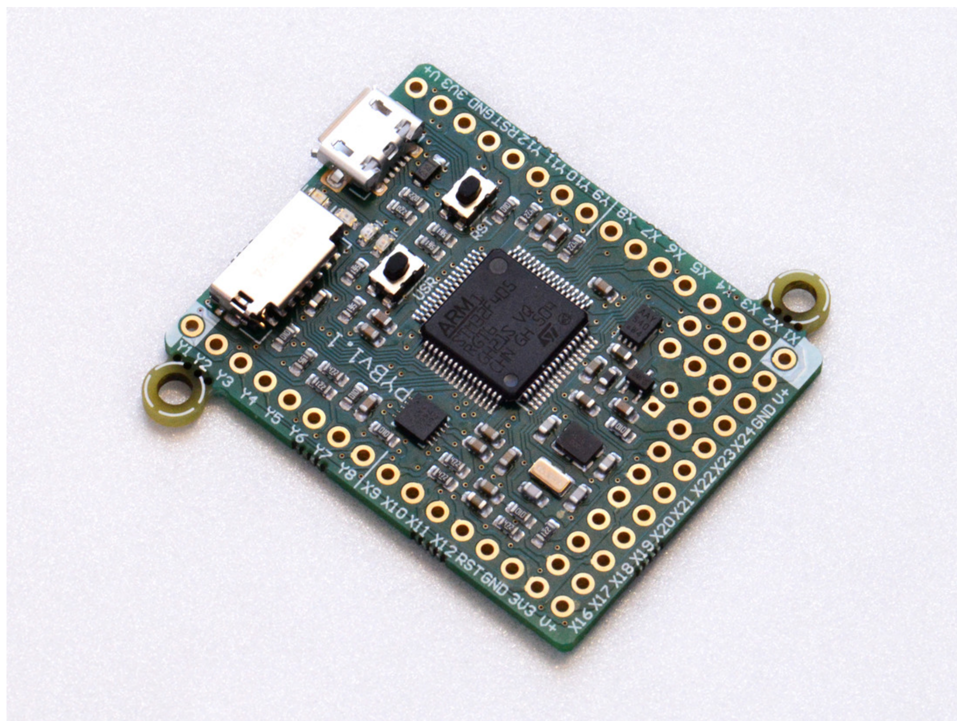
Převzato z: `https://arduinohistory.github.io/`

3.2 PyBoard

PyBoard je jednou z měně známých hardwarových platforem, které jsou vhodné pro IoT. V roce 2013 založil Damien George z university v Cambridge na portálu Kickstarter úspěšnou kampaň[4] pro vytvoření programovacího jazyka MicroPython,[5] který je zcela novou implementací populárního jazyka Python 3, zaměřenou pro běh na mikrokontrolerech. Společně s ním také vydal první verzi desky PyBoard, zobrazenou na obrázku 3.2, která je hlavní platformou pro kterou je Micro Python vyvíjen. [24]

Samotná deska pak obsahuje výkoný mikrocontroler STM32F405RG. Deska dále obsahuje velké množství pinů, které umožňují připojení různých senzorů. PyBoard je dále přizpůsoben pro ovládání až čtyř servo motorů. Při návrhu bylo také počítáno s napájením desky pomocí baterie a tak kromě základních vstupů pro napájení je na desce samostatné místo pro připojení např. knoflíkové baterie, která bude udržovat modul RTC[6] hodin i po ztrátě klasického napájení. [5]

Samotná deska je velmi výkoná a množství vstupů je velké plus pro využití v segmentu IoT. Bohužel jedinou vadou je absence síťového modulu. Micro Python a samotný PyBoard je sice nachystán na připojení několika různých síťových modulů, avšak cena samotné desky, která se pohybuje kolem 800 korun, společně s cenou wifi modulů dělá z PyBoardu poměrně drahou záležitost.



Obrázek 3.2: Deska PyBoard v1.1

Převzato z: `https://store.micropython.org/product/PYBv1.1`

[4] https://www.kickstarter.com/projects/214379695/micro-python-python-for-microcontrollers
[5] https://micropython.org/
[6] Real Time Clock

3.3 ESP

Další platformou hojně užívanou pro tvorbu IoT systémů jsou desky využívající čipy od firmy Espressif Systems[7] označované jako ESP. Tyto čipy byli původně určeny pro doplnění wifi konektivity pro již existující mikrokontrolery. To dokládá například wifi shield[8] pro populární Arduino Uno, využívající čip ESP8266. Samotných čipů, které firma Espressif vytvořila je velké množství, proto budou v následující kapitole rozebrány pouze ty nejrozšířenější. Tedy ESP8266 a jeho výkonnější nástupce ESP32. [23]

ESP8266

Tento čip se dostal do prodeje v roce 2013 a již velmi brzo po jeho uvedení se začaly objevovat různé vývojové kity. Patrně nejznámější je vývojový kit NodeMCU, na obrázku 3.3, který je vybaven USB převodníkem pro snadné programování ESP čipu a vyvedením pinů na standartní rozestupy 2,54mm[9]. Cena této desky je velmi příznivá. V českých E-shopech se pohybuje kolem 200 korun a při objednání z asijských E-shopů pak zhruba 50 korun[10].

Samotné ESP, s výrobním označením modulu jako ESP-12E, pak obsahuje jednojádrový procesor Xtensa LX 106 běžící na frekvenci 80MHz, ale se schopností přetaktování na dvojnásobnou rychlost, což poskytuje např. oproti Arduinu Nano s 16MHz[11] vysoký nárůst. Celková velikost paměti RAM je 96KB, avšak přibližně polovina tohoto prostoru je v základu zabrána kódem, který se stará o ovládání wifi modulu. Flash pamet je připojena pomocí SPI a může mít velikosti od 512KB až po 4MB. Pro komunikaci s okolím je přichystáno 16 GPIO[12] pinů, kdy pro volné použití je deset z nich a na všech je dostupná PWM modulace. Na jednom pinu, je navíc dostupný 10 bitový AD převodník. Velkou nevýhodou tohoto modulu je skutečnost, že se pro správnou funkci wifi modulu je třeba velmi častého předávání programového řízení kódu pro jeho správu. Tím pak přichází uživatelská aplikace o část výkonu, jelikož ESP nepodporuje běh programu ve více vláknech. [23, 15]

Dostupné periferie [15]

- **I^{2}C -** podporuje režimy Master a Slave, s rychlostí 100MHz
- **I^{2}S -** obsahuje jedno vstupní a jedno výstupní rozhraní
- **UART -** má dvě rozhraní, ale pouze jedno je plnohodnotné. S tím že druhé rozhraní figuruje jako debug port.
- **SPI -** k dispozici jsou opět dvě rozhraní, ale SPI0 je použito pro připojení FLASH paměti, proto uživatel může využívat pouze rozhraní SPI1

[7] https://www.espressif.com/
[8] https://arduino-shop.cz/arduino/1457-esp8266-esp-12e-ota-wemos-d1-ch340-wifi-arduino-ide-uno-r3-1478466175.html
[9] https://en.wikipedia.org/wiki/Pin_header
[10] Ceny jsou orientační ze dne 12. 1. 2019
[11] https://www.arduino.cc/en/products.compare
[12] General-purpose input/output

Obrázek 3.3: NodeMCU v3 s čipem ESP8266

Převzato z: https://www.root.cz/clanky/nodemcu-a-jeho-verzie-doska-s-wifi-cipom-esp8266/

ESP32

Jedná se o přímého nástupce čipu ESP8266, který byl představen v roce 2016. ESP32, zobrazený na obrázku 3.4, je výkonnější náhradou, která má navíc vyřešeny největší problémy které trápily starší čip. Navíc byl přidán další komunikační modul a to Bluetooth s podporou verze 4 a BLE.[13] [23]

Největšího zlepšení se modulu dostalo ve změně procesoru na typ Xtensa LX6, což je dvoujádrový procesor běžící na frekvenci 160MHz s možností přetaktování na 240MHz. Díky této změně pak jedno jádro vždy ovládá wifi modul a druhé je plně k dispozici uživateli. Dále se rozrostla velikost integrované paměti RAM a to na 500KiB. Zůstala možnost připojení externí Flash paměti s podporou až do velikosti 16MB. Přínosem je také hardwarová akcelerace šifrování AES a SSL. Kromě přesnějšího A/D převodníku je v čipu nově umístěn i D/A převodník. Počet GPIO pinů se rozrostl na 36, a 10 z nich má dokonce podporu pro vstup kapacitní dotykové vrstvy. Podle technické dokumentace by měl být také čip více úspornější. [23, 14]

Dostupné periferie [14]

- **SPI -** Počet rozhraní se rozrostl na čtyři s tím že SPI0 opět slouží pro komunikaci s Flash pamětí
- **I^{2}C -** počet rozhraní zůstal stejný, s přenosovou rychlostí 400Kb/s
- **I^{2}S -** stejné jako u ESP8266
- **UART -** na desce jsou tři plnohodnotná rozhraní s maximální rychlostí přenosu 5Mb/s
- **SDIO -** čip získal podporu pro komunikaci s SD kartami

[13] Bluetooth Low Energy

Obrázek 3.4: ESP-32S

Převzato z:
https://arduino-shop.cz/arduino/1614-esp-32s-wifi-bluetooth-modul-dvoujadrovy-1496044396.html

Tvorba softwaru pro ESP

Při vydání čipu ESP8266 nebyla žádná oficiální cesta jak tento čip programovat a bylo dostupné pouze API jak jej využít při stavbě zařízení. Poté co však komunita sama přišla na způsob programování a ESP8266 se stalo velmi populární, vydala firma Espressif oficiální SDK. U nového čipu ESP32 pak již vydala SDK přímo při vydání čipu. Díky tomu je je nyní k dispozici velké množství programovacích jazyků a různých frameworků kompatibilních s těmito čipy. Prvním a pravděpodobně i nejrozšířenějším je staticky kompilované C s rozšířeními pro Arduino, tedy přesněji ESP lze pak programovat pomocí stejného prostředí a stejných knihoven jako populární desky Arduino. Za zmínku také stojí NodeMCU, vyvíjené společně s vývojovými deskami NodeMCU obsahujícími ESP8266. Tento framework je založen na jazyku Lua. Dále můžeme nalézt porty jazyků jako je Javascript (framework Espruino), LISP nebo Basic. Velmi oblíbeným jazykem pro programování čipu, který lze zařadit na stejnou úroveň popularity jako programování v jazyce C, je Micropython. [2]

Jazyk Micro Python byl již vzpomenut v kapitole 3.2 popisující desku PyBoard. Port tohoto úspěšného jazyka pro čip ESP8266 byl opět financován pomocí Crowd Fungingové stránky Kickstarter.[14] Ve srovnání s deskou PyBoard, pro kterou byl Micro Python originálně vyvíjen ztrácí port pro ESP jen několik málo funkcí, které jsou navíc vesměs "hardware specific", tedy určeny přímo pro konkrétní harware. Port pro novější čip ESP32 je na tom o poznání hůře, kdy místy chybí i funkcionalita, která je na ESP8266 dostupná. To je hlavně důsledek doby, kterou je ESP32 na trhu. V této době teprve pomalu nahrazuje starší ESP a vývoj jazyka se tak teprve dostává do popředí.

Velká část knihoven, na které je uživatel používající Python 3 zvyklý, má svůj port pro MicroPython. K dispozici je pak také μPip pro instalaci těchto knihoven a REPL[15] pro ovládání systému na desce. Ke knihovnám můžeme přistupovat jako k obyčejnému programu a nahrát je do Flash paměti a odtud je poté spouštět, nebo je lze přidat při kompilaci přímo do firmwaru. Při přidávání knihoven přímo do firmwaru, mohou být knihovny psány kromě Pythonu také v jazyce C. Tyto knihovny pak můžou být rychlejší než při spuštění z Flash paměti.

[14] https://www.kickstarter.com/projects/214379695/micropython-on-the-esp8266-beautifully-easy-iot
[15] read–eval–print loop

Kapitola 4

Návrh systému

Cílem této práce je vytvoření systému, který bude schopen načítat a spouštět uživatelské aplikace. Celý systém bude postaven nad konceptem distribuovaných systému. Pro připomenutí takový systém pak není náchylný na případný výpadek uzlu, jelikož zbytek systému může pokračovat v činnosti na rozdíl od centralizovaného řešení. Systém je pak také dobře škálovatelný.

Návrh systému nebude probíhat od základu, ale bude navazovat na experimentální implementaci takového systému pojmenovanou MPOS[1], která vznikal jako bakalářská práce na Fakultě Informačních Technologií na Vysokém učení technickém v Brně. Systém vytvořený v této práci se tak bude snažit být navenek co nejvíce podobný původnímu MPOS, kvůli kompatibilitě komunikace v případě že by byly oba systémy nasazeny v rámci jedné sítě. Původní experimentální implementace navíc stavěla na systému PNOS[2] se kterým se snažila být také co nejvíce kompatibilní. [13]

Systém PNOS je distribuovaný řídicí sytém postavený na řízení pomocí Petriho sítí. Tento operační systém obsahuje PNVM[3], který spouští uživatelské aplikace, které jsou popsány Petriho sítěmi reprezentovanými pomocí bytekódu.

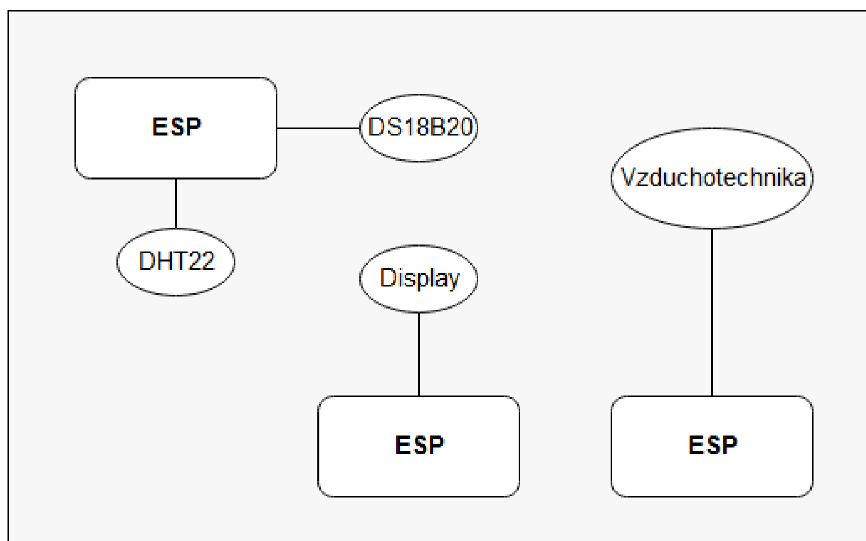
4.1 Ukázkový problém k řešení

Před samotným návrhem systému je potřeba nastínit nějaký problém, který bude pomocí tohoto systému řešen. Díky tomu pak bude možné se odkazovat na konkrétní prvky systému pro lepší pochopení.

Uvažujme skladovací prostory místní galerie. V takovýchto místnostech musí být udržována konstantní teplota a vlhkost pomocí složité vzduchotechniky. Pro potřeby prezentace systému z této práce podmínky lehce zjednodušíme. V místnosti se bude měřit teplota a vlhkost a při překročení horní hranice stanovených limitů, se vyšle signál pro sepnutí vzduchotechniky.

[1] Micro Python Operating System
[2] Petri Nets Operating System
[3] Petri nets virtual machine

Pro takovýto jednoduchý systém budou stačit tři moduly ESP. K prvnímu budou připojeny senzory pro měření teploty a vlhkosti. Další ESP bude figurovat jako ovládací panel systému. Tedy bude k němu připojen display s tlačítky pro nastavení limitů teploty a vlhkosti. Poslední uzel bude zapínat vzduchotechniku v případě potřeby. Pro přehlednost je jednoduchý náčrt na obrázku 4.1



Obrázek 4.1: Náčrt systému

4.2 Analýza

Výsledný systém by měl být schopný instalování a pozdějšího spouštění uživatelských aplikací. Tento proces by měl systém zvládnout za plného provozu uzlu a to tak aby nepřerušil již běžící aplikace.

Aplikace by pak měly být schopny odesílat a přijmat data jak z aplikací bežících na stejném uzlu tak z aplikací jinde v síti. Směrování samotných zpráv by nemělo být řešené na úrovni aplikací, ale systém by měl poskytovat jednoduché rozhraní pro komunikaci a sám by měl podle směrovací tabulky data přeposílat. Směrovací tabulka bude uložena ve Flash paměti a bude pro každý uzel unikátní. Do této tabulky by mělo jít přidávat a odebírat záznamy opět za běhu bez přerušení aktuální činnosti. Samotné zprávy mezi uzly budou předávány pomocí protokolu MQTT ve formátu, který bude popsán v kapitole 4.5 řešící samotnou formu komunikace.

Kromě rozhraní pro zasílání zpráv, by měl systém aplikacím poskytnout rozhraní pro ukládání a načítání parametrů z Flash paměti které tak budou chráněny proti náhodnému restartu modulu. Tyto parametry společně se systémovými by mělo jít upravovat i zvenku. Pro tyto účely by tak měl systém obsahovat webové rozhraní pro nastavení uzlu s možností editace uložených parametrů. Tento nastavovací režim bude automaticky nabíhat při nastavování nového uzlu, případně jej pujde vyvolat systémovým příkazem obdrženým pomocí MQTT.

4.3 Použítý hardware a software

Za cílovou platformu, pro kterou bude daný systém vytvářen, byly vybrány čipy od firmy Espressif konkrétně ESP8266 a ESP32. Hlavním důvodem pro vybrání byla přítomnost wifi modulu již v základu a také malá pořizovací cena, která se pohybuje okolo 100 korun. ESP8266 bylo vybráno společně s výkonějším modulem hlavně proto, že pro ESP32 je zatím menší podpora v programovacích jazycích.

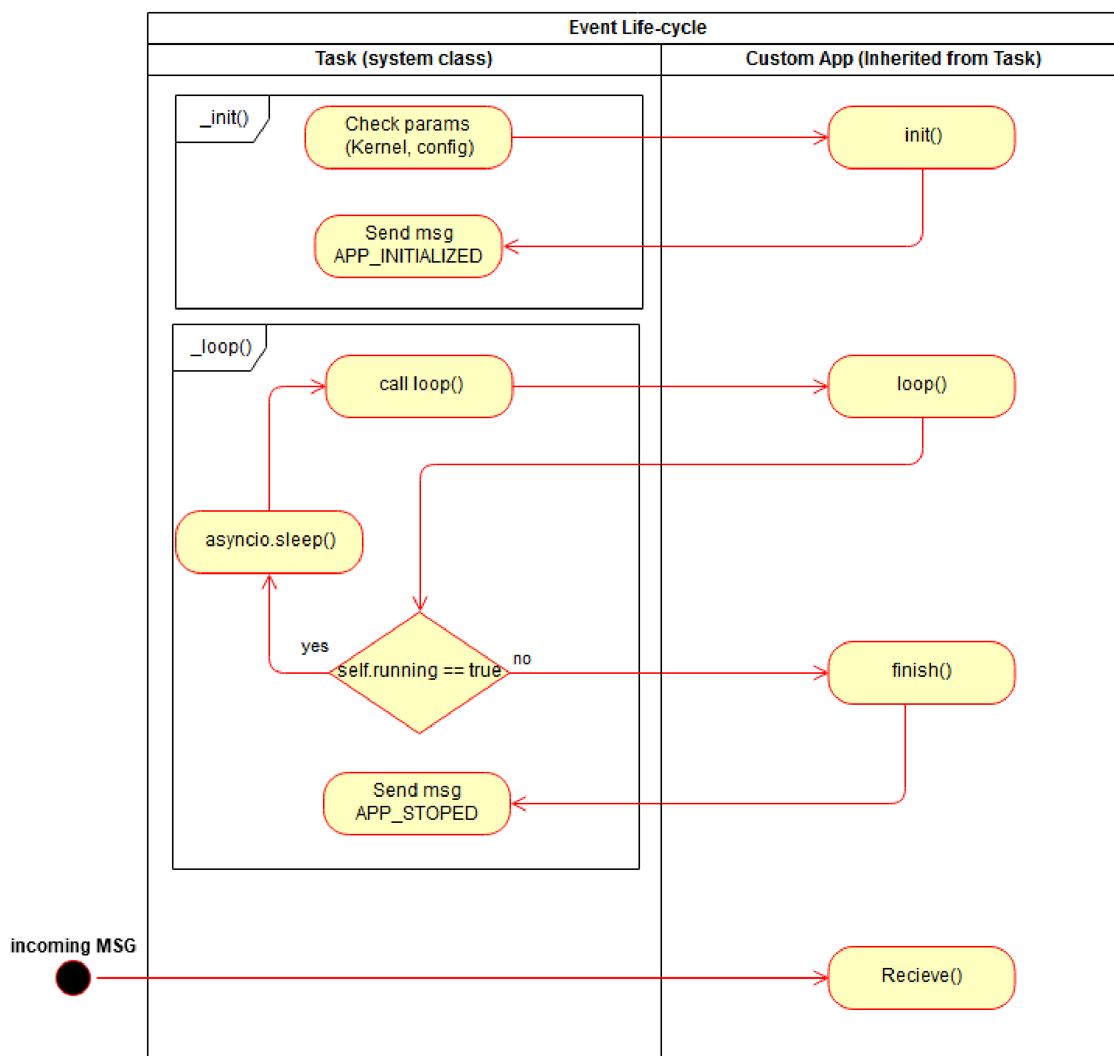
Při výběru jazyka, který bude využit pro samotnou implementaci byl výběr užší. Dobrou podporu desek mají pouze staticky kompilovaný jazyk C nahrávaný pomocí Arduino IDE a Micropython. Bohužel při použití jazyka C by bylo velmi složité dosáhnout modifikovatelného uzlu za běhu. Bylo by totiž nutno vytvořit interpret jiného jazyka. To právě řeší Micropython, který je již sám o sobě interpretem. Navíc poskytuje až na drobné výjimky stejné rozhraní pro programování obou vybraných modulů.

4.4 Uživatelské aplikace

Struktura uživatelských aplikací bude vycházet z již osvědčeného modelu, který je použit pro programování aplikací na deskách Arduino. Tedy samotná aplikace bude rozdělena na několik částí, kdy každá bude mít na starost určitou část životního cyklu aplikace. Popis jednotlivých částí je uveden v následujícím seznam a přibližné pořadí volání jednotlivých částí je zobrazeno na obrázku 4.2, kde je také zobrazená systémová část aplikace, ke které uživatel nebude mít přístup ale je nutná pro správný běh operačního systému.

Části uživatelské aplikace

- **Inicializační část** (`init()`) - Tato část bude spuštěna pouze jednou a to ve chvíli spouštění aplikace na uzlu. V této části by měl uživatel nastavit všechny potřebné parametry a nastavit porty pro přijímání zpráv.
- **Programová část** (`loop()`) - Tato část představuje samotnou programovou smyčku, kterou bude operační systém stále spouštět dokud nebude aplikace označena jako zastavená.
- **Ukončovací část** (`finish()`) - Bude spuštěna těsně před ukončením samotné aplikace.
- **Část pro zpracování zpráv** (`receive()`) - Funkce kterou bude operační systém volat v případě zprávy určené pro danou aplikaci.



Obrázek 4.2: Životní cyklus aplikace společně se systémovou částí.

4.5 Komunikační protokal

Celý komunikační protokol bude vycházet protokolu využitého v rámci systému MPOS. Samotný protokol pak může fungovat dvěma způsoby.

A to zasíláním ve formátu použitém v MPOS, kdy se zprávy zasílají v rámci jednoho MQTT topicu. Rozlišení a směrování samotných zpráv je uskutečněno až podle jejich obsahu na jednotlivých uzlech. Zpráva je rozdělena na tři hlavní části. První pouze určuje typ zprávy, zda se jedná o unicast (".") nebo broadcast ("*"). MQTT totiž všechny zprávy odesílá jako boadcast všem členům, kteří jsou přihlášení k danému topicu, ale pro potřeby operačního systému je potřeba unicast a broadcast rozlišit. Druhá část zprávy může mít dva významy a to podle typu zprávy. V případě unicastové zprávy je zde obsažena adresa cílo-

vého uzlu, jméno aplikace pro kterou je zpráva určena a název vstupního portu. Jednotlivé atributy musí být uvedeny přesně v tomto pořadí. V případě broadcastu jsou ve druhé části informace o odesilateli. Poslední částí zprávy jsou samotná posílaná data. Tvar takovéto zprávy, kdy mají být data předána aplikaci `light_control` na port `lamp`, je zobrazen na obrázku 4.3.

```
{"*", {"192.168.0.8", "light_control", "lamp"}, "ON"}
```

Obrázek 4.3: Tvar zpráv při využívání komunikace po jednom topicu.

Druhým typem je směrování zpráv pomocí více topiců, kdy každý uzel je přihlášen ke svém vlastnímu topicu. Směrování tedy probíhá již na úrovni MQTT a to tak, že ke kořenovému názvu topicu přidáme adresu uzlu ve tvaru IP adresy nebo jeho název, dále pak název aplikace a portu. Samotná zpráva je pak rozdělena na dvě části. Hlavičku a tělo. V hlavičce je na prvním místě název odesilatele zprávy. Pokud se jedná o systémovou zprávu tak na druhém místě bude hlavička obsahovat příkaz který má být proveden. V těle zprávy pak budou samotná odesílaná data. V případě kořenového topicu `home`, pak bude výše zmíněný příklad vypadat tak jak je možno vidět na obrázku 4.4. [19]

data: `{{"node_1234"}, {"ON"}}`
topic: `home/192.168.0.8/light_control/lamp`

Obrázek 4.4: Tvar zpráv při využití více topiců

Výsledný systém by měl podporovat oba typy komunikace, kdy u každého uzlu bude třeba vybrat jakým způsobem se bude komunikovat. V jedné síti by se poté mělo dodržet pravidlo že se bude komunikovat pouze jedním způsobem.

Při zasílání systémových, řídicích, zpráv se neurčuje vstupní port. Při využití směrování pomocí více topiců zůstane tvar zprávy stejný pouze se se sníží úroveň topicu o jednu úroveň. Avšak při zasílání zpráv pomocí prvního formátu, se změní druhá část zprávy, obsahující směrovací informace, a to tak že se odebere poslední část určující port, místo jména aplikace bude některé z klíčových slov a datová část zprávy může být prázdná. Příklad takové zprávy je na obrázku 4.5

```
{"*", {"192.168.0.8", "restart"}, ""}
```

Obrázek 4.5: Tvar systémové zprávy.

Seznam systémových zpráv

Systém obsahuje základní sadu řídicích zpráv určených pro ovládání uzlu. Celý seznam systémových zpráv je sepsán do tabulky 4.1. V případě zprávy `status` uzel odpovídá zprávou která ve své datové části obsahuje informace o názvu uzlu, jeho IP, platformě, času jak dlouho již běží, využité paměti, nainstalovaných a běžících aplikacích. Příklad takové zprávy je zobrazen na obrázku 4.6. Na zprávu `discover` odpoví systém zprávou obsahující jméno a IP adresu uzlu.

```
{{"node_name", "chodba"}", {"IP", "192.168.0.8"}, {"platform", "esp32"},
{"running_time", "123"}, {"memory", "123456"}, {"installed", {"display",
"temp"}}, {"running", {"display1", "temp1", "temp2"}}}
```

Obrázek 4.6: Tvar odpovědi na zprávu status nebo discover

Příkaz	**Obsah datové části**	**Popis**
`restart`	prázdný	Restartuje cílový uzel
`factory`	prázdný	Vrátí uzel do počátečního stavu
`status`	prázdný	Uzel jako odpověď na tuto zprávu zašle informace o svém stavu
`runningtime`	prázdný	Uzel odpovídá dobou běhu
`discover`	prázdný	Odesíláno broadcastově. Po obdržení se každý uzel ohlásí zprávou `NODE_ALIVE`.
`wifiapmode`	prázdný	Uzel se přepne do AP módu pro jeho konfiguraci
`settingsweb`	prázdný	Uzel ukončí všechny aplikace a přepne se do módu konfigurace.
`addroute`	`{{"srcApp", "srcPort"}, {{"DEST_NODE", "destApp", "destPort"}, ...}}`	Přidání záznamu do směrovací tabulky.
`removeroute`	stejný jako `addroute`	Odebere záznamy ze směrovací tabulky
`load`	`{"zdrojový kód aplikace"}`	Nainstaluje novou aplikaci do modulu
`unload`	`{"název zdrojového kódu"}`	Odinstaluje aplikaci z uzlu
`activate`	`{"název z. kódu", "název", "volitelně další parametry"}`	Spustí novou aplikaci se zvoleným názvem ze zadaného zdrojového kódu
`deactivate`	`{"název aplikace"}`	Ukončí běžící aplikaci

Tabulka 4.1: Seznam systémových zpráv

Kapitola 5

Implementace

Vývoj celého systému probíhal většinu času na platformě ESP32 s občasným testováním systému na starší verzi ESP8266. Tento přístup byl zvolen, protože i přes snahu udržet kód co nejmenší, zabírá systém na starším ESP většinu RAM paměti a hledání optimalizované varianty po přidání nové funkcionality, bylo časově náročné. Finální verze systému, je ale optimalizována aby běžela na obou platformách.

Z důvodu nedostatku RAM paměti u starší desky bylo taky upuštěno od původního záměru implementovat dvě varianty komunikačního protokolu. První variantou byla komunikace pomocí jednoho topicu a směrování na úrovni zpráv a druhou využití více topiců a ke směrování použít jejich zanořování. Ve výsledném systému je tedy implementována pouze druhá varianta, která má oproti první výhodu v tom, že jednotlivé uzly nejsou zbytečně zahlcovány zprávami, které nejsou určeny pro daný uzel.

5.1 Knihovny jazyka Micropython

Jazyk Micropython v základu neobsahuje veškerou funkcionalitu, která byla potřeba při vývoji a bylo jej nutné rozšířit o některé knihovny. Ale jelikož je jazyk Micropython pro platformu ESP stále v rané fázi vývoje a ve verzi systému pro ESP32 často chybí i funkcionalita, která je pro starší variantu ESP8266 již přítomna, mění se tak často i základní seznam knihoven, které jsou v systému obsaženy. Je tedy možné že v dalších verzích systému již nebudou mnou přidávané knihovny zapotřebí.

Nejdůležitější knihovnou kterou systém využívá, je knihovna **uAsyncio**[1]. Ta umožňuje souběžný běh více úloh najednou pomocí syntaxe async/await a řeší tedy problém kooperativního multitaskingu. Jedná se o částečný klon knihovny Asyncio, která je obsažena v jazyce Python, upravený pro běh v jazyce Micropython.

Další přidanou knihovnou je **uMqtt**[2] implementující klienta pro komunikaci s MQTT serverem. Pro vývoj byla použita verze knihovny `umqtt.robust`, která řeší automatické připojení k serveru, pokud nastane chyba nebo na okamžik vypadne spojení. Pro ESP32 je tato knihovna již v základním systému ale pro verzi ESP8266 je potřeba ji přidat. Knihovna bohužel postrádá funkci `unsubscribe` pro zrušení odběru zpráv z daného topicu.

Poslední je knihovna pro `HD44780` kompatibilní LCD displeje. Nejznámějším zástupcem této kategorie je 16 místný, dvou řádkový displej `1602A`. Repozitář této knihovny[3] obsahuje

[1] https://github.com/micropython/micropython-lib/tree/master/uasyncio
[2] https://github.com/micropython/micropython-lib/tree/master/umqtt.robust
[3] https://github.com/dhylands/python_lcd

implementaci komunikace s displejem pro více platforem. Do firmwaru Micropythonu byly přibaleny pouze soubory `lcd_api.py`, `esp8266_i2c_lcd.py` a `esp32_gpio_lcd.py`, které pokrývají nejčastější připojení displeje. Příklad aplikace využívající tuto knihovnu je na obrázku 5.6

V případě že je potřeba do jazyka Micropython přidat knihovny, jsou dva způsoby jak toho docílit. Nejjednodušším způsobem je nahrát knihovnu přímo do Flash paměti pomocí některého z nástrojů, umožňujících komunikaci s Micropythonem. Jedním z nich je například `mpfshell`[4], který byl použit při vývoji pro nahrávání zdrojových souborů vyvíjeného systému. Druhou možností je přibalit knihovny přímo do firmwaru. Podle dokumentace jsou pak takové knihovny zkomprimovány a měly by zabírat méně místa a být úspornější v rámci RAM paměti. V případě že chceme přibalit knihovny k firmwaru je potřeba nahrát jejich zdrojové soubory do složky `modules`, pro každou platformu zvlášť. Například tedy pro ESP32 bude umístění následující: `micropython/ports/esp32/modules`.

5.2 Struktura systému

Při vývoji systému bylo dbáno na co největší uzavřenost jednotlivých částí systému. Tím pádem byl systém rozdělen do pěti objektů, kdy každý má na starost pouze určitou část systému. Jednotlivé objekty si předávají informace pomocí callbacků. Za zmínku stojí soubor `main.py`, který je automaticky volán po startu desky a je tedy vhodným kandidátem pro umístění spouštěcího skriptu celého systému. Navíc jsou zde odchytávány všechny výjimky, které nebyly zpracovány systémem. V případě že taková situace nastane, pokusí se uzel restartovat a pokud je to možné tak před samotným restartem zaslat pomocí MQTT zprávu o tom že bude uzel restartován.

Modul Kernel

Modul Kernel je první, který je při startu uzlu spuštěn a jedná se také o modul obsahující většinu řídicí logiky uzlu. Modul po spuštění postupně vytváří a spouští zbylé části systému.

V inicializační části, kterou představuje funkce `init()` je vytvořen modul **Config** a zkontroluje se zda je uzel správně nakonfigurován. Pokud konfigurace není kompletní, je pomocí modulu **Webserver** vytvořen konfigurační web pro nastavení. Zde se ještě kontroluje zda už je uzel připojený na wifi síť, nebo je potřeba pro konfiguraci vytvořit vlastní přístupový bod. Pokud proběhla konfigurace v pořádku, je vytvořen klient starající s o komunikaci pomocí MQTT protokolu a je požadováno připojení na server. V případě že i tato část inicializace proběhne v pořádku, zašle se zpráva `NEW_NODE` na systémový topic.

Odesláním zprávy je považována inicializační část za kompletní a přichází na řadu funkce `boot()`, starající se o spuštění samotného systému. Ta se skládá pouze z vytvoření hlavní běhové smyčky knihovny **uAsyncio** a vytvoření tzv. podúlohy, čímž jsou v knihovně označovány procesy, které bude knihovna spouštět. Zde je podúlohou myšlena funkce `run()` z modulu Kernel, která obsahuje nekonečnou smyčku `while` ve které je volána funkce na kontrolu příchozích zpráv a funkce z knihovny **uAsyncio** pro předání řízení běhu.

Pokud jsou přítomny nějaké příchozí zprávy, jsou nejdříve podle topicu ze kterého přišly, rozděleny na zprávy pro uživatelské aplikace a na systémové zprávy. Zprávy pro aplikace jsou ihned předány a čeká se na jejich zpracování. V případě systémových zpráv se zkontroluje jaký příkaz byl zaslán a na jeho základě je zavolána funkce starající se o jeho obsloužení.

[4]https://github.com/wendlers/mpfshell

Pokud příkaz nevyžaduje odeslání specifické odpovědi, je odeslána zpráva obsahující výsledek zda se povedlo korektně provést příkaz. Odešle se tedy zpráva, která má v hlavičce `OK` nebo `ERROR`. Příklad takové zprávy je na obrázku 5.1

```
{{"ESP_kuchyne", "OK"}, ""}
```

Obrázek 5.1: Tvar systémové zprávy potvrzující úspěšné provedení příkazu

Většina systémových příkazů pouze vrátí předem sestavenou zprávu doplněnou o aktuální data, nebo volá funkce z ostatních modulů jejichž funkčnost bude teprve popsána. Avšak důležitým příkazem prováděným na úrovni tohoto modulu je příkaz `ACTIVATE`, který má za úkol spustit uživatelskou aplikaci. Ve skutečnosti je zde pouze vytvořen objekt **UserTask**, který zapouzdřuje všechny uživatelské aplikace aby měly z pohledu systému stejné rozhraní. Nad tímto objektem je pak vyžádána inicializace aplikace a pokud proběhne v pořádku, je funkce `loop` objektu UserTask předána knihovně uAsyncio jako další podúloha, čímž je zajištěno že bude aplikace pravidelně spouštěna.

Ještě je potřeba zmínit zpracování příkazu `LOAD`, který je používán pro nahrání zdrojových kódů uživatelských aplikací na uzel. Zde se totiž nachází jeden z velkých rozdílů oproti původnímu systému, který byl zmiňován v úvodu této práce. Ten si držel nahrané uživatelské aplikace pouze v proměnné a při restartu uzlu tak bylo potřeba všechny aplikace nahrát znova. Systém vyvíjený v rámci této práce, si po zpracování zprávy uloží zdrojový kód uživatelské aplikace jako soubor do Flash paměti do složky `apps`. Tím je zaručeno zachování uživatelských aplikací po restartu uzlu a navíc aplikace, které aktuálně nejsou používány nezabírají místo v RAM paměti.

Modul Kernel poté ještě obsahuje drobné funkce starající se např. o synchronizaci času nebo výpočet volného místa ve Flash paměti. Dále je zde ještě několik funkcí, které jsou předávány jako callbacky ostatním modulům čímž je propojuje.

Modul Config

Hlavním úkolem tohoto modulu je práce s konfiguračním souborem uloženým ve Flash paměti uzlu. Aby byla práce se souborem co nejjednodušší byl jako formát dat zvolen JSON. Velkou výhodou toho formátu je navíc i jednoduchá čitelnost. Tím pádem lze před připravit konfigurační soubor v počítači a na uzel jej nahrát např. pomocí nástroje **mpfshell**, díky čemuž odpadne nutnost konfigurovat uzel pomocí webového rozhraní.

Jako první věc při spuštění tohoto modulu se kontroluje zda existuje konfigurační soubor `config.json`. Pokud ne, tak je vytvořen soubor se základními hodnotami. Ukázku konfiguračního souboru lze vidět na obrázku 5.2. Dále následuje stručný popis jednotlivých konfiguračních parametrů.

- **name** - jméno uzlu v síti, které je použito pro směrování v rámci MQTT topiců. Dále při použití AP módu uzlu bude mít vytvořená wifi síť právě tento název.
- **pass** - určuje heslo, které bude potřeba pro připojení k wifi síti, která je vytvořena při AP módu.
- **port** - port na kterém bude dostupná webová konfigurace uzlu
- **topic** - určuje kořenový topic pro MQTT kounikaci.
- **aps** - seznam wifi bodů ke kterým se uzel bude umět připojit.
- **ssid** - název wifi bodu pro připojení
- **pass** (v objektu ap) - heslo pro připojení k wifi bodu
- **mqtt** - IP adresa MQTT serveru pro připojení
- **port** (v objektu ap) - port MQTT serveru
- **user** (volitelná položka) - jméno pro připojení k MQTT serveru
- **mqttpass** (volitelná položka) - heslo k MQTT serveru
- **ifconfig** (volitelná položka) - obsah tohoto parametru se použije pro nastavení statické IP adresu uzlu. Jednotlivé hodnoty v tomto poli musí být přesně v tom pořadí v jakém jsou uvedeny v ukázkovém příkladu a reprezentují: IP adresa uzlu, maska sítě, IP adresa brány, IP adresa DNS serveru
- **cp** - do tohoto objektu se ukládají všechny uživatelské parametry, které mají být perzistentní

Do konfiguračního souboru lze ukládat kromě systémových parametrů také uživatelská data a to pomocí funkce `set(key, val)`, které lze následně získat funkcí `get(key)`. Tyto parametry jsou pak dostupné i po restartu uzlu. Parametry jsou navíc v rámci souboru umístěny v objektu `cp`, tedy nemůže dojít k přepsání systémových parametrů pokud by uživatel zadal název parametru, který by se shodoval s názvem některého ze systémových parametrů.

Kromě práce s konfiguračním souborem se modul Config stará o připojení k wifi síti. Ve chvíli kdy již načetl konfigurační soubor, zkontroluje všechny aktuálně viditelné wifi sítě a porovná jejich názvy s těmi, které načetl z konfiguračního souboru. Poté se postupně snaží připojit k těm sítím u kterých se shoduje název s některým z názvů načtených ze souboru. U ESP8266 ještě před samotným připojováním zkontroluje zda již není uzel připojený k wifi síti, jelikož firmware pro ESP8266 disponuje tím, že při startu desky se připojí k poslední známe wifi síti, aniž by to programátor explicitně vyžádal. Pokud se uzlu nepodaří připojit k žádné síti, vrací informaci že se nepovedla konfigurace a modul Kernel poté zapíná AP mód pro webovou konfiguraci.

Dále jsou v modulu přítomny funkce pro získání aktuální IP adressy `get_actual_ip()` a získání celkové konfigurace síťového adaptéru `if_config()`.

```
{
  "name": "node_4862agf",
  "pass": "",
  "port": "80",
  "topic": "mpos",
  "aps": [
    {
      "ssid": "Wifi_AP_name",
      "pass": "wifiPassword",
      "mqtt": "192.168.0.6",
      "port": "1883",
      "user": "mpos",
      "mqttpass": "mpos",
      "ifconfig": [
        "192.168.0.165",
        "255.0.0.0",
        "192.168.0.1",
        "192.168.0.1"
      ]
    }
  ],
  "cp": {
    "user_data": "value"
  }
}
```

Obrázek 5.2: Ukázka konfiguračního souboru

Modul Communication

Cílem tohoto modulu je poskytnout jednoduché ucelené rozhraní pro přijímání a odesílání zpráv. Dále se modul stará o formátování odchozích zpráv aby vyhovovaly výše zmíněnému protokolu.

Při inicializaci tohoto modulu je třeba mu předat informace o MQTT serveru ke kterému se má připojit. Těmi jsou IP adresa a port na kterém server naslouchá. Navíc pokud byly v konfiguračním souboru uvedeny, tak uživatelské jméno a heslo pro autentifikaci. Připojení poté probíhá pomocí klienta z knihovny **umqtt.robust**. Před samotným připojením je nejdříve knihovně třeba předat callback funkci, která se bude volat v případě že uzel obdrží příchozí zprávu. Tou je funkce `__decode_message(topic, raw_msg)`. Ta při přijetí nové zprávy nejdříve zjistí kdo je odesilatelem zprávy a pokud se jméno odesilatele shoduje se jménem aktuálního uzlu, pak je zpráva ignorována. To že uzel dostane svou vlastní zprávu se stane ve chvíli, kdy odešle zprávu na topic, ke kterému je zároveň i přihlášený pro odběr. Pokud se jedná o cizí zprávu je z hlavičky zprávy odstraněn odesilatel jelikož dále již není potřeba. Dále je z topicu vytažen název aplikace a portu na kterou má být zpráva doručena. Tyto informace společně s daty obsaženými v těle zprávy jsou pomocí callback funkce, kterou modul dostal jako parametr při inicializaci, předány modulu Kernel, který se postará o předání dat aplikaci případně o provedení systémového příkazu.

MQTT klient není spuštěn asynchronně, proto je třeba pro správné přijímání zpráv co nejčastěji volat funkci `wait_msg()`. Toho je docíleno voláním funkce v modulu Kernel v hlavní systémové smyčce, která byla popsána v kapitole 5.2.

Při odesílání dat z uzlu se data musí zformátovat podle formátu protokolu. K tomu slouží funkce `__ecnode_msg()`, která data převede a navíc doplní hlavičku zprávy o odesilatele a případně klíčovým slovem pokud je poskytnuto.

Pokud odeslání zprávy vyžádá aplikace je potřeba vědět na který topic zprávu odeslat. K tomu účelu slouží směrovací tabulka, která je uložena v souboru `routing.json`. Jak lze vidět na obrázku 5.3, data jsou opět ve formátu JSON. V této tabulce je vždy pro dvojici název aplikace a port na který aplikace odesílá zprávu, seznam cílů na který se má daná zpráva odeslat. V případě že se cílový uzel shoduje se jménem nebo IP adresou aktuálního uzlu, tak se zprávy neodesílají pomocí MQTT ale předají se přímo Kernel callbacku k doručení. Jako cílový uzel jde také uvést klíčové slovo `self`. Zpráva pak bude doručena vrámci aktuálního uzlu.

Uživatelská aplikace tedy nemusí explicitně určovat cíl odesílané zprávy. Pouze zavolá systémovou funkci, které předá v parametru kromě zprávy port na který má být zpráva předána a systém ji pomocí výše zmíněné tabulky rozešle konkrétním příjemcům. V případě přijímaní zprávy, aplikace na začátku svého běhu nahlásí systému na kterých portech naslouchá, tedy ze kterých portů chce přijímat, a ten jí následně bude zprávy z těchto portů předávat.

Úpravy směrovací tabulky se provádí pomocí systémových zpráv `ADDROUTE` a `REMOVEROUTE`. Případně lze směrovací tabulku nahrát z počítače. Díky uložení tabulky ve Flash paměti zůstane směrovací tabulka i po restartu uzlu a není ji tak potřeba nahrávat znova.

V případě že je zapotřebí odeslat MQTT zprávu na zvláštní topic, nebo ve formátu jiném než určuje komunikační protokol, disponuje modul metodou `send_raw_msg(msg, topic)`, která ignoruje směrovací tabulku a formátovací metody. Místo toho na zadaný topic odešle zprávu přesně v tom tvaru v jakém ji obdrží od aplikace.

```
{
  "src_app_name": {
    "src_app_port": [
      {
        "t": "target_node",
        "p": "target_port",
        "a": "target_app"
      }
    ]
  }
}
```

Obrázek 5.3: Ukázka směrovací tabulky

Modul Webserver

O zobrazení webové konfigurace, kterou systém disponuje se stará modul **Webserver**. Toto konfigurační prostředí může být nastartováno ve dvou módech, které se liší stavem připojení na wifi síť. Tedy pokud uzel je připojen k wifi síti a web je pak dostupný na adrese uzlu a nebo je třeba vytvořit wifi hotspot a spustit tak AP mód kdy je webové rozhraní dostupné na adrese **192.168.4.1**. Wifi síť se bude jmenovat stejně jako je název daného uzlu. Pokud je v konfiguračním souboru vyplněný parametr `pass` bude potřeba při připojení k síti použít hodnotu tohoto parametru jako heslo. Pokud je parametr prázdný bude síť nezabezpečená.

Tento nastavovací režim můžou spustit dvě různé události. První z nich je přijetí systémové zprávy obsahující jeden z příkazů `SETTINGSWEB` nebo `APMODE`. Podle toho který byl přijat se spustí mód konfigurace. Druhou událostí, která může spustit nastavovací režim je nepřipojení se k wifi síti nebo MQTT serveru. V takovém případě je vždy spuštěna verze s wifi hotspotem. Pokud s webovým serverem nebude po dobu 5 minut komunikováno bude uzel restartován a pokusí se podle aktuálního konfiguračního souboru spustit systém.

Samotný webserver je implementován pomocí BSD soketů. Na portu uvedeném v konfiguračním souboru, je vytvořen socket čekající na příchozí spojení. Po přijetí, je klientovi poslán obsah webové stránky a spojení je ukončeno. Zároveň je obsah každého příchozího požadavku prohledán zda se nejedná o odeslání konfuguračního formuláře. Pokud je požadavek vyhodnocen jako odeslání formuláře, jsou do konfiguračního souboru znovu uložena všechna data z požadavku.

Obrázek 5.4: Rozhraní webové konfigurace

Modul UserTask

Tento modul figuruje jako mezi vrstva mezi systémem a samotnými uživatelskými aplikacemi. Navíc poskytuje neměnné rozhraní, a systém tedy nemusí řešit zda je uživatelská aplikace má implementovány všechny funkce.

Největším problémem, který bylo potřeba vyřešit je dynamické načítání uživatelských aplikací ze souboru. Použití základního příkazu `import` pro nahrání modulu nebylo možné využít, protože tento příkaz nepodporuje určení nahrávaného modulu pomocí proměnné. Python a Micropython naštěstí pro tyto případy disponují funkcí `__import__()`, která má podobné chování jako výše zmíněný `import`. Díky této funkci v kombinaci s funkcí `getattr()` je možné dynamicky načítat uživatelské aplikace podle jména uloženého v proměnné. Celý příkaz i s instancováním této třídy je zobrazen na obrázku 5.5. Tím pádem do paměti RAM jsou načítány pouze ty aplikace, které budou spuštěny.

UserTask si tedy drží ve své třídní proměnné instanci uživatelské aplikace, nad kterou volá požadované metody. Při instancování aplikaci předává callback funkce pro nejdůležitější akce a předává aplikaci parametry, které byli poslány v rámci systémové zprávy `ACTIVATE`. Celkový přehled toho co má uživatelská aplikace k dispozici bude uveden později u popisu samotných aplikací.

Funkce `run_task()`, která obsahuje hlavní běhovou smyčku, ve které je volána funkce `loop()` uživatelské aplikace, zároveň obsahuje i volání `uasyncio.sleep(0)` pro odevzdání řízení. V případě že by uživatel při vytváření aplikace zapomněl ve své aplikaci volat tento příkaz, je tím zaručeno že vždy před každým novým během aplikace je předáno řízení zpátky knihovně `uAsyncio`. Tím je zaručen souběžný běh více aplikací. Bohužel ani tato konstrukce nedokáže zabránit situaci, kdy se aplikace chybou uživatele zacyklí bez volání `uasyncio.sleep()`. V takovém případě nedojde nikdy k předání řízení a díky tomu nebudou kontrolovány příchozí zprávy a celý uzel tak přestane reagovat na příkazy z venčí. Jedinou možností je pak restart pomocí tlačítka umístěného přímo na desce uzlu.

```
1 imported_module = __import__('apps.' + class_name, globals(), locals(), [
     class_name])
2 task_class = getattr(imported_module, class_name)
3 user_app = task_class(app_params)
```

Obrázek 5.5: Příkazy pro dynamické načtení třídy s uživatelskou aplikací

5.3 Uživatelské aplikace

Za účelem poskytnutí uživateli co největší volnosti při tvorbě aplikace, ale zároveň za udržením co největší konzistence jednotlivých aplikací, jsou za uživatelské aplikace považovány ty třídy, které dědí od třídy `Task`, která se nachází v modulu UserTask. Takováto aplikace pak obsahuje předepsané rozhraní pro volaní a navíc díky dědičnosti má přístupné rozličné systémové metody, které může použít.

Při návrhu rozhraní aplikací se vycházelo z populárního formátu, který se používá pro tvorbu aplikací na platformě Arduino. Aplikace se tedy skládá z inicializační části vesměs reprezentovanou funkcí `init()` a hlavní běhovou částí, stávající se z funkce `loop()`, která je interním systémem volána v nekonečné smyčce. Pro aplikace spustitelné v rámci vyvíjeného systému bylo ponecháno stejné pojmenování a funkčnost. Výčet funkcí s popisem jich významu a doby volání bude popsán dále. Třída `Task` od které musí dědit všechny aplikace, již obsahuje základní definici všech funkcí. Díky tomu není uživatel nucen implementovat všechny funkce ale přepsat pouze ty funkce, které reálně využije.

Pro lepší představu jak psát uživatelské aplikace, je u zdrojových kódů systému, také soubor zdrojových kódů aplikací. Velká část kódů byla převzata z původní verze systému a pouze byla upravena do formátu používaného v aktuální verzi systému. Ukázkové aplikace byli vybírány a vytvářeny tak, aby pokryly nejčastější případy použití a také využití knihoven. Příklad uživatelské aplikace využívající externí knihovnu je na obrázku 5.6. Ve zdrojovém kódu každé aplikace je uveden komentář jaké parametry a v jakém pořadí, je aplikace očekává při její aktivaci.

Struktura uživatelské aplikace

Při tvorbě aplikace může uživatel přepsat čtyři systémové funkce, které jsou volány v různou dobu běhu aplikace. Všechny až na výjimku u `loop()` funkce jsou synchronní. V případě že uživatel nepřepíše některou funkci, je použita verze ze třídy `Task`, kde v těle funkce je pouze příkaz `pass`.

- **`def init()`** - Tato funkce je volána při inicializaci aplikace a měla by sloužit pro nastavení všech potřebných parametrů, případně spuštění potřebných periferií nutných pro běh aplikace. Zároveň se jedná o jediné místo kde se může aplikace přihlásit k odběru zpráv z daného portu.

- **`async def loop()`** - Tělo této funkce bude spouštěno smyčkou stále dokola. Uživatel se o nekonečný běh nemusí starat. Ten zajistí modul `UserTask`, který se stará o spouštění uživatelských aplikací.

- **`def finish()`** - V případě že má být aplikace ukončena, bude tato funkce zavolána po posledním spuštění funkce `loop()`

- **`def receive(msg, port)`** - Pokud má aplikace implementovánu tuto metodu a ve funkci `init()` se přihlásila k odběru zpráv, bude tato funkce zavolána v případě nové příchozí zprávy. Parametry které dostane jsou, tělo zprávy a port na který byla zpráva odeslána.

Třída `Task` obsahuje ještě dvě metody, které lze teoreticky v aplikaci předefinovat. Funkce `is_running()`, která vrací booleovskou hodnotu reprezentující zda aplikace stále běží a funkce `stop()` která je používána pro zastavení běhu aplikace. Avšak přepsání těchto funkcí není doporučeno. Jejich změna může vést k nefunkčnosti uživatelské aplikace.

Dostupné prostředky v aplikaci

Aplikace dále disponují sadou proměnných a funkcí, které rozšiřují její schopnosti. Následující seznam obsahuje jejich kompletní seznam se stručným popisem.

- **`self._name`** - Proměnná obsahuje název pod kterým je spuštěna aktuální aplikace
- **`self._params`** - Pole obsahující parametry, které byli aplikaci zaslány pomocí systémové zprávy `ACTIVATE`
- **`self.is_running()`** - Funkce vrací hodnotu True pokud je aplikace aktivní
- **`self.stop()`** - Po zavolání této funkce bude aplikace po dalším běhu funkce `loop()` zastavena.
- **`self.subscribe(port)`** - Aplikace se pomocí této funkce může přihlásit k odebírání zpráv na zadaném portu. Metodu lze volat pouze při inicializaci aplikace. (Funkce `init()`)
- **`self.send_msg(msg, port)`** - Tato funkce provede odeslání zprávy na zadaný port
- **`self.send_raw_msg(msg, topic)`** - Funkce pro odeslání zprávy na konkrétní topic. Při odesílání nebude zpráva upravena do formátu komunikačního protokolu a bude ignorována směrovací tabulka. Metoda je tak vhodná pro odesílání zpráv do externích systémů.
- **`self._config.get(key)`** - Získání hodnoty z konfiguračního souboru uložené pod daným klíčem. Lze vyžádat jak uživatelská data tak i systémové parametry.
- **`self._config.set(key, val)`** - Uložení hodnoty do konfiguračního souboru. Nelze přepsat systémové parametry.
- **`self._config.del_param(key)`** - Odstranění hodnoty z konfiguračního souboru. Nelze smazat systémové parametry.
- **`self._config.get_params(include_custom=True)`** - Vrátí asociativní pole obsahující parametry z konfiguračního souboru. Změnou parametru `include_custom` lze ovlivnit zda budou navráceny pouze systémové parametry nebo všechny včetně uživatelských parametrů.
- **`self._config.get_actual_ip()`** - Vrátí aktuální IP adresu uzlu. V případě chyby při jejím získávání vrací hodnotu False.
- **`self._config.ifconfig()`** - Funkce pro získání kompletních informací o připojení k síti. Její návratovou hodnotou je pole obsahující položky v tomto pořadí: IP adresa uzlu, maska sítě, IP adresa brány, IP adresa DNS serveru

```
from UserTask import Task
from machine import Pin
from esp32_gpio_lcd import GpioLcd

# Params - [TOPIC_TO_LISTEN]
class Lcd(Task):

    def init(self):
        self.lcd = GpioLcd(rs_pin=Pin(4),
                           enable_pin=Pin(14),
                           d4_pin=Pin(5),
                           d5_pin=Pin(18),
                           d6_pin=Pin(21),
                           d7_pin=Pin(22),
                           num_lines=2, num_columns=16)
        self.subscribe(self._params[0])

    def receive(self, msg, port):
        if port == self._params[0]:
            self.lcd.clear()
            self.lcd.putstr("Temp: "+ str(msg[0]) + "\nHum: " + str(msg[1]))

    def finish(self):
        self.lcd.clear()
```

Obrázek 5.6: Ukázková aplikace pro výpis na display 1602A

5.4 Ukládání hodnot a Domoticz

Z povahy celého systému se dá odvodit že největší nasazení bude v rámci řízení inteligentní domácnosti. V takové situaci ale chceme většinou znát kromě aktuálních dat také data historická. Je tedy potřeba si aktuální naměřené hodnoty nebo stav zařízení ukládat do databáze. Bohužel aktuálně na platformě ESP nelze databáze použít. ESP8266 má malý výkon a hlavně příliš malou paměť RAM pro samotné spuštění databáze. U ESP32 by již databáze spustit šla bohužel velikost paměti by opět omezovala běh dalších aplikací. Umístění databázového serveru na uzel by tedy nebylo výhodné.

Jednou z možností jak tyto hodnoty ukládat je odesílat je např. do populární služby Domoticz[5]. Tato služba poskytuje rozsáhlý soubor funkcí pro ovládání chytré domácnosti. Při ukládání hodnot využijeme zobrazení na dashboardu, kde je zobrazena poslední přijatá hodnota nebo po otevření detailu konkrétního senzoru graf, zobrazující historická data. Služba navíc poskytuje možnost vytváření různých eventů, které lze například po propojení se službou IFTTT[6] využít pro zasílání notifikací přímo na mobilní telefon. Eventy lze programovat interaktivně pomocí předpřipravených bloků nebo pomocí několika jazyků mezi kterými jsou Python a Lua.

Pro využití služby Domoticz je nutné ji nainstalovat například na Raspberry Pi. Aby bylo možné připojit Domoticz k MQTT serveru, je třeba doinstalovat rozšíření pro MQTT podle návodu uvedeného v dokumentaci[7]. Jakmile bude Domoticz spuštěn, je potřeba v jeho nastavení vytvořit nové hardwarové zařízení typu **MQTT Client Gateway** a v tomto zařízení nastavit připojení k MQTT serveru. Ten musí být stejný jako server, který využívají uzly ze kterých budeme chtít odesílat data. Jako poslední je potřeba vytvořit virtuální senzor pro každou hodnotu kterou budeme chtít ukládat.

Samotné zaslání dat pak lze provést pomocí funkce `send_raw_msg(msg, topic)`, která je dostupná pro každou aplikaci. Jako topic vždy zadáme **domoticz/in**, ale zpráva se bude lišit podle toho jaký virtuální senzor je nastaven v Domoticz. Zpráva bude ale vždy obsahovat hodnotu `idx`, která označuje id virtuálního senzoru ve službě Domoticz. Např. pro odesílání teploty lze využít zprávu, která je zobrazena na obrázku 5.7. V přiložených aplikacích je v aplikaci **Dht**, jednoduchý kód starající se právě o zaslání teploty do služby Domoticz.

Službu Domoticz lze obecně využít i pro samotné ovládání zařízení přímo z jeho webového rozhraní. Bohužel zprávy které služba odesílá nejsou kompatibilní s komunikačním protokolem který využívají uzly a Domoticz navíc nepodporuje změnu formátu odesílaných zpráv. Proto aktuálně lze využít Domoticz pouze pro přijímání a zobrazování dat. Pokud bychom chtěli ovládat systém pomocí této služby, je nutno vytvořit aplikaci, která se bude starat o příjem zpráv ze služby Domoticz a jejich následné upravení do formátu komunikačního protokolu z kapitoly 4.5 a odeslání na správný uzel sítě.

```
{"idx":11, "nvalue":0, "svalue":"25.2"}
```

Obrázek 5.7: Formát zprávy pro odeslání teploty do služby Domoticz

[5] http://www.domoticz.com/
[6] https://ifttt.com/
[7] https://www.domoticz.com/wiki/MQTT#Installing_software

Kapitola 6

Nástroje pro správu systému

Při vývoji systému byly navíc vytvořeny dva nástroje ulehčující práci se systémem. Těmito nástroji jsou **Remote Control** pro jednodušší práci s nastavováním jednotlivých uzlů a micro **Monitoring**, zaznamenávající aktuální stav na síti.

Pro vytvoření zmíněných nástrojů byl použit Python 3.7 do kterého je potřeba doinstalovat několik knihoven. Ty je možné nainstalovat pomocí správce balíčku pro Python **pip**. Pro komunikaci pomocí MQTT protokolu byla zvolena nejznámější knihovna v této oblasti a to **Paho MQTT Client**[1]. Oba nástroje pracují s databází, do které ukládají aktivní uzly na síti. Aby bylo možné oba nástroje jednoduše rozšířit byl zvolen ORM[2] přístup. Je tedy potřeba nainstalovat knihovnu **SQLAlchemy**[3], která poskytuje metody jak pro přímou komunikaci s databází pomocí SQL dotazů, tak právě ORM přístup. Python ani SQLAlchemy v základu neobsahují třídy pro komunikaci s MySQL databází a je tedy třeba ještě přidat knihovnu **mysqlclient**[4]. Databáze je poté společná pro oba nástroje.

Před samotným spuštění jednotlivých nástrojů je potřeba nastavit připojení k MQTT serveru a databázi. To lze provést v souboru `settings.py`. Zde je pro každé připojení vytvořeno asociativní pole, do kterých je třeba vyplnit správné údaje.

6.1 Remote control

Celým cílem toho to nástroje bylo ulehčit nastavování a nahrávání aplikací na uzly. Výsledkem je konzolová aplikace, která nabízí soubor nejčastěji používaných příkazů.

Spuštění nástroje se provede pomocí příkazu `python remote.py` spuštěného ve složce s nástroji. Při prvním spuštění se nástroj pokusí vytvořit databázové schéma. Dále je uživateli zobrazeno textové menu, které je možno vidět na obrázku 6.1, s výčtem všech dostupných akcí a čeká se na uživatelský vstup s výběrem akce. Uživatel je pak proveden průvodcem, který si vyžádá zadání všech potřebných informací pro provedení konkrétní akce. První informací, kterou musí uživatel zadat je název uzlu v síti nebo jeho IP adresa. Pokud byl zadán uzel, který ještě není uložen v databázi je zkontrolována síť zda takový uzel vůbec existuje. Pokud ano je uložen do databáze a pokračuje se vykonáváním uživatelem vybrané akce.

Každá akce si ukládá do databáze informace o tom co bylo provedeno. V případě že použijeme **Remote control** pro upload zdrojových kódů uživatelské aplikace na uzel, nebo

[1]https://github.com/eclipse/paho.mqtt.python
[2]Object-relational mapping
[3]https://www.sqlalchemy.org/
[4]https://github.com/PyMySQL/mysqlclient-python

přidání záznamu do směrovací tabulky, je tato informace uložena ke konkrétnímu uzlu. Později tak můžeme využít akce `Restore Node`, která si pro zadaný uzel zjistí z databáze všechny známé informace a pokusí se na uzel postupně nahrát směrovací tabulku, zdrojové kódy aplikací a spustit ty aplikace u kterých má záznam že byli při původním nahrávání aktivovány.

V případě že uživatel při zadávání informací pro vybranou akci zjistí že zadal některou z hodnot špatně, nebo zvolil špatnou akci, může se stiskem kombinace kláves `control+c` vrátit do hlavního menu nástroje.

```
C:\Users\      \PycharmProjects\MPOS_monitoring>python remote.py
######## MPOS Remote control ########
a) Restart node
b) Factory reset
c) Add route
d) Remove route
e) Upload module
f) Remove module
g) Activate module
h) Deactivate modlue
i) Status info
j) Settings mode
t) Restore Node
q) Exit Tool
Choose action:
```

Obrázek 6.1: Menu nástroje Remote Control

6.2 Monitoring

Druhým nástrojem je aplikace, která bude běžet na pozadí a monitorovat komunikaci probíhající na síti. Po startu zkontroluje zda je databázové schéma aktuální, připojí se na MQTT server a začne odposlouchávat komunikaci na systémových topicích. V případě že přijme zprávu `NODE_ADD` označující spuštění uzlu, zkontroluje zda o něm má již v databázi záznam případně jej vytvoří. Každých pět minut navíc vyšle zprávu `DISCOVER`, aby se mu ozvaly všechny aktuálně spuštěné uzly. V takovém případě očekává zprávy `NODE_ALIVE`. Při jejím přijetí aktualizuje čas poslední kontroly uzlu.

Pokud monitorovací nástroj přijme zprávu `NODE_ADD` u uzlu který byl online v posledních patnácti minutách, vyhodnotí to jako restart uzlu. Pak se na základě informací uložených v databázi snaží znovu spustit aplikace, které na zadaném uzlu běželi. Záznam o tom jaké aplikace jsou na uzlu spuštěny je přidána do databáze nástrojem **Remote control**.

Kapitola 7

Demo příklad

Testování systému probíhalo na jednoduchém příkladu, kdy úkolem systému je spínání vzduchotechniky. Stručný návrh příkladu byl popsán v kapitole 4.1. Následující kapitola se bude zabývat jeho popisem a zároveň zde bude uveden postup nastavení uzlu od nahrání firmwaru až po samotné spuštění aplikací.

Výsledný příklad byl ponechán spuštěný několik dní pro otestování zda nedojde k chybě při delším běhu. Během této doby se však žádné neprojevily. Výsledné zapojení hardwaru pro příklad je možné vidět na obrázku 7.6.

7.1 Příprava ESP

V případě přidávání nového uzlu do sítě, je první potřeba zkontrolovat zda je na ESP nahrán Micropython firmware. Pokud ne, lze toho docílit pomocí nástroje `esptool`[1]. Ke zdrojovým souborům systému, jsou přiloženy předkompilované verze firmwaru, které lze použít pro nahrání. Pokud uživatel bude chtít použít nejnovější firmware, bude třeba si firmware skompilovat podle návodu uvedeného u portu pro konkrétní platformu, umístěného na gitovém repozitáři Micropythonu[2]. Před kompilací je ale potřeba přidat všechny potřebné knihovny, které byly zmíněny v kapitole 5.1. Před nahráním firmwaru je nutné nejdříve smazat aktuální obsah flash paměti a poté teprve provést příkazy pro nahrání. Všechny příkazy jsou na obrázku 7.1. Příkaz pro nahrání firmwaru, `write_flash`, je zde uveden dvakrát, jelikož pro pro každou verzi desky se lehce liší. První nahrávací příkaz je určen pro desku ESP8266 a druhý pro ESP32.

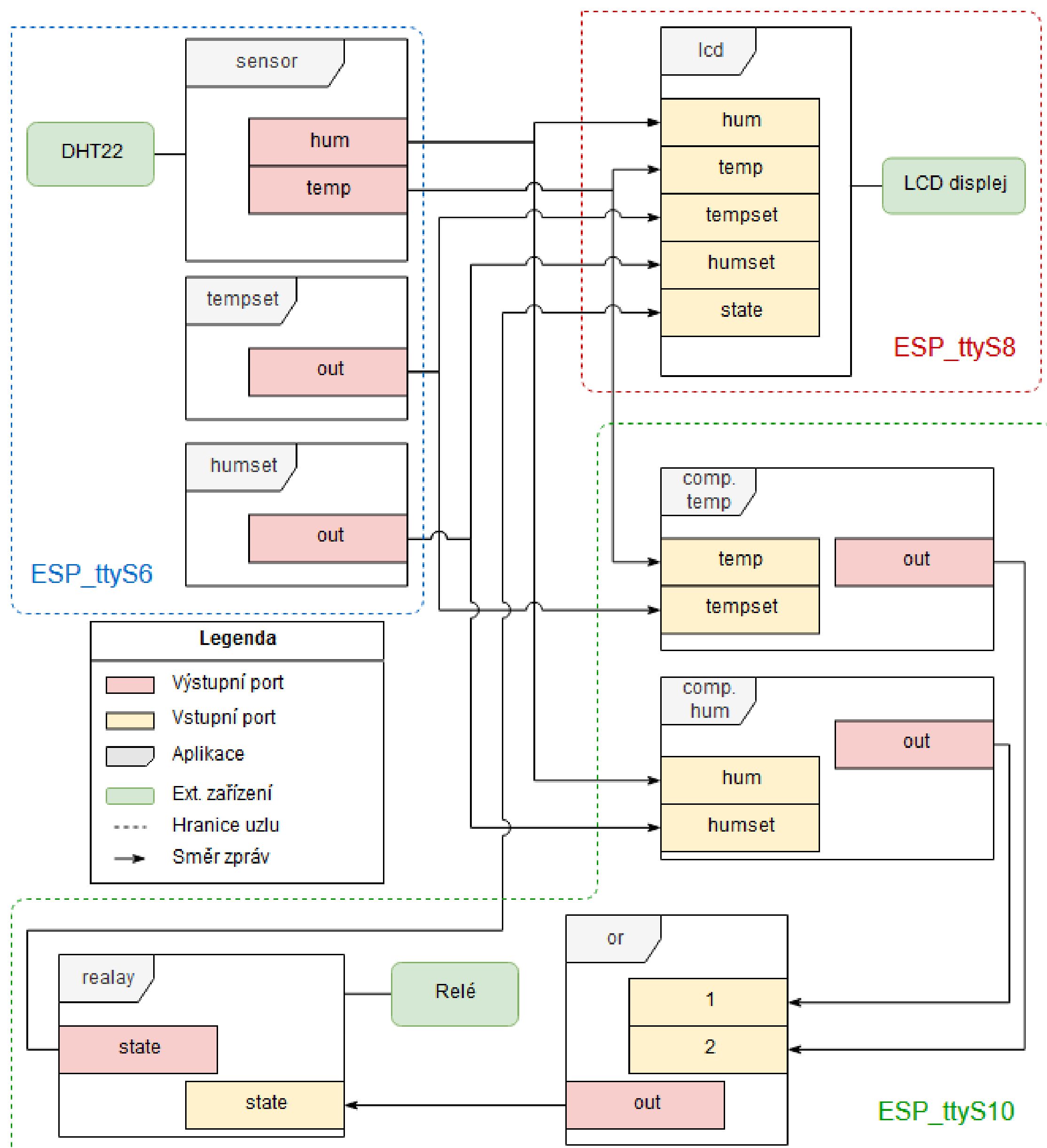
```
esptool.py -p /dev/ttyUSB0 erase_flash

esptool.py -p /dev/ttyUSB0 write_flash -fm dio 0 firmware_esp8266.bin
esptool.py -p /dev/ttyUSB0 write_flash -z 0x1000 firmware_esp32.bin
```

Obrázek 7.1: Příkazy pro smazání flash paměti a nahrání firmwaru

Pokud deska již firmware obsahuje lze do ní např pomocí nástroje `mpfshell`, který byl již zmiňován výše, nahrát všechny soubory systému.

[1] https://github.com/espressif/esptool
[2] https://github.com/micropython/micropython/tree/master/ports/esp32



Obrázek 7.2: Schéma aplikací a komunikace v demo aplikaci

7.2 Nastavení uzlů

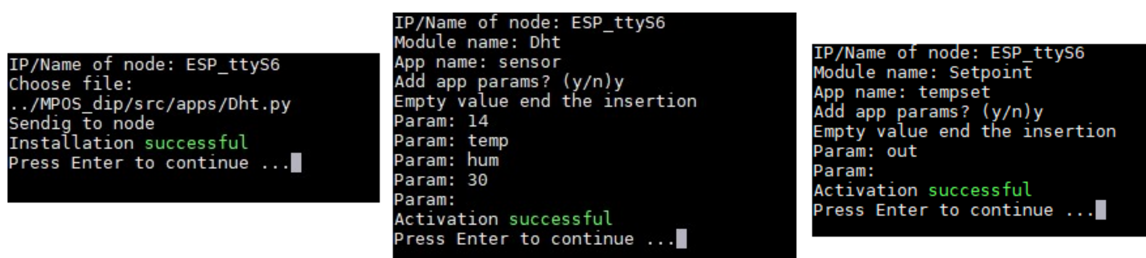
V rámci demo příkladu byly na jednotlivé uzly nahrány již předvytvořené konfigurační soubory, se jmény uzlů a nastavením wifi bodů pro připojení. Jména jednotlivých uzlů neodpovídají jejich funkcionalitě, ale byly voleny podle sériových portů, které jim byly přiřazeny v počítači na kterém probíhal vývoj. Toto pojmenování bylo použito pro rychlejší ladění chyb. Díky nahraným konfiguračním souborům, tak nebylo třeba použít konfiguračního webu.

Celý demo příklad se bude skládat ze třech uzlů, na kterých dohromady poběží osm aplikací, které se budou starat o monitorování aktuální teploty a vlhkosti v místnosti. Pokud teplota bude nižší než nastavená, nebo vlhkost bude vyšší než nastavený limit, bude rozsvícena Led dioda simulující relé, které by bylo v reálném provozu použito pro sepnutí vzduchotechniky. Všechny hodnoty, které se v systému pohybují navíc budou zobrazeny na dvou řádkovém LCD displeji. Schéma celého systému i s vyznačenými komunikačními cestami je zobrazeno na obrázku 7.2.

V původním návrhu se počítalo že aplikace `tempset` a `humset`, sloužící pro nastavení hraničních hodnot, bude možno ovládat pomocí tlačítek, které jsou umístěny na modulu s displejem. Bohužel tyto tlačítka se nepovedlo zprovoznit, proto ovládání těchto dvou aplikací bude prováděno pomocí odesílání zpráv z nástroje **Remote control**. Pomocí tohoto nástroje budou také nahrávány všechny aplikace na jednotlivé uzly.

Uzel ESP_ttyS6 - Teploměr

Na tomto uzlu boudou spuštěny tři aplikace. Jedna aplikace pro samotné měření hodnot ze senzoru **DHT22** a dvě aplikace **Setpoint**, které boudou obsahovat uživatelem nastavené hodnoty pro sepnutí relé. Na samotný uzel tedy stačí nahrát dva zdrojové kódy. Nahrání obou souborů se liší pouze cestou ke zdrojovému souboru. Proto na obrázku 7.3 je pouze nahrání souboru s modulem **Dht** a poté aktivace aplikací `sensor` a `tempset`. Aktivace aplikace `humset` není zobrazena, protože až na změnu názvu je stejná jako aktivace aplikace `tempset`. U aktivací je navíc na obrázku vidět zadávání parametrů, které jsou potřeba pro její spuštění. Aplikace **Dht** potřebuje číslo pinu, na kterém je připojen senzor, jména portů pro odesílání teploty a vlhkosti a poslední parametry značí po kolika vteřinách má proběhnout nové měření a odeslání hodnot. Modul **Setpoint** využívá jen jeden parametr a to název portu pro výstup.



Obrázek 7.3: Nahrávání a aktivace aplikací do uzlu

Aby aplikace odesílaly správně svá data je potřeba nastavit směrovací tabulku. Zde uvedu jen jeden příklad nastavení a poté už jen celý obsah směrovací tabulky pro tento uzel. Po dokončení nastavení směrovací tabulky již není potřeba dále nic nastavovat. Na uzlu běží všechny potřebné aplikace a zprávy jsou odesílány do sítě.

```
IP/Name of node: ESP_ttyS6
Src app: sensor
Src port: temp
Target Node: ESP_ttyS8
Target App: lcd
Target Port: temp
Add next target?(y/n): y
Target Node: ESP_ttyS10
Target App: comptemp
Target Port: temp
Add next target?(y/n): n
2 routes add
Press Enter to continue ...
```

Obrázek 7.4: Nastavení směrovací tabulky pro aplikaci `temp` a port `temp`

```
1  {
2    "sensor": {
3      "hum": [
4        {"t": "ESP_ttyS8", "p": "hum", "a": "lcd"},
5        {"t": "ESP_ttyS10", "p": "hum", "a": "comphum"}
6      ],
7      "temp": [
8        {"t": "ESP_ttyS8", "p": "temp", "a": "lcd"},
9        {"t": "ESP_ttyS10", "p": "temp", "a": "comptemp"}
10     ]
11   },
12   "humset": {
13     "out": [
14       {"t": "ESP_ttyS8", "p": "humset", "a": "lcd"},
15       {"t": "ESP_ttyS10", "p": "humset", "a": "comphum"}
16     ]
17   },
18   "tempset": {
19     "out": [
20       {"t": "ESP_ttyS8", "p": "tempset", "a": "lcd"},
21       {"t": "ESP_ttyS10", "p": "tempset", "a": "comptemp"}
22     ]
23   }
24 }
```

Obrázek 7.5: Směrovací tabulka pro uzel ESP_ttyS6

Uzel ESP_ttyS8 - Displej

Tento uzel jako jediný obsahuje pouze jednu aplikaci a to pro zobrazování přijatých hodnot na displeji. Zdrojové kódy této aplikace jsou přiloženy u systému v souboru `Lcd.py`. Nahrání aplikace na uzel a její aktivace bude probíhat stejným způsobem jako u předchozího uzlu. Uzel navíc nepotřebuje žádnou směrovací tabulku jelikož nebude odesílat žádná data.

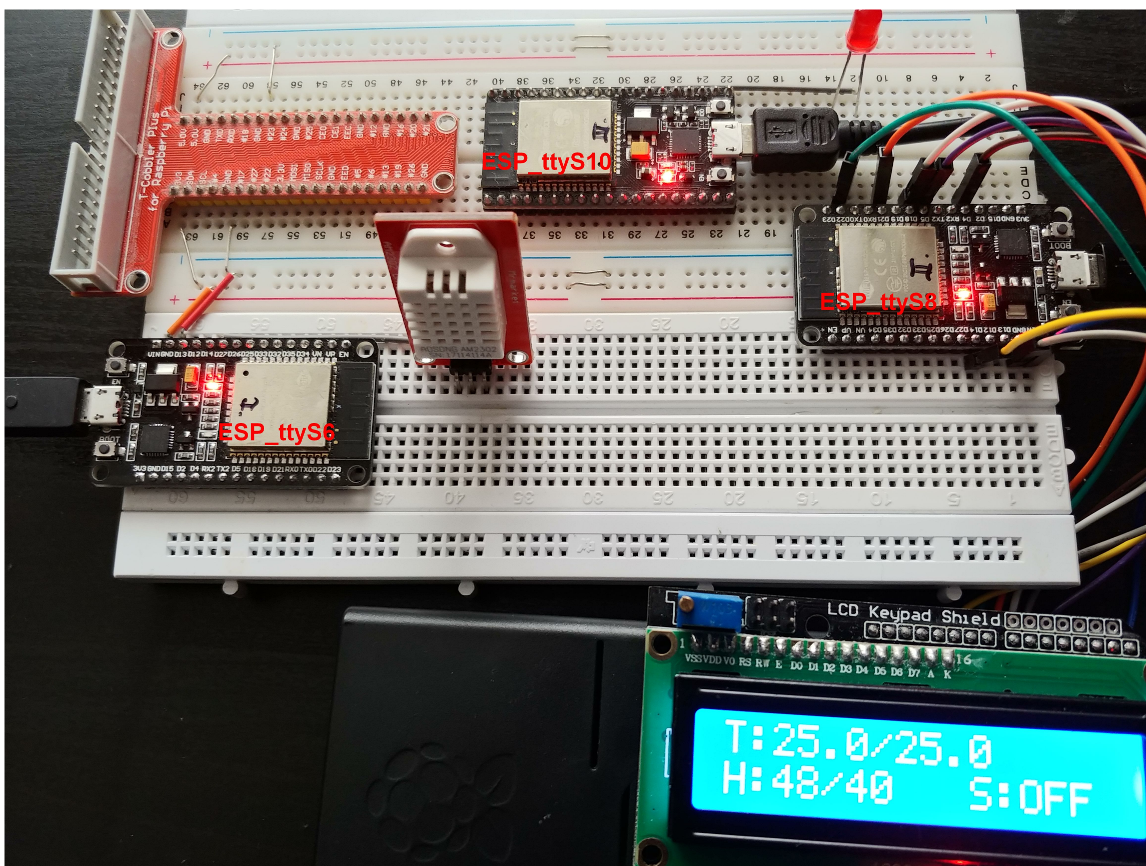
Teplota a vlhkost jsou na displeji zobrazeny jako „`aktuální hodnota"`/„`nastavená hodnota"`. Stav aplikace `relay` z uzlu **ESP_ttyS10** je zobrazen v pravém dolním rohu displeje. Toto zobrazení je možné vidět na obrázku 7.6.

Uzel ESP_ttyS10 - Relé

Zde bude běžet nejvíce aplikací a bude zde probíhat hlavní logika pro určení v jakém stavu má být aplikace `relay`.

Prvním typem aplikace která běží na tomto uzlu je aplikace **Comparator**, která bude spuštěna ve dvou instancích. A to `comptemp`, která na vstupu dostává aktuální teplotu a na druhém vstupu teplotu nastavenou uživatelem. Pokud bude aktuální teplota nižší než nastavená pošle na výstup hodnotu jedna. V ostatních případech bude na výstupu hodnota nula. Druhá instance aplikace s názvem `comphum`, dostává na vstupech informace o nastavené a aktuální vlhkosti, ale na rozdíl od aplikace pro kontrolu teploty, vrací na výstup jedna tehdy, když je aktuální vlhkost vyšší než nastavená hodnota.

Výstupy aplikací porovnávajících teplotu a vlhkost jsou poslány do aplikace `or`, která provádí logickou operaci **OR** ze svých vstupů a výsledek této operace předává na výstup. Ten je přesměrován do aplikace `relay`, která podle něj přepne výstup, který ovládá Led diodu. Aplikace navíc při změně stavu svůj nový stav přepošle na displej.



Obrázek 7.6: Fotografie reálného zapojení

Kapitola 8

Závěr

Cílem této práce bylo nastudovat aktuální trendy v oblasti Internetu věcí a vývoje rekonfigurovatelného softwaru. K dispozici byla také experimentální implementace operačního systému pro rekonfigurovatelný IoT uzel založený na platformě ESP8266 a ESP32. Na základě získaných informací poté navrhnout a implementovat operační systém uzlu, který poskytne větší funkcionalitu oproti původní implementaci. Dále se zaměřit na ukládání získaných dat a monitorování sítě.

Implementace zmíněného systému pro platformu ESP8266 a ESP32 proběhla v jazyce Micropython, který dovoluje změnu programu za běhu. Výsledný systém je složen z několika objektů díky čemuž nebude do budoucna problém jej upravit nebo rozšířit. Největší rozdíl oproti experimentální implementaci je v ukládání uživatelských aplikací na uzlech. V původní implementaci byly všechny aplikace drženy v paměti RAM, kde zabíraly místo, které by mohlo být použito pro načtení knihoven pro ovládání hardwaru. Při restartu uzlu byly navíc všechny aplikace zapomenuty a bylo je třeba nahrát znovu. V aktuální verzi se aplikace ukládají jako soubory do Flash paměti uzlu a jsou načteny až v případě že jsou potřeba. Podobné úpravy se dočkala i směrovací tabulka, která je nyní také uložena v souboru. Při restartu uzlu tedy stačí provést pouze reaktivaci aplikací.

Při návrhu systému byl navíc lehce upraven komunikační protokal a rozložení komunikace na více MQTT topiců, čímž bylo eliminováno zahlcení uzlu zprávami, které nebyly pro něj. Původně se počítalo i s implementací původního komunikačního protokolu, avšak kvůli nedostatku paměti RAM, zvláště u desky ESP8266, byl tento záměr opuštěn a byla implementována pouze upravená verze protokolu.

Aktuální verze Micropythonu bohužel nedovolovala upravit některé klíčové vlastnosti systému. Především se jedná o paralelní běh uživatelských aplikací, kterého je docíleno pomocí knihovny μAsyncio, která obsahuje implementaci kooperativního multitaskingu. Aplikace tedy musí manuálně předávat řízení ostatním subjektům. Tento přístup musel být použit jelikož ESP8266 nepodporuje vlákna a u ESP32 je implementace v experimentální fázi a je velmi nestabilní. V případě že aplikace nepředá procesor dále, celý uzel přestane komunikovat s okolím a je nutný jeho restart pomocí tlačítka na desce. Tento problém jsem se pokusil řešit pomocí hardwarového časovače, který by kontroloval stav uzlu. Bohužel přerušení vyvolané tímto časovačem způsobilo ztrátu kontextu v rámci knihovny μAsyncio a nebylo tedy možné dál pokračovat v běhu. Dále i když ESP32 obsahuje dvoujádrový procesor, lze zatím pro běh aplikací využít pouze jedno. Při psaní této práce se však začaly objevovat první pokusy o rozšíření implementace Micropythonu o možnost využití obou jader.

Ani jedna z desek svou kapacitou RAM paměti nedostačuje k plnohodnotnému využití databázového serveru spuštěném přímo na uzlu. Proto je potřeba pro ukládání historických dat využít externí databázi. V rámci vývoje byla pro ukládání využita služba Domoticz. Domoticz lze navíc využít jako přehledný dashboard se zobrazením aktuálních hodnot na senzorech. Navíc díky programování eventů, spouštěných na základě uživatelem určených podmínek lze Domoticz využít i jako monitorovací systém. Jedinou nevýhodou je nemožnost ovlivnit odchozí zprávy z této služby což bez implementace další aplikace znemožňuje využití služby Domoticz pro přímé ovládání sítě uzlů.

V rámci vývoje systému byly vytvořeny dva pomocné nástroje. A to nástroj pro jednoduché nahrávání a spuštění aplikací na uzlech a monitorovací služba. Jelikož ale v případě chyby na uzlu ze které se operační systém sám nedokáže zotavit, dojde ke ztrátě komunikace, nelze pomocí monitorovací služby nijak zasáhnout. Tato služba se proto pouze soustředí na ukládání aktuálně aktivních uzlů do databáze a v případě zaznamenaného restartu uzlu se pokusí na tomto uzlu aktivovat všechny aplikace o kterých má záznam že na něm před restartem běžely.

Systém je teoreticky připraven pro nasazení pro řízení reálné domácnosti. Avšak nemožnost vzdáleného restartu v případě že na uzlu nastane chyba, znamená velkou překážku. Primárním rozšířením do budoucna by mělo tedy být vyřešení tohoto problému, díky čemuž by se i otevřela možnost vytvoření inteligentnějšího monitorovacího systému, který by mohl aktivně zasáhnout do sítě. Velmi přínosné by bylo vytvoření aplikace, která by se starala o překlad zpráv ze služby Domoticz případně úprava samotné služby aby ji bylo možno využít pro ovládání sítě.

Literatura

[1] *Co je to SCADA?* Promotic.eu, [Online; navštíveno 04.01.2019].
URL https://www.promotic.eu/cz/pmdoc/WhatIsPromotic/WhatIsScada.htm

[2] *ESP8266*. esp8266.net, [Online; navštíveno 13.01.2019].
URL http://esp8266.net/

[3] *Everything You Need to Know About Distributed Control System*. Elprocus.com, [Online; navštíveno 26.12.2018].
URL https://www.elprocus.com/distributed-control-system-features-and-elements/

[4] *History of Arduino*. core-electronics.com.au, [Online; navštíveno 11.01.2019].
URL https://core-electronics.com.au/tutorials/history-of-arduino.html

[5] *MicroPython pyboard feature table*. micropython.org, [Online; navštíveno 10.01.2019].
URL https://store.micropython.org/pyb-features

[6] *SCADA Basic*. integraxor.com, [Online; navštíveno 04.01.2019].
URL https://www.integraxor.com/support/scada-training/scada-basics/

[7] *SCADA systems (Supervisory Control and Data Acquisition): evolutions and market 2022*. i-scoop.eu, [Online; navštíveno 04.01.2019].
URL https://www.i-scoop.eu/industry-4-0/scada-supervisory-control-data-acquisition/

[8] *What is a Distributed Control System?* ControlStation.com, [Online; navštíveno 26.12.2018].
URL https://controlstation.com/what-is-a-distributed-control-system/

[9] *Gartner Says 8.4 Billion Connected "Things"Will Be in Use in 2017, Up 31 Percent From 2016*. Gartner.com, Únor 2017, [Online; navštíveno 06.01.2019].
URL https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016

[10] *What is SCADA?* inductiveautomation.com, Září 2018, [Online; navštíveno 04.01.2019].
URL https://inductiveautomation.com/resources/article/what-is-scada

[11] Barragán, H.: *The Untold History of Arduino* . arduinohistory.github.io, [Online; navštíveno 11.01.2019].
URL https://arduinohistory.github.io

[12] Burgess, M.: *What is the Internet of Things?* Wired.co.uk, Únor 2018, [Online; navštíveno 06.01.2019].
URL https://www.wired.co.uk/article/internet-of-things-what-is-explained-iot

[13] Drahovský, P.: *Rekonfigurovatelný IoT uzel na bázi ESP8266/ESP32*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018.
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=20280

[14] Espressif: *ESP32 Technical reference manual*. espressif.com, [Online; navštíveno 12.01.2019].
URL https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

[15] Espressif: *ESP8266 Technical reference*. espressif.com, [Online; navštíveno 12.01.2019].
URL https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf

[16] Fruhlinger, J.: *The Mirai botnet explained: How teen scammers and CCTV cameras almost brought down the internet*. csoonline.com, Březen 2018, [Online; navštíveno 06.01.2019].
URL https://www.csoonline.com/article/3258748/security/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought-down-the-internet.html

[17] Goyal, S.: *Centralized vs Decentralized vs Distributed*. Medium.com, Červenec 2015, [Online; navštíveno 26.12.2018].
URL https://medium.com/delta-exchange/centralized-vs-decentralized-vs-distributed-41d92d463868

[18] Haughn, M.: *What is a Distributed Control System?* WhatiIs.com, Září 2017, [Online; navštíveno 26.12.2018].
URL https://whatis.techtarget.com/definition/distributed-control-system

[19] Korejtko, T.: *Internet věcí s uzly na bázi PNVM*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2017.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=19254

[20] Krambeck, D.: *An Introduction to SCADA Systems*. allaboutcircuits.com, Srpen 2015, [Online; navštíveno 04.01.2019].
URL https://www.allaboutcircuits.com/technical-articles/an-introduction-to-scada-systems/

[21] Linda Rosencrance, I. W., Sharon Shea: *What is the Internet of Things (IOT)?* techtarget.com, Červen 2018, [Online; navštíveno 06.01.2019].
URL https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

[22] Ranger, S.: *What is the IoT? Everything you need to know about the Internet of Things right now*. Zdnet.com, Srpen 2018, [Online; navštíveno 06.01.2019].

URL https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/

[23] Stehlík, P.: *ESP32 je tu. Co přinese nástupce ESP8266?* . Root.cz, Září 2016, [Online; navštíveno 12.01.2019].
URL https://www.root.cz/clanky/esp32-je-tu-co-prinese-nastupce-esp8266/

[24] Yegulalp, S.: *Micro Python's tiny circuits: Python variant targets microcontrollers.* infoworld.com, Červen 2014, [Online; navštíveno 11.01.2019].
URL https://www.infoworld.com/article/2608012/python/micro-python-s-tiny-circuits--python-variant-targets-microcontrollers.html