VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KOORDINACE INTERNETU VĚCÍ POMOCÍ PETRIHO SÍTÍ

COORDINATION OF INTERNET OF THINGS BY MEANS OF PETRI NETS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. PATRIK UNZEITIG**
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: **Unzeitig Patrik, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Koordinace internetu věcí pomocí Petriho sítí**
Coordination of Internet o Things by Means of Petri Nets
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 ESP32, komunikující protokolem MQTT. Seznamte se s knihovnou SNAKES pro podporu implementace Petriho sítí v jazyce Python.
3. Navrhněte interpret vysokoúrovňových Petriho sítí, umožňující v rámci DCS specifikovat řízení na různých úrovních Petriho sítěmi (PN). Kód pro komponenty na bázi PN generujte z vhodného vizuálního editoru Petriho sítí.
4. Navržený interpret a podpůrné prostředky implementujte a proveďte testování za pomoci vhodné DCS 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
- SNAKES. URL: https://www.ibisc.univ-evry.fr/~fpommereau/SNAKES/
- RENEW. URL: http://www.renew.de/

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 tvorbou systému umožňujícího dynamické nahrávání a běh aplikací s použitím interpretovaného jazyka Python. Vytvářený systém je založen na MPOS. Do systému lze nahrávat aplikace ve formě zdrojového kódu v jazyce Python nebo vysokoúrovňové Petriho sítě popisující komponenty systému posílané ve formátu PNML RefNet. Pro vytvoření Petriho sítě se používá vizuální editor Renew, který umožňuje export do PNML RefNet. Přijatá Petriho síť je převedena na aplikaci systému v jazyce Python, která umožňuje interpretaci dané Petriho sítě s využitím rozšířené knihovny SNAKES. Pro komunikaci mezi aplikacemi běžícími na libovolném uzlu ve stejné síti se používá protokol kompatibilní s protokolem použitým v MPOS, komunikující prostřednictvím MQTT. Výsledný software může být použit k řízení vytápění v budově a jeho funkcionalita lze dále rozšiřovat použitím nových aplikací. K provádění kontroly, ovládání a ukládání dat do databáze slouží systém pro automatizaci domácnosti Domoticz.

Abstract

This thesis focuses on implementation of system which supports dynamic load and run of user applications with use of interpreted language Python. It's based on MPOS (operation system for ESP8266/ ESP32). The system is able to load application in the form of source code or High-level Petri Net used to describe components in PNML RefNet format. The Petri Nets are created in visual editor Renew which supports exporting to a file in PNML RefNet format. Receieved Petri Net is converted to system's application in Python which can be used to interpret received Petri Net by using SNAKES library with created plugin. User applications use MPOS compatible protocol to communicate with other applications running on any node in the same network via MQTT. The implemented software can be used for heating control in home and new features can be added by using of new applications. Home automation system Domoticz is used to control, monitor and save data to database.

Klíčová slova

Internet věcí, IoT, Vysokoúrovňové Petriho sítě, SNAKES, Renew, ESP32, Raspberry Pi, MQTT, PNML RefNet, Domoticz

Keywords

Internet of Things, IoT, High-level Petri Nets, SNAKES, Renew, ESP32, Raspberry Pi, MQTT, PNML RefNet, Domoticz

Citace

UNZEITIG, Patrik. *Koordinace internetu věcí pomocí Petriho sítí*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Vladimír Janoušek, Ph.D.

Koordinace internetu věcí pomocí Petriho sítí

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.

.....
Patrik Unzeitig
20. května 2019

Poděkování

Děkuji svému vedoucímu doc. Ing. Vladimíru Janouškovi, Ph.D. za cenné rady a připomínky při tvorbě této diplomové práce.

Obsah

Kapitola 1

Úvod

V posledních letech se díky nízké ceně, malým rozměrům, nízké spotřebě a širokým možnostem využití začala rozšiřovat zařízení vhodná k použití v oblasti Internetu věcí. Nejčastěji se s těmito typy zařízení můžeme setkat chceme-li dálkově ovládat nějaká zařízení či sbírat data ze senzorů. Nezpochybnitelnou výhodou je schopnost bezdrátové komunikace nebo rovnou možnost připojení k internetu a tudíž možnost komunikace s téměř libovolným zařízením.

Tato práce se bude zabývat využitím vysokoúrovňových Petriho sítí (High-level Petri Nets) v oblasti Internetu věcí. Hlavním předmětem je návrh a vytvoření interpretu vysokoúrovňových Petriho sítí. Pomocí Petriho sítí bude možné specifikovat řízení na různých úrovních DCS. Pro jednodušší a rychlejší návrh bude možné vygenerovat požadovanou Petriho síť s použitím vizuálního editoru. Vyvinutý interpret bude využit v systému pro řízení vytápění budovy, jehož komponenty budou moci být navrženy Petriho sítěmi. Komunikace mezi zařízeními bude probíhat po síti s využitím komunikačního protokolu MQTT. Pro uživatelsky přívětivější přístup k informacím o naměřených hodnotách a aktuálnímu stavu vytápění bude použit některý ze systémů domácí automatizace. Ten umožní sledování a nastavování hodnot systému skrze webový prohlížeč jak z počítače tak z mobilních zařízení.

V první fázi se práce zabývá distribuovanými řídicími systémy, systémy SCADA a Internetem věcí. V další fázi se zaměřuje na open-source systémy používané k automatizaci domácnosti. Dále pak na cílové platformy, které lze použít při realizaci výstupu této práce. Následuje seznámení s komunikačním protokolem MQTT, základem do teorie Petriho sítí a vysokoúrovňových Petriho sítí. Další fáze obsahuje analýzu a návrh systému, který bude podporovat interpretaci Petriho sítí. Poslední fáze se zabývá implementací a testováním výsledného systému pro řízení teploty v domácnosti.

Kapitola 2

Systémy řízení

Tato kapitola se zabývá teoretickými informacemi ohledně systémů zaměřených na řízení a sběr dat. V první části jsou popsány principy distribuovaných řídicích systémů a jejich rozdělení do úrovní. Další kapitola seznamuje se systémy SCADA a jejich využitím. V poslední podkapitole je vysvětlen vznik a význam internetu věcí a technologie používané ke komunikaci mezi zařízeními.

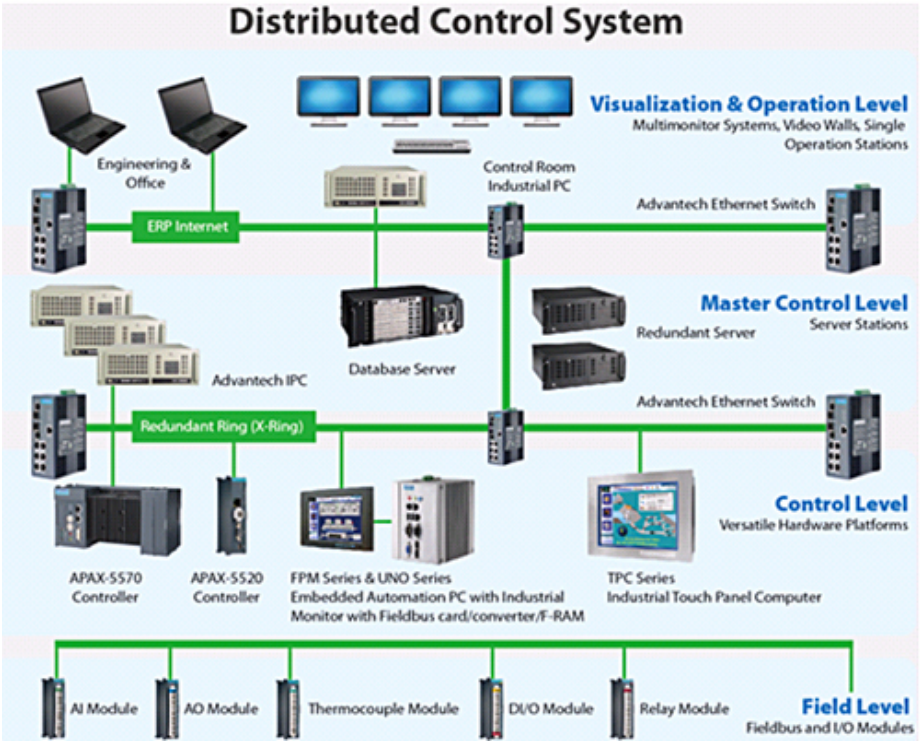
2.1 Distribuované řídicí systémy

Distribuovaný řídicí systém[1] je složen, na rozdíl od centralizovaných řídicích systémů, z několika samostatně fungujících podsystémů. Jelikož centralizované systémy obsahují pouze jeden centrální prvek, při poruše tohoto prvku systém havaruje. Tento problém se distribuovaných systémů netýká. Samostatně fungující podsystémy se společně podílejí na řízení a mohou používat zcela odlišný hardware, operační systém apod. Mezi sebou jsou podsystémy propojeny komunikační sběrnicí, prostřednictvím které komunikují. Smyslem je efektivně koordinovat práci všech systémů k dosažení určitého cíle, případně koordinovat používání sdílených zdrojů. Díky vzniku výkonných a spolehlivých sběrnic a také díky nižší ceně jednotlivých zařízení s procesorem, se začal tento přístup v automatizaci rychle rozšiřovat. Zlevněním došlo k možnosti nahrazení jednoho výkonného řídicího prvku více méně výkonnými prvky, které jsou schopny vykonat stejnou činnost. Komunikace je tedy jedním ze základních předpokladů pro realizaci takového typu systému. Výhodami jsou vyšší výkonnostní možnosti, vysoká spolehlivost, možnost výstavby složitějších systémů, jednodušší ladění aplikací a možnost postupného rozšiřování. Nevýhodou je nepříznivý vliv prostředí na systém. [16, 26]

Architektura:

- procesory (RTU, PLC)
- komunikační infrastruktura
- vstupně/výstupní moduly spojené se senzory a řídicími zařízeními
- řízení a monitorování prostřednictvím Human-machine interface

[1] v angličtině Distributed Control System (ve zkratce DCS)



Obrázek 2.1: Úrovně distribuovaných řídicích systémů[2]

2.2 SCADA systémy

Supervisory Control And Data Acquisition neboli SCADA je systém, který obecně nezastává funkci plnohodnotného řídicího systému dané technologie, ale zaměřuje se spíše na roli supervizora. Zpravidla je to software fungující nad skutečným řídicím systémem založeným např. na PLC[3] nebo jiných hardwarových zařízeních. Skládá se ze softwaru a hardwaru umožňujícího řídit lokálně nebo vzdáleně procesy, monitorovat, sbírat a zpracovávat real-time data, přímo interagovat se zařízeními jako jsou senzory, ventily, pumpy, motory a dalšími, skrze HMI[4] a zaznamenávat události do logovacích souborů. Mohou komunikovat prostřednictvím průmyslových linek (např.: RS-232, RS-485, profibus), ale v dnešní době jsou také používány klasické počítačové sítě typu Ethernet. Mezi příklady nabízených SCADA systémů lze uvést DAQIS[5], Promotic[6] nebo Reliance[7]. [9, 4]

[2] Převzato z: https://www.advantech.com/power-and-energy/case%20studies/%7Bdf086973-9ef6-4fb6-99e9-f2040bee562/

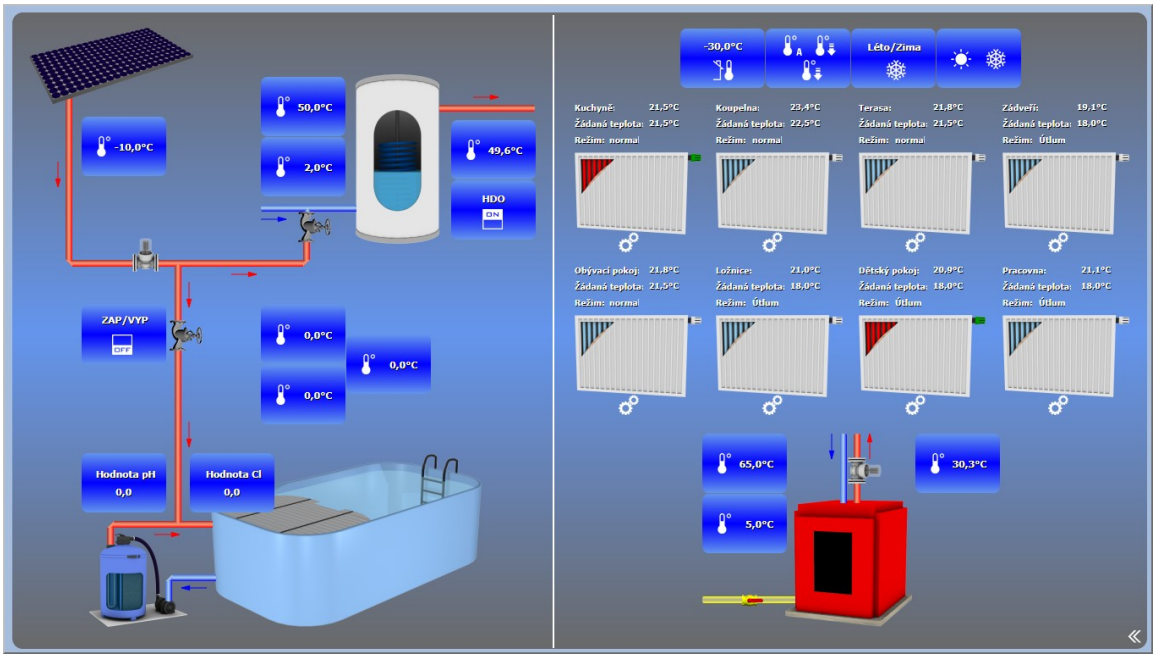
[3] Programovatelný logický automat

[4] Human-machine interface

[5] http://daqis.cz/

[6] https://www.promotic.eu/cz/index.htm

[7] https://www.reliance-scada.com/cs/main



Obrázek 2.2: Ukázka SCADA/HMI systému Reliance[8]

Historie SCADA systémů:

1. generace - ostrovní systémy
   - izolované, nákladné jednoúčelové systémy
   - centrální podnikový mainframe[9]
2. generace - distribuované systémy
   - propojení většího počtu menších stanic proprietárními (uzavřené, neveřejné) komunikačními protokoly
   - jednotlivé stanice mají specifickou funkčnost
3. generace - síťové systémy
   - rozmach počítačových sítí
   - používání otevřených, standardizovaných komunikačních protokolů
   - propojení stanic – PCN (Process Control Network)
4. generace - internet věcí
   - v síti internet může být propojeno téměř vše
   - začínají dominovat cloudové služby

[8] Převzato z: https://www.promotic.eu/cz/index.htm
[9] sálový počítač

2.3 Internet věcí

Pojem Internet věcí poprvé použil ve stejnojmenné prezentaci pan Kevin Ashton v roce 1999 [10], ačkoliv myšlenka propojených zařízení zde byla již dříve. Prvním připojeným spotřebičem k internetu byl automat na nápoje umístěný na univerzitě Carnegie Mellon na začátku 80. let. Prostřednictvím webového prohlížeče bylo možné kontrolovat, zda se v automatu nachází vychlazený nápoj nebo jestli je automat prázdný. Na základě odhadu společnosti Cisco se vznik Internetu věcí datuje na období mezi lety 2008 a 2009, kdy počet zařízení připojených k internetu překonal počet světové populace. [13]

IoT (Internet of Things) je obecně označení pro všechna zařízení připojená k internetu, ale využívá se spíše pro fyzickou síť tvořenou zařízeními vybavenými elektronikou, softwarem, senzory, aktuátory a internetovou konektivitou, sloužící k vzájemné výměně dat. IoT je také přirozeným rozšířením systémů SCADA. Takto propojená zařízení umožňují sběr velkého množství dat, která lze dále zpracovávat a využívat v nejrůznějších oblastech, kterými může být logistika, zdravotnictví, energetika, doprava, meteorologie atd. Dále se tato technologie uplatňuje v oblasti chytrých domů, které mohou pomocí senzorů sledovat počet obyvatel uvnitř a využívat tuto informaci ke snížení výdajů za energii. Zařízení, která lze zapojit do chytré domácnosti mohou být domácí spotřebiče jako lednice, TV, topení, kávovar, žárovky, zásuvky nebo také automobily. Mezi jejich základní vlastnosti patří automatizace, miniaturizace, nízká energetická náročnost a využití bezdrátových technologií. [20]

Typy komunikace

Hlavní myšlenkou internetu věcí je možnost komunikovat s ostatními zařízeními. Propojení nemusí být realizováno jen připojením do sítě, ale je možné použít také přímé připojení. Komunikace IoT se liší od M2M (Machine-to-Machine). V M2M spolu zařízení komunikují jednorázově a naprogramovaně, tedy vědí jaká aplikace či zařízení bude daná data zpracovávat. Příkladem využití M2M komunikace je telemetrie, řízení dopravy, robotika a další. Oproti tomu v IoT probíhá komunikace neustále a neuspořádaně, takže odesílatel nemusí vědět, jaká aplikace či zařízení bude daná data zpracovávat. [24, 22] Některá zařízení mohou být připojena přes kabel, ale většinou je k připojení používána bezdrátová technologie. Použitou technologii je možné zvolit na základě požadavků na spotřebu zařízení a rychlost přenosu dat. Nejčastěji používané typy technologií pro komunikaci IoT:

- Bluetooth
- WiFi
- ZigBee
- Z-Wave
- LoRa
- NB-IoT
- SigFox
- Mobilní síť
- Ethernet

Kapitola 3

Rozbor

Obsahem této kapitoly je automatizace domácnosti, která se zabývá open-source systémy pro monitorování a ovládání domácnosti. Konkrétně multiplatformními systémy Domoticz, openHAB a Home Assistant. V další části se nachází zběžné seznámení s často používanými zařízeními využitelnými nejen pro IoT. Následuje seznámení s jednoduchým komunikačním protokolem MQTT vhodným nejen pro úsporná a méně výkonná zařízení, ale i pro servery. V poslední části se nachází shrnutí pojmů z oblasti Petriho sítí jako jsou síť, P/T Petriho síť, Časové Petriho sítě a poté vysokoúrovňové Petriho sítě.

3.1 Automatizace domácnosti

Využitím vlastností internetu věcí a SCADA systémů v domácnostech získáme možnost automatizace a centrálního ovládání domácnosti. Možností využití je spousta, od ovládání rolet oken (v danou hodinu či podle množství slunečního svitu), řízení vytápění (v každé místnosti zvlášť nebo v předstihu než se vrátíme domů), automatického zhasínání při opuštění domu, až k detektorům kouře či vzdálenému ovládání. Může také ovládat zabezpečení domu řízením přístupu nebo pomocí alarmu. Hlavním cílem je zvýšení komfortu zjednodušením ovládání, automatizací a zefektivnění procesů v domě.

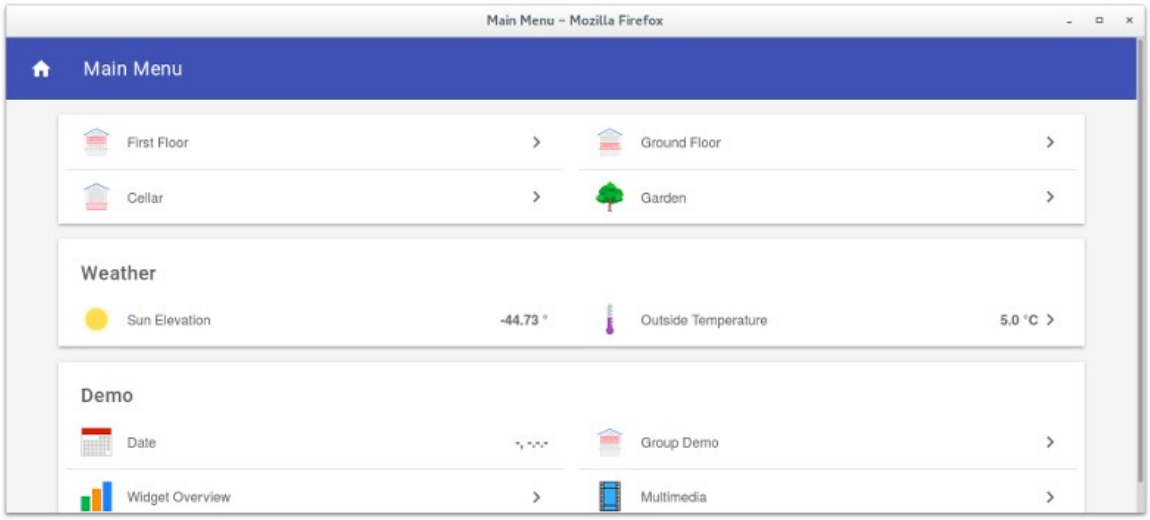
Domoticz

Open-source systém který je navržen, aby fungoval na různých operačních systémech, implementován v C++ s vlastním webovým serverem. Uživatelské prostředí je navrženo v HTML5 a automaticky se přizpůsobuje dle typu zařízení. Systém umožňuje monitoring a konfiguraci různých zařízení, jako světla, vypínače, senzory (teplota, déšť, vítr, UV záření, plyn, ...). Získaná data ze senzorů lze zobrazit pomocí přehledných grafů s různými pevnými časovými intervaly. Podporuje také zasílání upozornění/notifikací do jakéhokoliv mobilního telefonu, emailů či SMS. Po nastavení umožňuje ke komunikaci použití MQTT. Konfigurace se ve většině případů provádí skrze webové rozhraní a jeho funkčnost lze rozšířit použitím doplňků nebo vlastních skriptů. [3]

Obrázek 3.1: Ukázka Domoticz rozhraní[1]

openHAB

Open Home Automation Bus neboli openHAB je open-source platforma k sloužící k automatizaci domácnosti, implementován v jazyce Java, čímž je zaručena přenositelnost. Systém je založen na modulární platformě OSGi. Jedná se o dynamický modulární systém pro jazyk Java, který umožňuje nahrávání balíčků za chodu. K instalaci je dostupná image SD karty pro Raspberry Pi a Pine A64 takže, v případě potřeby, není nutné se zabývat nastavováním systému. Software integruje různé systémy řízení domácnosti, zařízení a technologie do jediného řešení. Podporuje rozšíření pomocí balíčků, takže pokud je potřeba přidat hardware s novým komunikačním protokolem a existuje pro něj potřebný balíček, lze jej i za běhu přidat. MQTT je vzhledem k existenci takového balíčku podporováno. Běží na mnoha platformách včetně Linux, Windows a Mac OS. [7]

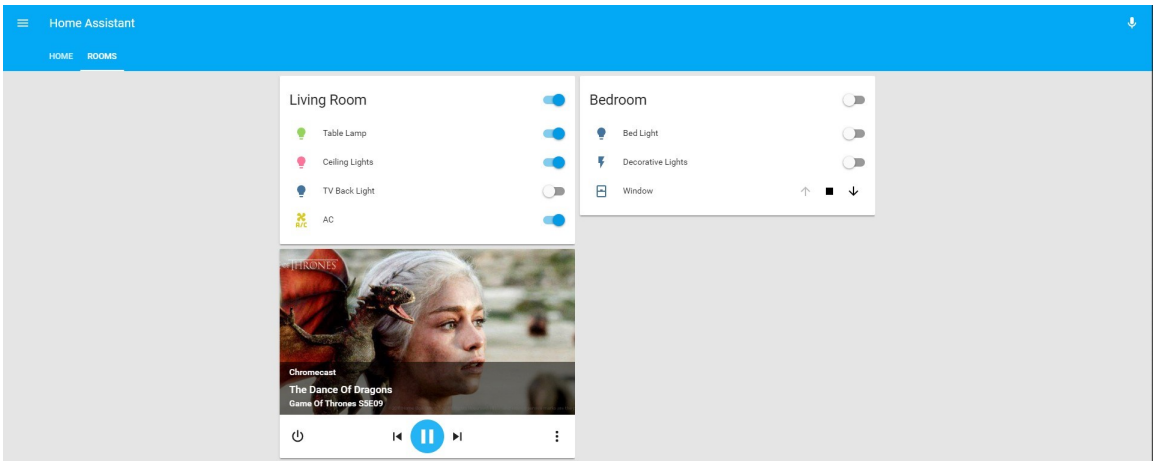


Obrázek 3.2: Ukázka openHAB rozhraní [7]

[1] Převzato z https://durdle.com/2017/01/18/more-on-smartthings-and-domoticz/

Home Assistant

Jedná se o novější z platforem k automatizaci domácnosti, která je taktéž rozšiřitelná pomocí doplňků a je implementována v jazyce Python. Hlavní podporovanou hardwarovou platformou je Raspberry Pi, pro kterou existuje instalační image Hassbian, ale je možné ji nainstalovat i na zařízení s jinými operačními systémy. Obsahuje integrované webové rozhraní sloužící k nastavení systému a umožňuje jednoduché vytvoření a správu zálohování. Podporuje mnoho rozšíření jako například Google Assistant, Let's Encrypt nebo Duck DNS. S využitím rozšíření podporuje komunikaci MQTT taktéž. [6]

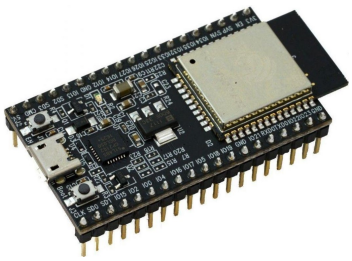


Obrázek 3.3: Ukázka Home Assistant rozhraní [6]

3.2 Cílové platformy

ESP32

Jedná se o nástupce levného WiFi modulu ESP8266 od firmy Espressif Systems[2]. Obsahuje dvě CPU jádra 32bit s nastavitelnou taktovací frekvencí od 80 MHz do 240 MHz, 512 kB SRAM, integrovanou Flash paměť 4 MB, 802.11b/g/n WiFi, Bluetooth ve verzi 4.2 s podporou BLE(Bluetooth Low Energy) a periferie zahrnují kapacitní dotykové senzory, Hallův snímač, zesilovač s nízkým šumem, rozhraní pro SD kartu, Ethernet, vysokorychlostní SPI, UART, I2S a I2C. Klidový proud je menší než 5 µA, tudíž je modul vhodný i pro využití s bateriovým napájením a nositelnou elektroniku. K programování lze použít například jazyky C, C++, JavaScript, Lua nebo MicroPython. [8, 21]



Obrázek 3.4: ESP32-DevKitC [8]

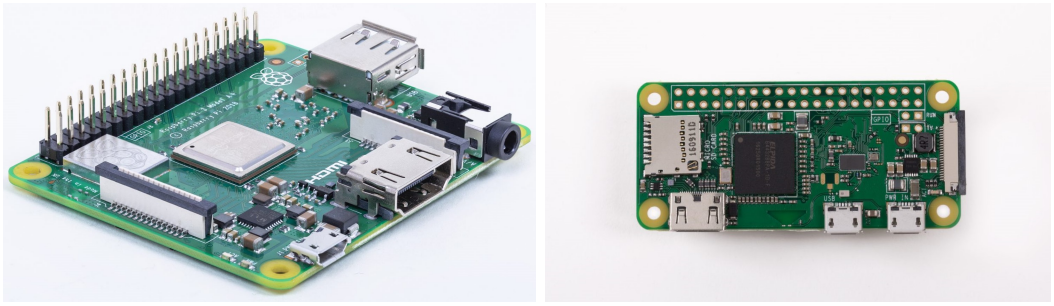
[2] https://www.espressif.com/

Raspberry Pi

Základní verzí je jednodeskový počítač o rozměru platební karty (85x56mm), přičemž nejmenší varianta Zero je přibližně poloviční (65x30mm). Existuje v několika variacích a aktuálně se nachází již ve třetí generaci. V současnosti jsou nabízeny tři velikosti desek s různými specifikacemi. Mezi periferie patří USB, GPIO, SPI, I2C, UART, DPI. V případě potřeby lze variantu Raspberry Pi 3 Model B+ dovybavit příslušenstvím The PoE HAT umožňující napájení skrze Ethernet. Oficiálně podporovaným operačním systémem je Raspbian, který je založený na Debianu. Pro jednodušší instalaci OS je možné využít nástroje NOOBS, který umožní nainstalovat také Ubuntu, OSMC, Windows 10 IoT a další. [2]

Model	RPi 3 A+	RPi 3 B	RPi Zero W
CPU	ARM Cortex-A53 (64bit) 4x 1,4 GHz	ARM Cortex-A53 (64bit) 4x 1,2 GHz	ARM1176JZF-S (32bit) 1x 1 GHz
RAM	512 MB	1 GB	512 MB
USB	4x USB 2.0	4x USB 2.0	1x microUSB
Síť	WiFi 802.11ac, Bluetooth 4.2 BLE	10/100 Mbit/s Ethernet, WiFi 802.11n, Bluetooth 4.1	WiFi 802.11n, Bluetooth 4.1
Napájení	5V micro USB	5V micro USB	5V micro USB

Tabulka 3.1: Přehled vybraných variant Raspberry Pi s WiFi [2]



Obrázek 3.5: Raspberry Pi 3A+ a Raspberry Pi Zero W [2]

3.3 Protokol MQTT

MQ Telemetry Transport[3] byl navržen jako jednoduchý M2M/IoT komunikační protokol, vhodný pro malá zařízení s nízkou spotřebou a malou propustností sítě. Přenos probíhá pomocí TCP/IP a je založen na předávání zpráv mezi klienty prostřednictvím centrálního serveru(tzv. broker).

Broker se stará o přijímání zpráv od poskytovatelů(tzv. publisher) a ty následně předává k přečtení jednomu či více čtenářům(tzv. subscriber), kteří o ně mají zájem. Jeden broker může mít více čtenářů a poskytovatelů zpráv, a přitom čtenářům předává pouze zprávy, ke

[3] http://mqtt.org/

kterým se každý čtenář přihlásil k odběru. V praxi jsou publisher i subscriber tvořeni MQTT klienty, kteří jsou připojeni k MQTT serveru, skrze který poté komunikují. [23, 17, 5]

Zprávy MQTT jsou publikovány přes `topicy`, které není potřeba nějak konfigurovat. S `topicy` se pracuje jako s cestami v souborovém systému, kde je uspořádání témat umožněno díky možnosti oddělení jednotlivých úrovní hierarchie pomocí lomítka.

- sensors/JMENO_POCITACE/temperature/NAZEV_ZARIZENI

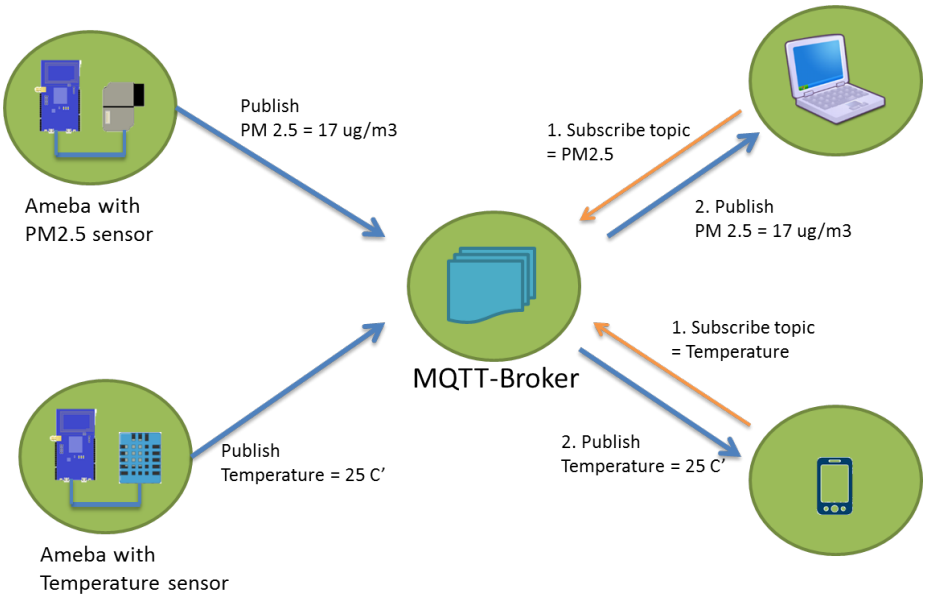
Klienti mohou přijímat zprávy z určitého topicu přihlášením k odběru. Takové přihlášení je možné vykonat buď přihlášením ke konkrétnímu topicu, kdy klient obdrží pouze zprávy publikované na daném topicu, nebo se může přihlásit k odběru s použitím `wildcards`. K použití jsou dostupné dvě varianty wildcards a to `+` a `#`. Topicy s nulovou délkou úrovní jsou též validní, ale způsob jejich chování nemusí být na první pohled zřejmý.

`+` může být použito pro jednu úroveň hierarchie

- sensors/+/temperature/+

`#` může být použito pro všechny zbývající úrovně hierarchie, musí být posledním znakem

- `#`
- sensors/`#`
- sensors/+/`#`
- sensors/+/temperature/`#`



Obrázek 3.6: Schéma přenosu MQTT [23]

MQTT též obsahuje Quality of Service, která je rozdělena na 3 úrovně podle toho jak moc se broker či klient snaží zajistit, že bude zpráva doručena:

0 - „Nanejvýš jednou“ - Broker či klient se pokusí doručit zprávu jedenkrát, bez potvrzení. Tohoto způsobu se využije například pokud posíláme hodnoty ze senzorů a nevadí nám, když se některá zpráva nedoručí. jedná se tedy o nedůležitá nebo často odesílaná data.

1 - „Alespoň jednou“ - Broker či klient se pokusí doručit zprávu alespoň jedenkrát, s potvrzením. Zprávu je tedy nutné doručit, ale nevadí nám, že některá zpráva může být doručena vícekrát.

2 - „Přesně jednou“ - Broker či klient doručí zprávu přesně jednou s použitím 4-way handshake[4]. Použije se v případě, kdy je potřeba zaručit doručení a větší množství stejných zpráv by bylo na škodu.

U všech zpráv je možné nastavit parametr `retained` k jejímu uchování. V případě, že je parametr k uchování zpráv nastaven na `true`, broker uloží poslední odeslanou zprávu a korespondující QoS pro daný topic. V okamžiku přihlášení nového odběratele topicu uchované zprávy mu bude daná zpráva odeslána. Broker tedy ukládá pouze jednu zprávu pro každý topic. Tento typ zpráv pomáhá k aktualizaci statusu nového odběratele, ihned po přihlášení k odběru, v případě že jsou zprávy odesílány nepravidelně a po delší době.

Pokud chce klient zanechat zprávu v případě neočekávaného odpojení, informuje při připojení brokera, že má „Last Will“. Broker v takové situaci odešle zprávu na stejný topic, QoS a retain status jako ostatní zprávy. [5, 1]

3.4 Petriho sítě

Petriho sítě v angličtině Petri Nets je označení široké třídy diskrétních matematických modelů. Pomocí nich jsme schopni specifickými prostředky popsat řídící toky a informační závislosti uvnitř modelovaných systémů. Petriho sítě byly vytvořeny roku 1962, německým matematikem C. A. Petrim v jeho disertační práci „Kommunikation mit Automaten“ a vznikly rozšířením modelovacích schopností konečných automatů. Základními prvky sítě jsou stavy reprezentovány místy (◯), a události mezi nimi reprezentovány přechody (□) se vstupními a výstupními místy. Stav systému je reprezentován značením (•) v místech. Kapacita míst může být neomezená nebo omezená celočíselnou nezápornou hodnotou. Pro zjednodušení grafů sítě se místo násobných hran používá celočíselné ohodnocení hrany, určující počet odebíraných či přidávaných značek.

Síť

Síť lze formálně definovat jako trojici $N = (P, T, F)$ [25], jestliže

- P a T jsou disjunktní množiny
- $F \subseteq (P \times T) \cup (T \times P)$ je binární relace

Množina P značí množinu `míst`(Places), množina T značí množinu `přechodů`(Transitions) a množina F značí `tokovou relaci`(Flow relation) sítě N.

[4] https://standards.ieee.org/standard/802_11i-2004.html

P/T Petriho síť

P/T Petriho síť (Places/Transitions PN), je definována jako šestice $N = (P, T, F, W, K, M_0)$, jestliže [25]

- trojice (P, T, F) je konečná síť
- $W : F \Rightarrow \mathbb{N} \setminus \{0\}$ je ohodnocení hran grafu určující kladnou `váhu` každé hrany
- $K : P \Rightarrow \mathbb{N} \cup \{\omega\}$ je zobrazení určující `kapacitu`(i neomezenou) každého místa
- $M_0 : P \Rightarrow \mathbb{N} \cup \{\omega\}$ je `počáteční značení` míst respektující kapacitu míst $\forall p \in P : M_0(p) \leq K(p)$

Časové Petriho sítě - Time PN

Všechny změny v Petriho sítích, které nepracují s časem jsou provedeny okamžitě. Pro potřeby simulace, řízení, real-time a stochastické analýzy je však potřeba zavést do Petriho sítí práci s časem. V případě, že potřebujeme, aby změny trvaly určitou dobu a čas zavedený v síti nemáme, lze jej do různých variant Petriho sítí zavést také dodatečně. [18, 14]

Přiřazení časů může být provedeno:

- deterministicky (přiřazené časy jsou konstanty) - Časované Petriho sítě
- stochasticky (přiřazené časy jsou náhodné) - Stochastické Petriho sítě
- kombinovaným způsobem (přiřazené časy jsou pro některé přechody konstanty a pro jiné realizacemi náhodných veličin) - Zobecněné stochastické Petriho sítě

Zavedení času může být spojeno s:

- přechody (T-timed PN) - provedení přechodu trvá určitou dobu, po kterou token pobývá uvnitř přechodu
- místy (P-timed PN) - token pobývá stanovenou dobu ve vstupním místě přechodu, jež má být proveden.
- hranami (A-timed PN) - přesun tokenu po příslušné hraně trvá určitou dobu
- tokeny (Token timed PN) - provádění přechodů v síti je sice okamžité, ale tokeny opouštějící příslušný přechod jsou opatřeny časovým razítkem (time stamp), které udává, kdy může být daný token znovu použit

3.5 Vysokoúrovňové Petriho sítě

Z teoretického hlediska lze s využitím klasických Petriho sítí modelovat cokoliv, co lze vyjádřit pomocí algoritmu. V případě potřeby podrobného modelu, kdy se nelze spokojit s hrubou abstrakcí, však představují příliš nízkoúrovňový model. Jednoduché věci jako vyhodnocení aritmetického výrazu pak jsou modelovány příliš složitě. Analogií by mohlo být programování konstrukcí programovacích jazyků pomocí instrukcí procesoru. Právě proto nejsou tyto sítě ideálním prostředkem pro modelování reálných systémů. [14]

Multimnožina

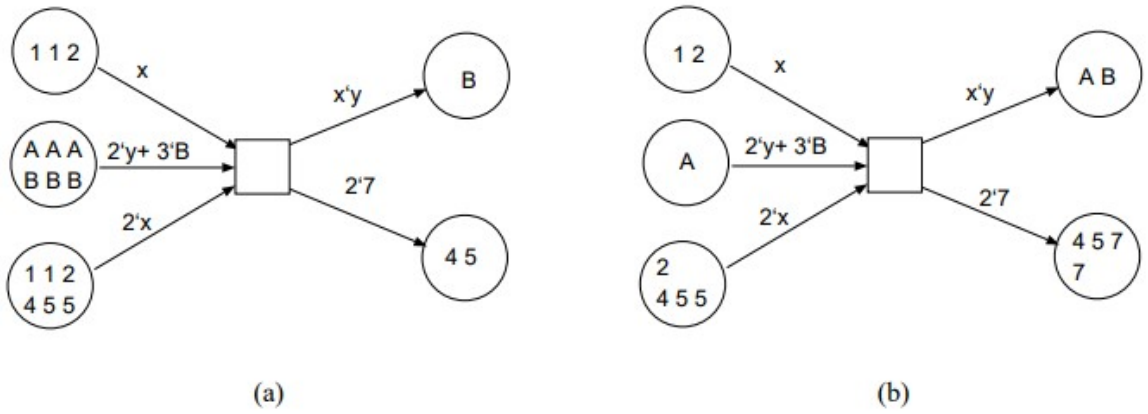
Pro vysvětlení vysokoúrovňových sítí je vhodné také zmínit pojem multimnožiny. Jde o zobecnění množiny. Nebo v případě množiny ji lze chápat jako speciální případ podmnožiny. Rozdíl mezi množinou a multimnožinou je ten, že v multimnožina připouští vícenásobný výskyt jednoho prvku na rozdíl od množiny, která to neumožňuje. [25]

Význam symbolů $\in$, $\subset$ a $\subseteq$ je v případě multimnožin obdobný jako v případě množin:

- $a \in A$ ... v multimnožině A se vyskytuje prvek a alespoň jedenkrát
- $a \subseteq B$ ... všechny prvky multimnožiny A jsou také obsaženy v multimnožině B a to ve stejném, nebo větším počtu výskytů
- $a \subset B$ ... jako případě množin, tedy $A \subseteq B \wedge A \neq B$.

Základní koncept

Značky ve vysokoúrovňových sítích mohou reprezentovat libovolná data, která mohou být prováděním přechodů libovolně zpracovávána. Hranové výrazy specifikují multimnožiny značek. Mohou tedy obsahovat i proměnné, které musí být při provádění přechodu navázány na konkrétní hodnoty. Přechod navíc může obsahovat i strážní podmínku, jejíž splnění je nutné pro jeho provedení. Přechod může též obsahovat akci, která provede libovolný výpočet nad vstupními daty, jehož výsledek poté může být uložen do výstupního místa. Na obr. 3.7 se nachází příklad přechodu vysokoúrovňové Petriho sítě bez strážní podmínky a akce. Obsahuje dvě proměnné x a y a je proveditelný pro dvě různá navázání proměnných $\{x = 1, y = A\}$ a $\{x = 1, y = B\}$. [15]



Obrázek 3.7: Provedení přechodu vysokoúrovňové Petriho sítě po navázaní proměnných {x=1, y=A}; (a) před provedením, (b) po provedení [15]

Kapitola 4

Analýza a návrh systému

V této kapitole budou definovány cíle a návrh postupu k jejich dosažení. Popisuje návrh systému sloužícího k měření teploty ve více místnostech a následným využitím získaných hodnot k řízení vytápění domácnosti, který bude komunikovat se systémem pro automatizaci domácnosti.

Cílem této práce je navrhnout a následně vytvořit systém umožňující interpretovat vysokoúrovňové Petriho sítě s využitím interpretovaného jazyka. S použitím vizuální aplikace bude možné navrhnout a vytvořit Petriho síť, která bude následně exportována do souboru. Ze získaného souboru se vyexportovaná Petriho síť vhodným způsobem převede na kód pro komponenty systému. Následně bude provedeno otestování funkčnosti interpretu v navrženém systému. Tento systém bude používat komunikační protokol kompatibilní se systémem MPOS, který vycházel ze systému PNOS. [12]

4.1 Analýza

Pro naprogramování systému obsahujícího interpret vysokoúrovňových Petriho sítí bude použit programovací jazyk Python 3.5, jehož výhodou je jednodušší údržba a možnost rozšiřitelnosti. Díky použití jazyka Python bude moci být výsledný systém přenositelný, výjimkou bude kód obsluhující hardwarovou část a rozšiřitelný za běhu. Pro práci s Petriho sítěmi bude použita knihovna SNAKES 0.9.26 (kapitola 4.2), která umožňuje definovat a provádět různé druhy Petriho sítí. Bohužel se použitím této knihovny dostáváme do situace, kdy není možné použít levné a úsporné zařízení ESP32 s WiFi a Bluetooth (kapitola 3.2), které by bylo vhodné např. pro čtení ze senzorů. Kvůli omezením jazyka MicroPython, což je zeštíhlený jazyk Python 3 optimalizovaný pro chod na úsporných zařízeních jako jsou Pyboard, ESP8266, ESP32 a nedostatku paměti není možné knihovnu SNAKES na platformě ESP použít. Také z toho důvodu bude pro vývoj a následné použití zvolen jednodeskový počítač Raspberry Pi s podporou jazyka Python 3.

Pro tvorbu Petriho sítí a jejich následný export do souboru bude použit multiplatformní vizuální editor a simulátor Renew 2.5 (kapitola 4.2), který je napsán v jazyce Java. Kvůli interpretaci v jazyce Python nebude možné použít kontrolu syntaxe a simulátor. Výstupní soubor z aplikace Renew bude obsahovat navrženou Petriho síť ve formátu kompatibilním s PNML (kapitola 4.2). Následně bude nutné převést formát výstupního souboru na formát kompatibilní s knihovnou SNAKES a kódem pro provádění.

Protože si aplikace běžící na uzlu a dalších uzlech v síti budou muset mezi sebou předávat zprávy bude pro předávání zpráv mezi aplikacemi v rámci systému použit jednoduchý a

nenáročný komunikační protokol MQTT. Jelikož je potřeba použít pro správnou funkčnost také MQTT broker, který se stará o výměnu zpráv, bude pro tuto úlohu použit open-source broker Eclipse Mosquitto, který je navíc taktéž multiplatformní.

Pro správu skrz webový prohlížeč z PC i mobilních zařízení bude sloužit systém pro automatizaci domácnosti Domoticz 4.10. Tento systém bude s vytvořenými systémem komunikovat taktéž prostřednictvím MQTT. Tato funkčnost se v systému Domoticz standardně nenachází, ale je možné ji přidat dodatečnou instalací doplňku a bude tedy do systému přidána.

4.2 Navrhované prostředky

Renew

V nezkrácené podobě The Reference Net Workshop[1] je multiplatformní open-source simulátor vysokoúrovňových Petriho sítí napsaný v jazyce Java. Zároveň se jedná o vývojové prostředí tohoto typu sítí. Umožňuje tedy modelovat, vizualizovat a simulovat různé druhy Petriho sítí. Jednou z variant formátů pro export, který je vhodný pro naše využití je PNML-kompatibilní formát, který nemusí být kompatibilní se současným standardem. [11] Díky použití jazyka Java se jedná o multiplatformní nástroj. Tento nástroj bude využíván pro modelování Petriho sítí a jejich následný export do souboru ve formátu PNML RefNet.

PNML

Neboli Petri Net Markup Language[2] je návrhem textového formátu sloužícího k zápisu Petriho sítí. Tento formát byl vytvořen k usnadnění přenosu výstupů mezi různými nástroji pracujícími s Petriho sítěmi. Jeho syntaxe je založena na XML formátu.

Kvůli otevřenosti se rozlišuje mezi obecnými vlastnostmi a specifickými vlastnostmi specifických Petriho sítí. Tudíž se zavádí samostatný PNTD (Petri Net Type Definition) pro každý typ sítě, kde se provádí deklarace jmenného prostoru.

SNAKES

SNAKES[3] je knihovna napsaná v jazyce Python, poskytující vše potřebné k definici a provádění mnoha typů Petriho sítí. Jejím hlavním cílem je poskytnout všeobecnou knihovnu pro práci s Petriho sítěmi, která je schopna pracovat s většinou modelů Petriho sítí a je nástrojem k rychlému prototypování. Klíčovou funkcí je schopnost použít objekt jazyka Python jako token a možnost použít výrazy jazyka Python jako strážní podmínku nebo použití výrazů na výstupních hranách přechodů. Další významnou funkcí je možnost rozšíření funkčnosti pomocí plugin systému. [19]

[1] http://renew.de/
[2] http://www.pnml.org/index.php
[3] https://snakes.ibisc.univ-evry.fr/

Eclipse Mosquitto

Jedná se o open-source message broker, který navržen k použití v IoT aplikacích. Implementuje MQTT protokol (viz 3.3) ve verzi 3.1 a 3.1.1. Jde o centrální bod, který se stará o výměnu zpráv mezi zařízeními, která mohou být ve dvou režimech subscriber nebo publisher. Tento broker je nenáročný a použitelný na všech typech zařízení, od úsporných jednodeskových počítačů až po servery.

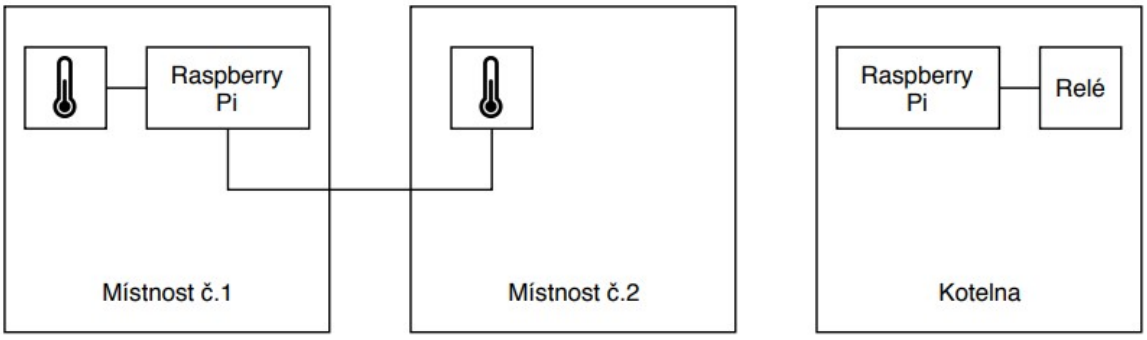
4.3 Návrh systému

Popis zadání

Pro zjednodušení problému máme dvě místnosti s radiátory, přičemž v první z těchto místností se nachází termostat spínající kotel a součástí radiátoru ve druhé místnosti je běžná termostatická hlavice. Pokud v místnosti s termostatem klesne teplota pod požadovanou úroveň, na základě nastavení termostatu sepne kotel. V případě, že klesne teplota pouze ve druhé místnosti, reakce na tuto událost nebude žádná a proto bude v této místnosti nadále chladno, dokud neklesne teplota v první místnosti pod zadanou úroveň.

Návrh řešení

Navrženým řešením tohoto problému je do daných místností umístit programovatelné termostatické hlavice a teplotní čidla, např. DS18B20 nebo DHT22 s Raspberry Pi ideálně s WiFi připojením (Zero W či verzi 3) a sepnout kotel, pokud teplota v první nebo ve druhé místnosti klesne pod požadovanou teplotu zadanou do systému. Díky použití programovatelných termostatických hlavic můžeme předpokládat, že nedojde k přetopení žádné ze sledovaných místností. Pokud při zavedení tohoto systému dojde k poklesu teploty v první místnosti pod požadovanou úroveň, kotel sepne a totéž se provede i pokud poklesne teplota pod požadovanou úroveň pouze v místnosti druhé, nebo v obou místnostech zároveň. Zároveň můžeme měnit požadované teploty v jedné místnosti nezávisle na druhé.



Obrázek 4.1: Návrh rozmístění [23]

4.4 Komunikační protokol

Při návrhu se vycházelo z protokolu použitého v MPOS. Posílaná zpráva je rozdělena na tři části, které jsou odděleny znakem ",". Celková zpráva je obalena složenými závorkami a taktéž její podčásti, které se skládají z více částí. [12]

První část Určuje zda se jedná o zprávu odeslanou ve formě unicast nebo broadcast. V případě, že se v této části nachází "." je zpráva typu unicast a je tedy určena pro konkrétní uzel, pokud se zde nachází "*" nemá zpráva určena konkrétního příjemce, je tedy typu broadcast a zpracována všemi uzly. Většina zpráv je posílána v režimu unicast. V režimu broadcast jsou zasílány pouze servisní zprávy, chybová hlášení nebo zprávy informující ostatní o přihlášení nového uzlu.

Druhá část Má dvě varianty:

a) V případě, že se jedná o zprávu typu unicast, obsahuje druhá část údaje cílového uzlu skládající se ze tří prvků. První část značí IP adresu či název uzlu, druhá cílovou aplikaci nebo příkaz a v případě, že se jedná o aplikaci, je nutné rozšířit tuto část o třetí prvek, kterým je port.

b) V případě, že zpráva nemá zadaný konkrétní uzel (první část tedy obsahuje "*"), obsahuje druhá část údaje o odesílateli. V její první části je IP adresa odesílatele a ve druhé se nachází typ zprávy.

Třetí část Obsahuje samotné tělo zprávy. V případě servisní zprávy je formát pevně daný, jinak je možné definovat formát libovolně.

```
{".",{"192.168.0.8","server","temp1"},{"21.5"}}
```

Př. 4.1: Odeslání obsahu zprávy `21.2` na port `temp1` do aplikace `server` na zařízení s IP adresou `192.168.0.8`.

Servisní zprávy

V této části budou popsány všechny zprávy, které lze použít ke komunikaci se systémem, ale neslouží ke komunikaci mezi jednotlivými aplikacemi. Na základě kombinace povelu a obsahu zprávy, lze do zařízení nahrávat nové aplikace v Pythonu nebo PNML, smazat uložené aplikace, spouštět nebo zastavovat uložené aplikace, přidávat nebo odebírat záznamy ze směrovací tabulky, vypsat směrovací tabulku, vypsat zprávu s informacemi o zařízení a jeho aplikacích, restartovat systém, zobrazit dobu běhu zařízení nebo se dotázat všech zařízení zda jsou aktivní. Jednotlivé příkazy vypadají následovně:

load (příkaz 4.2) Zasláním zprávy typu `load` se nahraje aplikace napsaná v jazyce Python do konkrétního zařízení. Tělo zprávy musí obsahovat zdrojový kód v zadaném formátu (viz 5.2).

```
{".",{"192.168.0.8","load"},{"""zdrojový_kód"""}}
```

Př. 4.2: Nahrání nové aplikace odesláním zprávy s jedním parametrem obsahujícím zdrojový kód.

loadpnml (příkaz 4.3) Zasláním zprávy typu `loadpnml` se nahraje aplikace popsaná Petriho sítí ve formátu PNML do konkrétního zařízení, kde je následně provedena konverze na aplikaci v jazyce Python. S výslednou aplikací lze následně pracovat stejně jako s aplikací nahranou pomocí zprávy `load`. První část těla zprávy musí být kompatibilní Petriho síť (viz 5.3) ve formátu PNML-RefNet, vygenerovaná např. z aplikace Renew (kapitola 4.2), druhá část obsahuje název nově vytvářeného souboru na zařízení.

```
{".",{"192.168.0.8","loadpnml"},{"""pnml_kód""", "název_souboru"}}
```

Př. 4.3: Nahrání nové aplikace odesláním zprávy s jedním parametrem obsahující popis Petriho sítě ve formátu PNML-RefNet.

unload (příkaz 4.4) Zpráva typu `unload` odstraní odpovídající zdrojový kód aplikace ze zařízení. Tělo zprávy obsahuje název aplikace k odstranění ze zařízení.

```
{".",{"192.168.0.8","unload"},{"název_aplikace"}}
```

Př. 4.4: Odstranění zdrojového kódu aplikace.

activate (příkaz 4.5) Po nahrání aplikace zprávou typu `load` nebo `loadpnml` je možné tuto aplikaci spustit. Jedna aplikace může být spuštěna na jednom zařízení i vícekrát pod různými názvy. Zasláním zprávy typu `activate` se spustí aplikace se zadaným zdrojovým kódem a s požadovaným názvem. Tělo zprávy obsahuje název zdrojového kódu a název aplikace, pod kterým bude zdrojový kód spuštěn.

```
{".",{"192.168.0.8","activate"},{"název_zdroj_kódu","název_aplikace"}}
```

Př. 4.5: Spuštění zdrojového kódu aplikace s požadovaným názvem.

deactivate (příkaz 4.6) Zasláním zprávy typu `deactivate` dojde k deaktivaci běžící aplikace na daném zařízení. Tělo zprávy obsahuje název deaktivované aplikace.

```
{".",{"192.168.0.8","deactivate"},{"název_aplikace"}}
```

Př. 4.6: Deaktivace běžící aplikace se zadaným názvem.

addroute (příkaz 4.7) Zpráva typu `addroute` slouží k přiřazení záznamu do směrovací tabulky. Tělo zprávy obsahuje zdroj a seznam cílů. Zdroj je definován zdrojovou IP adresou/názvem, názvem aplikace a portem, cíle jsou definovány cílovou IP adresou/názvem, názvem aplikace a portem.

```
{".",{"192.168.0.8","addroute"},{{{"zdroj_IP","cíl_apl","zdroj_port"},{
                    {"cíl_IP","cíl_apl","cíl_port"}}}}}
```

Př. 4.7: Přidání záznamu do směrovací tabulky.

removeroute (příkaz 4.8) Použitím zprávy typu `removeroute` dojde k odstranění odpovídajícího záznamu ze směrovací tabulky. Tělo zprávy obsahuje zdroj a seznam cílů. Zdroj je definován zdrojovou IP adresou/názvem, názvem aplikace a portem, cíle jsou definovány cílovou IP adresou/názvem, názvem aplikace a portem.

```
{".",{"192.168.0.8","removeroute"},{{{"zdroj_IP","cíl_apl","zdroj_port"},{
    {"cíl_IP","cíl_apl","cíl_port"}}}}}
```

Př. 4.8: Odstranění záznamu do směrovací tabulky.

getroutes (příkaz 4.9) Zpráva typu `getroutes` složí k výpisu směrovací tabulky na zařízení. Tělo může být prázdné.

```
{".",{"192.168.0.8","getroutes"},{}}
```

Př. 4.9: Výpis záznamů ze směrovací tabulky.

status (příklad 4.10) Zasláním zprávy typu `status` dojde k výpisu informací o zařízení, obsahující IP adresu a název zařízení, platformu, dobu běhu, volnou paměť RAM[MB](v případě Raspberry Pi), běžící a nainstalované aplikace. Tělo může být prázdné.

```
{".",{"192.168.0.8","status"},{}}
```

Př. 4.10: Zpráva vyžadující výpis informací o zařízení.

restart (příklad 4.11) Zaslání zprávy typu `restart` způsobí v případě úspěšného přijetí restart cílového systému. Tělo může být prázdné.

```
{".",{"192.168.0.8","restart"},{}}
```

Př. 4.11: Zpráva vyvolávající restart systému.

runningtime (příkaz 4.12) Zpráva typu `runningtime` vyžádá od systému výpis doby jeho běhu. Tělo může být prázdné.

```
{".",{"192.168.0.8","runningtime"},{}}
```

Př. 4.12: Zpráva vyžadující informaci o době běhu systému.

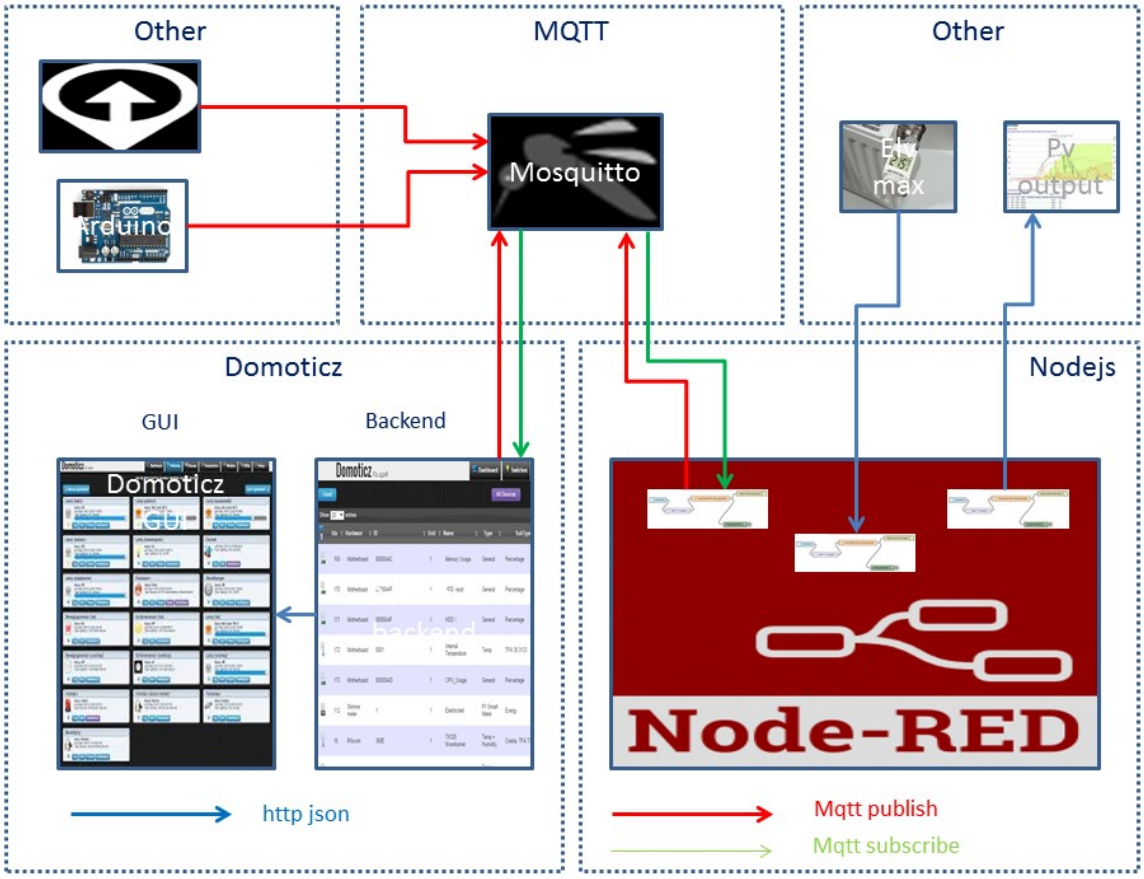
discover (příkaz 4.13) Zasláním zprávy typu `discover` dojde k odeslání odpovědi od všech běžících zařízení s informací o názvu, IP adrese a době běhu. Vhodná pro monitoring.

```
{"*",{"192.168.0.8","discover"},{}}
```

Př. 4.13: Zpráva vyžadující odpověď od všech běžících zařízení v síti.

4.5 Dashboard

K vizualizaci a ovládání bude použit systém pro domácí automatizaci Domoticz (kapitola 3.1). Po instalaci rozšíření[4] pro podporu MQTT, bude možné komunikovat prostřednictvím MQTT zpráv a využívat tyto zprávy k zasílání informací o aktuální teplotě nebo stavu vytápění a případných dalších potřebných informacích. Komunikace do a z Domoticz systému využívá k přenosu specifický zápis v JSONu. Výchozími topicy pro zasílání a přijímání zpráv jsou `domoticz/in` a `domoticz/out`. Dále toto rozšíření obsahuje Node-RED, což je alternativní nástroj pro vytváření malých programů(flows) k propojení se zařízeními, ale jeho použití není povinné. Pro posílání upozornění lze vytvořit prostřednictvím aplikace nebo emailu lze nadefinovat vlastní skripty.



Obrázek 4.2: Domoticz MQTT architektura[4]

[4]https://www.domoticz.com/wiki/MQTT#Installing_Node.JS

Kapitola 5

Implementace a Testování

Implementace a testování probíhaly na zařízeních Raspberry Pi 2 v kombinaci s Raspberry Pi 3, v obou případech s operačním systémem Raspbian ve verzi Stretch. Implementačním jazykem je Python 3.5 s knihovnami SNAKES 0.9.26 pro práci s Petriho sítěmi, W1ThermSensor[1] pro čtení ze senzoru DS18B20 a Adafruit_Sensor[2] pro čtení z DHT11 nebo DHT22. Pro návrh Petriho sítí je využívána aplikace Renew 2.5 (kapitola 4.2). Při implementaci systému pro práci s aplikacemi a komunikací mezi aplikacemi a zařízeními se vycházelo z existujícího systému MPOS pro zařízení ESP8266 a ESP32, který umožňuje spouštět pouze aplikace napsané v jazyce MicroPython. [12] Výsledný systém umožňuje za běhu nahrávat, mazat, aktivovat, deaktivovat aplikace, obsluhovat komunikaci mezi aplikacemi a dalšími zařízeními. Systém přijímá aplikace napsané v jazyce Python či aplikace získané převodem z formátu PNML-RefNet, které byly navržené pomocí Petriho sítí v aplikaci Renew. Po aktivaci jsou aplikace převedené z PNML-RefNet formátu interpretovány s využitím knihovny SNAKES, která byla rozšířena o práci s fyzickým časem. Pro usnadnění spouštění systému byly vytvořeny funkce pro nahrání aplikací po startu a jejich aktivaci spolu s nastavením směrovací tabulky.

5.1 Moduly systému

Systém se skládá ze tří modulů system, config a convert. K aktivaci systému dojde spuštěním `system`. Modul system je hlavním z modulů obsluhujícím komunikaci a uživatelské aplikace, jehož spuštěním dojde k načtení modulů `config`, pro ukládání a načítání nastavení ze souboru a `convert` pro převod z Petriho sítí ve formátu PNML-RefNet na uživatelské aplikace v jazyce Python.

Modul system

Základní modul starající se o dynamické nahrávání a běh uživatelských aplikací na zařízení a komunikaci mezi aplikacemi na aktuálním zařízení nebo jiných zařízeních. Pro implementaci klienta MQTT sloužícího ke komunikaci s ostatními zařízeními byla použita knihovna `Eclipse Paho`[3]. Dynamické nahrání nových aplikací se provádí přijetím zprávy s předmětem `load` nebo `loadpnml` s následným převedením do jazyka Python. Zpráva s předmětem `load` přijímá zdrojový kód aplikace v definovaném formátu (kapitola 5.2) a zpráva typu

[1] https://github.com/timofurrer/w1thermsensor
[2] https://github.com/adafruit/Adafruit_Sensor
[3] https://www.eclipse.org/paho/

`loadpnml` přijímá kompatibilní Petriho síť ve formátu PNML-RefNet (kapitola 5.3), kterou převede pomocí modulu `convert` do formátu aplikace systému a uloží do souboru do složky `pnml`. Po načtení zdrojového kódu aplikace ze zprávy, respektive souboru (při příkazu loadpnml) ve formě řetězce se vykonáním příkazu `exec(code,globals(),tasks)` uloží třída z proměnné `code` do asociativního pole `tasks` a vytvoří se soubor aplikace s příponou `".py"` ve složce `apps`. Přijetím zprávy s předmětem `activate` se provede vytvoření objektu aplikace a její spuštění. Aby aplikace mohly běžet paralelně, provede se v jejich konstruktoru vytvoření nového vlákna pomocí standardní knihovny `threading`, které zavolá metodu `loop` obsahující hlavní cyklus aplikace. Pro interpretaci Petriho sítí je využívána knihovna `SNAKES` (kapitola 4.2), která byla rozšířena o podporu fyzického času na výstupních hranách.

Po spuštění proběhne načtení souboru `settings.json` pomocí modulu `config` a v případě úspěšného načtení údajů dojde k pokusu o připojení k MQTT brokeru. Po úspěšném připojení jsou příchozí zprávy zpracovávány pomocí callbacku `on_message`. Následně dojde k načtení uložených aplikací ve složce `apps` a k načtení zpráv s předměty `activate` a `addroute` uložených v souboru `routing.txt`, pokud tento soubor existuje. V případě úspěšné inicializace se odešle zpráva o vytvoření nového uzlu s názvem, IP adresou, platformou, dobou běhu v sekundách a nahranými aplikacemi (příkaz 5.1). Nakonec je zavolána funkce `loop`, což je nekonečný cyklus programu se zpracováním odchozích zpráv z uživatelských aplikací. Modul obsahuje třídy Route, Point, Message a Task. Třídy Route a Point slouží k popisu směrovací tabulky a zdrojů nebo cílů zprávy. Třída Task popisuje formát uživatelských aplikací a metody potřebné k jejich nastavení a komunikaci a třída Message reprezentuje odesílané zprávy z aplikací. Metody `__repr__` tříd Route a Point byly přepracovány, aby vyhovovaly formátu odesílané zprávy.

```
{"*",{"192.168.1.8","new_node",{{"node_name","rpi_obyvak"}
,{"platform","rpi"} ,{"IP":"192.168.1.8"}, {"running_time","35"},
                {"installed",{"SensorPN","OrPN"}}}}}
```

Př. 5.1: Zpráva odeslaná po spuštění nového uzlu

Odchozí zprávy

Odchozí zprávy aplikací jsou odesílány v nekonečném cyklu systému. Rozlišují se odesílané zprávy typu broadcast, tedy určené všem zařízením a unicast, určené jednomu konkrétnímu zařízení. Odesílané zprávy typu broadcast jsou označeny symbolem `"*"` na první pozici a zprávy určené jednomu příjemci obsahují taktéž na první pozici symbol `"."`. Všechny zprávy odesílané aplikacemi jsou zpracovány funkcí `sendMsg` s jedním parametrem, kterým je obsah posílané zprávy. V rámci této funkce se nahlédne do směrovací tabulky, aby se zjistilo zda je zpráva odesílána lokálně nebo po síti. V případě nenalezení záznamu se zpráva odešle pouze, je-li nastaven parametr `broadcast_everything` na `1`. Pokud je zpráva odesílána lokálně předá se obsah zprávy příslušné aplikaci. V případě posílání po síti se zkontroluje je-li nastaven parametr `broadcast_everything`. Standardně se zprávy odesílají příjemci zjištěnému ze směrovací tabulky, pouze v případě kdy je parametr `broadcast_everything` nastaven na hodnotu `1` se odesílají všechny zprávy také v režimu broadcast. Systém standardně na zprávy typu broadcast nereaguje, jedinou výjimkou je systémová zpráva s předmětem `discover`.

Příchozí zprávy

Při příchodu nové zprávy přes MQTT je zavolán callback `on_message`, který převede zprávu do kódování utf-8 a zavolá funkci `decodeMsg` pro zjištění předmětu zprávy a její zpracování. Systém MPOS nepodporuje posílání složených závorek uvnitř zpráv, protože všechny složené závorky po přijetí nahrazuje za hranaté, kvůli dalšímu zpracování zprávy. Při přijetí obsahu se složenými závorkami vzniká problém, protože jsou tyto závorky taktéž nahrazeny a to způsobuje zpracování odlišného obsahu zprávy, než jaký byl odeslán. Proto je nutné dodržovat formát zpráv a v případě přijetí obsahu obaleného třemi uvozovkami `'"""'` značícího víceřádkový řetězec, nedojde uvnitř řetězce k nahrazení závorek. Nad zpracovaným textem je následně zavolána funkce `eval`, sloužící k převedení řetězce na pole obsahující zprávu. V případě chyby se vyvolá výjimka, která odešle broadcast chybovou zprávu s předmětem `message_format` a popisem chyby. V případě přijaté zprávy typu broadcast, reaguje systém pouze na zprávu s předmětem `discover`(příkaz 5.2). Systém na přijatou zprávu s předmětem `discover` odpovídá zprávou typu unicast obsahující název, IP adresu a dobu běhu v sekundách.

```
{".",{"192.168.0.8","discover"} ,{"rpi_loznice","192.168.1.5","105"}}
```

Př. 5.2: Odpověď na zprávu discover

Pokud je přijata zpráva typu unicast, zkontroluje se, zda sedí IP adresa příjemce nebo název příjemce v případě nastaveného parametru `route_by_name` a teprve poté, začne být zkoumán předmět a obsah zprávy. Systém reaguje na servisní zprávy typu unicast s předmětem status (příkaz 5.3), load, loadpnml, unload, activate, deactivate, addroute, removeroute, getroutes, restart nebo runningtime. Odpověď status obsahuje název uzlu, platformu, IP adresu, dobu běhu[s], volnou paměť RAM[MB], běžící aplikace a nahrané aplikace.

```
{"*", {"192.168.0.8", "status"}, {{"node_name", "rpi_obyvak"}, {"platform",
   "rpi"}, {"IP", "192.168.0.8"}, {"running_time", "122412"}, {"memory",
 "703"}, {"running", {"sensor1","sensor2","or"}}, {"installed", {"Sensor",
                                    "Or"}}}}
```

Př. 5.3: Odpověď na zprávu status

Na ostatní zprávy odpovídá systém zprávou typu broadcast s nezměněným předmětem zprávou OK (příkaz 5.4) nebo Error (příkaz 5.5). V případě, že je předmětem zprávy název běžící uživatelské aplikace, systém příslušné aplikaci předá obsah zprávy na odpovídající port získaný ze směrovací tabulky.

```
{"*", {"192.168.0.151", "load"}, {"OK"}}
```

Př. 5.4: Kladná odpověď na zprávu load

```
{"*", {"192.168.0.151", "unload"},{"Error", "Not loaded"}}
```

Př. 5.5: Odpověď s chybou na zprávu unload

Modul config

Tento modul se spouští z modulu `system`. Používá knihovnu `json` pro práci s formátem JSON. Tvoří rozhraní pro práci se souborem `settings.json`, do kterého se ukládají informace o připojení k MQTT brokeru a data ze zařízení. Po spuštění se vytvoří v případě neexistence soubor (příkaz 5.6) ve formátu JSON, v případě již existujícího souboru se načtou data do asociativního pole pro usnadnění další práce se souborem. Aplikace zároveň mohou používat tento soubor pro ukládání hodnot, a mohou přepisovat veškeré uložené hodnoty. Pro oddělení dat aplikace od ostatních hodnot, lze při volání funkcí pro práci se souborem přidat parametr s názvem aplikace, čímž se použije JSON objekt s názvem aplikace. Pro načtení nejdůležitějších hodnot potřebných k uskutečnění komunikace slouží předdefinované názvy parametrů. Parametr `node_name` značí název zařízení, kterým lze nahradit IP adresu v komunikaci mezi zařízeními v případě použití `route_by_name`, `platform` značí použitou platformu. Parametry `mqtt_broker` a `topic` je nutné v případě potřeby upravit ručně a přidáním parametrů `username` a `password` bude systém k MQTT připojení používat také uživatelské jméno a heslo.

```
{{"node_name":"rpi_name"},{"platform":"rpi"},{"mqtt_broker":"192.168.0.8"},
                                {"topic":"pn"}}}
```

Př. 5.6: Obsah souboru `settings.json` vytvořeného po spuštění

Modul convert

Modul, který má za úlohu provádět převod (kapitola 5.4) Petriho sítí kompatibilních se systémem (kapitola 5.3) z PNML-RefNet formátu, na aplikace systému v jazyce Python, které provádějí převedenou Petriho síť. Převod je spuštěn při přijetí zprávy s předmětem `loadpnml`. Obsahuje třídy Declaration, Place, Transition a Arc, které jsou využity pro vnitřní reprezentaci Petriho sítě. Pro načtení PNML-RefNet formátu je použito XML API `xml.etree.ElementTree`.

5.2 Uživatelské aplikace

Formát aplikací je popsán třídou `Task`, ze které nově vytvořené aplikace musí dědit vlastnosti. Pro vytvoření a nahrání uživatelské aplikace je tedy potřeba, aby obsahovala funkci loop s cyklem a aby dědila z třídy Task. Aplikaci jsou poté přiřazeny parametry, kterých může být libovolné množství. Parametry jsou přiřazeny z aktivační zprávy prostřednictvím pole `self.params`, přičemž první parametr je vždy název aplikace (bez přípony) a druhý parametr značí název aktuální instance. Stejnou aplikaci lze spustit na jednom zařízení i vícekrát, jedinou podmínkou je aktivace aplikace s jiným názvem instance. Pro inicializaci atributů může být ve vytvořené aplikaci definována metoda `init`, která je zavolána pouze jednou stejně jako část před a za cyklem v metodě `loop`. Metoda `loop` pro udržení běhu programu je povinná a obsahuje cyklus programu jehož běh je podmíněný atributem self._running. Aplikaci je možné zastavit pomocí metody `stop`, která je zavolána systémem po přijetí zprávy s předmětem `deactivate`, čímž se změní hodnota atributu `self._running` na hodnotu `False`. Pro odesílání zpráv z aplikací slouží metoda `send` ze třídy Task s parametry obsahující port a obsah zprávy, a pro přijímání je možné nadefinovat vlastní metodu `receive` jejíž implementace je dobrovolná se stejnými parametry.

Po vytvoření aplikace se zavolá její konstruktor s přiřazením parametrů do `self.params`, atribut self._running značící běh aplikace se nastaví na hodnotu `True`, zavolá se metoda `init` pro inicializaci volitelných atributů a spustí se provádění metody `loop` v novém vlákně s názvem instance. Tím se umožní paralelní vykonávání aplikací. Pro ukončení běhu se tedy nastaví tento atribut na `False` a to zavoláním metody `_stop`. Pro odesílání zprávy slouží metoda volaná příkazem `super().send` s parametry obsahujícími cílový port a obsah zprávy. Pro přijetí se používá vlastní metoda `receive` taktéž obsahující parametry port a obsah zprávy. Není-li při odesílání nalezen cíl ve směrovací tabulce a není nastaven parametr `broadcast_everything`, zpráva je zahozena, jinak je odeslána zprávou typu broadcast.

Uživatelské aplikace je možné nahrávat ve formě třídy jazyka Python ze souboru s příponou `".py"`, která dědí ze třídy Task, nebo ve formátu PNML-RefNet ze souboru s příponou `".pnml"`, který systém převede do na kód v jazyce Python v požadovaném formátu aplikace. Pro nahrání aplikace v jazyce Python slouží zpráva s předmětem `load`, jejíž obsah je tvořen zdrojovým kódem aplikace (např. výpis 5.1). U tohoto typu zprávy není potřeba zadávat žádný název, protože se k uložení do souboru použije název třídy s příponou `".py"`. K nahrání aplikace popsané Petriho sítí slouží zpráva s předmětem `loadpnml` a obsahem skládajícím se z PNML-RefNet kódu a názvu vytvářené aplikace. Po přijetí zprávy se kód uloží do složky `pnml` s příponou `.pnml` a následně se spustí konverze, jejímž výstupem je zdrojový kód aplikace, umožňující interpretaci zadané Petriho sítě, uložený do složky `apps` s příponou `".py"`. K odlišení aplikací poslaných zprávou `loadpnml` je vhodné přidat k názvu nové aplikace řetězec `PN` (např. `OrPN`).

```
#[Or, app_name, number_of_inputs]
class Or(Task):
	def init(self):
		self.output = 0
		self.interval = 100
		self.number_of_inputs = 2
		if len(self.params) > 2 and self.params[2].isdigit():
			self.number_of_inputs = int(self.params[2])
		self.inputs = {}

	def loop(self):
		while self._running:
			if len(self.inputs) >= self.number_of_inputs:
				self.output = 0
				for i in self.inputs.values():
					if bool(i):
						self.output = 1
						break
				super().send("output", str(self.output))
				time.sleep(self.interval)
			time.sleep(1)

	def receive(self, port, payload):
		if str(payload).isdigit():
			self.inputs[port] = int(payload)
```

Výpis 5.1: Zdrojový kód uživatelské aplikace Or

Na výpisu 5.1 se nachází zdrojový kód aplikace `Or`, který je totožný s názvem třídy definující aplikaci. Metoda `init` slouží k inicializaci vstupů, výstupu a doby čekání mezi cykly. Po inicializaci se zavolá v novém vlákně metoda `loop` obsahující cyklus s podmínkou `self._running`. Pokud je aplikace aktivovaná provede se po přijetí všech vstupů každých 100 s logický `OR` nad všemi přijatými vstupy. Výstup je pak odeslán na port `output` aplikaci ze směrovací tabulky.

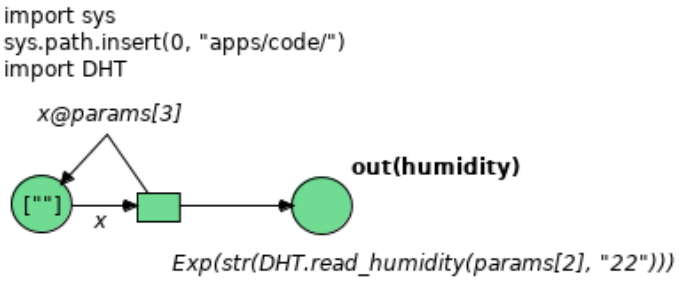
5.3 Návrh Petriho sítí

Pro návrh vysokoúrovňových Petriho sítí popisujících chování komponent systému se využívá aplikace Renew napsaná v jazyce Java. Hlavní snahou bylo neodlišovat se příliš od původně používaného zápisu, ale ne vždy toho bylo možné dosáhnout. Vznikly tedy nějaké odlišnosti, které je nutné dodržovat. Kvůli provedeným změnám a použití jazyka Python není použitelná kontrola syntaxe a simulace.

V rámci návrhu lze vytvářet pouze jednu síť. Vytvářené sítě se skládají z míst, přechodů a hran. Pro návrh těchto prvků lze použít pouze nástroje Place Tool, Transition Tool nebo Arc Tool. K inicializaci míst, popisu hran a strážních podmínek slouží nástroj Inscription Tool, přičemž pro každý objekt může existovat maximálně jeden popis. Pojmenování míst se provádí nástrojem Name Tool. Vstupní a výstupní místa mají předem definovaný tvar jmen, a proto by se názvy v tomto formátu neměly jinde vyskytovat. Místa a přechody by měly mít maximálně jedno pojmenování a každé jméno by mělo být unikátní v rámci sítě. K importu modulů a deklaraci proměnných do Petriho sítě slouží nástroj Declaration Tool. Ostatní nástroje z aplikace Renew nejsou podporovány. Vygenerování sítě do souboru ve formátu PNML-RefNet se používá postup Export, Export current drawing, XML a následně PNML-RefNet current drawing.

Nachází-li se v síti více objektů vytvořených nástrojem Declaration Tool, nelze zaručit pořadí ve kterém bude jejich obsah vykonán, proto je vhodnější mít v síti pouze jeden objekt a jednotlivé příkazy oddělit pomocí klávesy Enter. Toho bylo možné docílit použitím víceřádkového řetězce při zpracování, jinak by bylo potřeba zapsat příkazy na jeden řádek s oddělovačem `"\n"`, nebo zapsat příkazy na více řádků s oddělovačem `"\n\\"` mezi jednotlivými řádky.

Nejviditelnější změnou je použití zápisu pro vstupy a výstupy Petriho sítí. Pro vstupy a výstupy Petriho sítí se používají vybraná místa. Vstupní místa jsou pojmenována ve formátu `in(port)`, kde port je nahrazen za vstupní port aplikace na kterém je očekávána zpráva a výstupní místa jsou pojmenována `out(port)`, kde port je nahrazen za výstupní port aplikace. Při pojmenování místa pouze `out`, bez zadání portu, se použije výchozí port `output`. Navrhovaná síť může obsahovat větší množství vstupních a výstupních míst, ale je možné také nezadat žádné. Znak uvozovky `'"'` lze ve výrazech na rozdíl od znaku apostrofu `"'"` používat bez zpětného lomítka. Problematické je použití speciálních znaků nepodporovaných XML a tedy i PNML-RefNet formátem, např. `<` nebo `>`. Tyto znaky je nutné nahrazovat zápisem `<` nebo `>` apod. z důvodu kompatibility. Kvůli interpretaci navržených Petriho sítí v jazyce Python, je nutné používat ve výrazech strážních podmínek a výstupních přechodů kód taktéž v jazyce Python. Mezi libovolným místem a přechodem se nachází maximálně jedna vstupní hrana a jedna výstupní hrana, přičemž každý přechod má právě jednu vstupní hranu.



Obrázek 5.1: Zjednodušená Petriho síť pro čtení vlhkosti v aplikaci Renew

Při inicializaci míst se používá zápis v podobě neprázdného seznamu, proto je v aplikacích používán zápis `[""]`. Podporovány jsou datové typy jazyka Python rozšířené o typ `BlackToken` s hodnotou `dot` reprezentující token. Pro zápis strážních podmínek se používá notace `guard cond`, kde cond je požadovaná strážní podmínka zapsaná v jazyce Python, v případě negace podmínky se používá zápis `!guard cond`. Na vstupních hranách se musí vyskytovat buď konstanta nebo proměnná a na výstupních hranách konstanta, proměnná, nebo výraz s volitelnou dobou zpoždění taktéž zapsaný v jazyce Python. Doba provádění výstupní hrany se značí symbolem `"@"` a číselnou hodnotou. Konstanty se zadávají ve formě hodnoty, proměnné se zadávají názvem a výrazy se doporučuje zadávat ve formě `Exp(výraz)`, čímž se při konverzi rozezná mezi konstantami, proměnnými a výrazy. V případě problému při rozeznání lze konstanty zadat ve formě `Value(konstanta)` a proměnné `Variable(proměnná)`. Při návrhu v nástroji Renew je možné většinu varování ignorovat, protože se používá odlišná forma zápisu.

Na obrázku 5.1 můžeme vidět Petriho síť pro čtení vlhkosti ze senzoru DHT22. Tato síť přijímá dva parametry, kterými jsou číslo pinu a doba zpoždění hrany. Síť nemá žádné vstupní místo a jedno výstupní místo s názvem `out(humidity)`, hodnoty na výstupu tedy budou odesílány na port humidity. Nástroj Declaration Tool byl použit pouze jednou, i když příklad obsahuje deklaraci s více řádky. Na základě importu se načte modul DHT, který se nachází v adresáři `apps/code/`. Tato síť nemá žádné vstupní místo, protože neobsahuje žádné místo pojmenované ve formátu `in(název_portu)`. Po spuštění ihned započne provádění jediného přechodu, během kterého se provede funkce `read_humidity(params[2], "22")` z modulu DHT, kde první parametr značí pin pro čtení ze senzoru a druhý parametr označuje typ senzoru, kterým je v daném případě DHT22. Výsledek této funkce bude převeden na textový řetězec a odeslán na port s názvem humidity. Po čase získaném z `params[3]` v sekundách od zahájení provádění přechodu bude vstupní token vrácen do počátečního místa a proces provádění se opakuje.

5.4 Konverze

Pro převod ze souboru ve formátu PNML-RefNet do souboru v jazyce Python 3 se používá program `convert.py`. Lze jej spustit s povinnými parametry -r readfile a -w writefile, kde readfile je původní soubor s Petriho sítí ve formátu PNML-RefNet a writefile je výstupní soubor v jazyce Python. Volitelným parametrem je `-l` pro vytvoření samostatně spustitelného kódu s Petriho sítí, tento parametr slouží spíše pro ladící účely a nepodporuje veškerou funkcionalitu. Bez použití parametru -l je výstupem soubor s aplikaci systému převedenou do jazyka Python v požadovaném formátu. Po spuštění s parametrem -h dojde k vypsání nápovědy. Hlavní funkcí tohoto programu je jeho využití pro potřeby systému, který jej

používá při přijetí zprávy s předmětem `loadpnml`, ke konverzi a vytvoření nové aplikace systému interpretující navrženou Petriho síť. Po vykonání převodu dojde k načtení nově získané aplikace do systému a zároveň se vytvoří nový soubor ve složce `apps` s příponou `".py"`, aby při startu nebo restartu systému bylo možné aplikaci opět automaticky načíst. Na následujícím výpisu 5.2 vidíme Petriho síť pro čtení vlhkosti ze senzoru DHT22 ve formátu PNML-RefNet s odstraněnými elementy `type` sloužícími k popisu typu hran a `graphics` popisující rozmístění prvků v aplikaci Renew.

```
<pnml xmlns="RefNet">
   <net id="netId1557328316969" type="RefNet">
      <place id="55">
         <label>
            <text>[""]</text>
         </label>
      </place>
      <place id="56">
         <name>
            <text>out(humidity)</text>
         </name>
      </place>
      <transition id="57">
      </transition>
      <arc id="58" source="55" target="57">
         <label>
            <text>x</text>
         </label>
      </arc>
      <arc id="59" source="57" target="56">
         <label>
            <text>Exp(str(DHT.read_humidity(params[2],"22")))</text>
         </label>
      </arc>
      <arc id="60" source="57" target="55">
         <label>
            <text>x@params[3]</text>
         </label>
      </arc>
      <declaration>
         <text>import sys
sys.path.insert(0,"apps/code/")
import DHT</text>
      </declaration>
      <name>
         <text>Humidity</text>
      </name>
   </net>
</pnml>
```

Výpis 5.2: Zkrácený zápis Petriho sítě pro čtení vlhkosti ve formátu PNML-RefNet

Protože jsou Petriho sítě složeny z míst, hran a přechodů, používá modul convert pro jejich reprezentaci třídy Place, Arc a Transition, pro deklarace obsahuje třídu Declaration. Všechny elementy reprezentující síť jsou vnořeny v elementu `pnml` a následně v `net`. Všechny hodnoty jsou obsaženy ve vlastním elementu `text`. Jako první jsou vyhledány elementy `declaration` obsahující popis deklarace. Poté se zpracovávají elementy `place` a jejich atribut `id`. Název místa se nachází uvnitř `name` a případné hodnoty v `initialMarking` nebo `label`. Následně se zpracovávají elementy `transition` s atributem `id`. Přechod může obsahovat název `name` a hodnotu v `expression`, `inscription`, `label` nebo `guard`. Nakonec se načítají elementy `arc` s atributy `id`, `source` a `target`, které obsahují `type`, který není při převodu použit a konstantu, proměnnou nebo výraz uvnitř elementu `label`. Poté se podle parametru `-l` volí mezi funkcí s výstupem ve formě samostatně spustitelného programu nebo uživatelskou aplikací implementovaného systému.

Získaný text z elementu `declaration` je následně v aplikaci použit jako parametr příkazu `globals.declare`, který slouží k načtení obsahu do kontextu sítě. Pro každé místo se vytvoří objekt `Place`. Jestliže byl zadán název, použije se jako první parametr, jinak se použije načtené id. Druhým parametrem jsou v případě jejich zadání načtené hodnoty. K reprezentaci přechodu slouží třída `Transition`, jehož první parametr je název nebo id jako u vytváření místa a druhý parametr se strážní podmínkou, který je volitelný. Pro reprezentaci hrany slouží objekt `Arc`. Nejprve je třeba určit, zda se jedná o vstupní nebo výstupní hranu. Toho se docílí nalezením shody u atributů `source` a `target` s `id` u míst a přechodů. Pro přidání vstupních a výstupních hran se použijí metody Petriho sítě `add_input` a `add_output`. Parametry `add_input` jsou název místa, název přechodu a konstanta nebo proměnná a u metody `add_output` jsou to název místa, název přechodu, konstanta nebo proměnná nebo výraz a volitelný parametr s dobou provedení.

5.5 Interpretace Petriho sítí

Pro účely práce s Petriho sítěmi v rámci jazyka Python 3.5, byla zvolena knihovna `SNAKES` ve verzi 0.9.26, vzhledem k jejím možnostem práce s Petriho sítěmi, schopnosti volat vlastní zdrojový kód a plugin systému s možností rozšíření její funkčnosti. Knihovna v základu nepodporuje fyzický čas v Petriho sítích a proto bylo přidáno zpoždění na výstupních hranách pro větší praktické využití. Rozšíření je provedeno přidáním souborů `timed.py` a `calendar.py` do složky `plugins` nacházející se ve složce s knihovnou `SNAKES` a provedením importu (výpis 5.3). Soubor `timed.py` obsahuje upravené metody ze souboru knihovny `nets.py` a soubor `calendar.py` obsahuje implementaci kalendáře událostí.

```
import snakes.plugins.timed
snakes.plugins.load(snakes.plugins.timed, "snakes.nets", "tpn")
from tpn import *
```

Výpis 5.3: Import rozšíření knihovny SNAKES

V třídě `Expression` v metodě `bind` bylo zredukováno množství volání hranových výrazů při kontrole proveditelnosti přechodů, které není žádané. Zredukování bylo dosaženo dočasným ukládáním získaných hodnot do asociativního pole, které jsou při opakovaném volání načteny. Tyto hodnoty jsou kvůli aktualizaci hodnot pravidelně mazány po provedení cyklu zavoláním funkce `rmBindings` z běžící aplikace. Pro určení zda se jedná o hranový výraz nebo výraz strážní podmínky se provádí rozpoznání v objektu `Transition`.

Každá aplikace si vytváří vlastní kalendář událostí při inicializaci objektu `PetriNet`, který následně sdílí s objekty reprezentujícími přechody. Kvůli potřebě zadat dobu zpoždění výstupních hran, byl přidán do metod `add_output`, které slouží k přidání výstupních hran do sítě, v třídách `Transition` a `PetriNet` parametr `time`. Jde o parametr udávající dobu zpoždění v sekundách o který se prodlouží předání tokenu do výstupního místa. Při volání metody `fire`, kterou se provádí přechod, se v době po odstranění tokenu ze vstupního místa odloží zavolání metody pro umístění tokenu do výstupního místa. Místo zavolání metody se umístí objekt `Entity`, s hodnotami pro získání výstupního tokenu, do kalendáře událostí a nastaví se provedení na aktuální čas odložený o zadané zpoždění.

Při vytváření nové aplikace ze souboru ve formátu PNML-RefNet se nejprve vytvoří třída aplikace s názvem zadaným jako první parametr ve zprávě s předmětem `loadpnml`, která dědí ze třídy `Task`. Následně se vytvoří metoda `init` (tělo metody 5.4 pro čtení vlhkosti ze senzoru DHT22), která je volána z konstruktoru aplikace a provede se tedy pouze jednou. Metoda `init` slouží k inicializaci nahraných aplikací a není povinná. Na začátku se provede inicializace asociativních polí s názvy vstupních a výstupních míst. Dojde k vytvoření objektu Petriho sítě zavoláním `PetriNet` s jejím názvem. Poté se přeuloží parametry, aby byly přístupné i v rámci Petriho sítě pod názvem `params`. Provede se přiřazení do kontextu již vytvořené Petriho sítě (zde import modulu DHT) a do proměnné `params` se přiřadí parametry, využitelné např. v případě ukládání perzistentních dat do souboru `settings.json` zavoláním metody `config.put` s parametry: název proměnné, data, název instance (params[1]). Poté se začnou vytvářet objekty míst a přechodů, které jsou přidávány do již vytvořené sítě a následně jsou propojeny vstupními a výstupními hranami.

```
def init(self):
    self.inPlaces = {}
    self.outPlaces = {"out(humidity)":"humidity"}
    self.inputs= {}
    params = self.params
    self.n = PetriNet('HumidityPN')
    self.n.globals['params'] = params
    self.n.globals.declare("""import DHT""")
    self.n.add_place(Place('p55', [""]))
    self.n.add_place(Place('out(humidity)'))
    self.n.add_transition(Transition('t57'))
    self.n.add_input('p55', 't57', Value('"x"'))
    self.n.add_output('out(humidity)', 't57', \
        Expression('str(DHT.read_humidity(params[2],"22"))'))
    self.n.add_output('p55', 't57', Variable('x'), time=params[3])
```

Výpis 5.4: Pseudokód metody init z aplikace pro čtení vlhkosti z DHT22

Po inicializaci je v novém vlákně zavolána metoda `loop`, která obsahuje cyklus while interpretující zadanou Petriho síť. Tento cyklus je vykonáván dokud není běh aplikace deaktivován prostřednictvím atributu `self._running`. Pokud aplikace obsahuje alespoň jedno vstupní místo pojmenované jako `in(port)`, kde port je název vstupního portu, bude výsledný kód obsahovat část pro zpracování vstupů (výpis 5.5). Jedná se o cyklus, kontrolující zda byly aplikací přijaty nějaké vstupy, které jsou po přijetí vloženy na příslušná místa Petriho sítě.

```
for p in self.inPlaces:
       if self.inPlaces[p] in self.inputs:
              self.n.place(p).add(self.inputs[self.inPlaces[p]])
              del self.inputs[self.inPlaces[p]]
```

Výpis 5.5: Zpracování vstupů v metodě loop

Následující část(viz výpis 5.6) metody `loop` mají všechny vygenerované aplikace interpretující Petriho sítě totožnou a je složena ze tří cyklů. První cyklus zjišťuje pro každý přechod v různém pořadí, zda existují nějaká navázaní se kterými je proveditelný. Pokud dojde k nalezení, provede se odebrání příslušných tokenů a vykonání požadovaných operací. V průběhu vykonávání přechodu se vloží do kalendáře událostí objekt s údaji potřebnými k získání výstupních tokenů a čas jejich navrácení do sítě. Druhý cyklus kontroluje zda kalendář událostí neobsahuje události, které by měly být vykonány. Pokud takové události existují, smaže je z kalendáře a provede umístění příslušných tokenů do výstupních míst. Poslední cyklus kontroluje výstupní místa. Pokud se nacházejí nějaké tokeny ve výstupních místech, provede se jejich odeslání na příslušný výstupní port aplikace a smazání odeslaných tokenů z výstupního místa. Odeslání probíhá zavoláním `super().send`, kde prvním parametrem je výstupní port a druhým obsah zprávy.

```
for t in self.n._trans:
       modes=self.n.transition(t).modes()
       if len(modes) is not 0:
              rand=randint(0,len(modes)-1)
              if self.n.transition(t).enabled(modes[rand]):
                     self.n.transition(t).fire(modes[rand])
while self.n.getCalendar().getFirstTime() <= datetime.now():
       e = self.n.getCalendar().getFirst()[0]
       place = e.getPlace()
       label = e.getLabel()
       binding = e.getBinding()
       place.add(label.flow(binding))
       label.rmBindings()
for o in self.outPlaces.keys():
       out = self.n.place(o).tokens
       if out != {}:
              self.outKeys = []
              [self.outKeys.append(k) for k in out.keys()]
              for k in range(len(self.outKeys)):
                  del self.n.place(o).tokens[self.outKeys[0]]
                  self.output = self.outKeys.pop(0)
                  super().send(self.outPlaces[o],str(self.output))
```

Výpis 5.6: Hlavní část těla metody loop

5.6 Nastavení

Pro uchování informací po restartu implementovaného systému nebo zařízení si systém vytváří soubor `settings.json`. Používá se pro nastavení systému, připojení k `MQTT brokeru` a data uložená uživatelskými aplikacemi, která jsou ihned po spuštění načtena do asociativního pole. Pro ukládání do souboru se používá zápis ve formátu `JSON`, který je snadno čitelný a v případě potřeby je možné nastavit si vlastní hodnoty parametrů. Při ukládání dat z aplikace je vhodné používat navíc parametr název aplikace. Pro ukládání slouží metoda `put` s parametry název, data, název aplikace a pro čtení metoda `get` a pro smazání metoda `remove` s parametry: název parametru a název aplikace.

```
{
    "mqtt_broker": "192.168.0.8",
    "settemp1": {"setPoint": "19.0"},
    "username": "name",
    "password": "passwd",
    "platform": "rpi",
    "node_name": "rpi_loznice",
    "topic": "pn",
    "settemp2": {"setPoint": "20.0"}
}
```

Výpis 5.7: Formát souboru settings.json

node_name Jméno daného uzlu. V případě zadání jména uzlu, je možné používat jméno uzlu místo IP adresy pro směrování MQTT komunikace. Zároveň se bude tímto jménem hlásit ostatním uzlům.

platform Název použité platformy.

route_by_name Nabývá hodnot 0 - vypnuto nebo 1 - zapnuto. V případě zapnutí se používají pro směrování kromě IP adres také jména uzlů.

mqtt_broker IP adresa MQTT brokeru, kterou je potřeba zadat ručně.

username Uživatelské jméno pro přihlášení k MQTT, které je v případě potřeby nutné zadat ručně.

password Heslo sloužící pro přihlášení k MQTT brokeru, které je v případě potřeby nutné zadat ručně.

topic Topic na kterém probíhá komunikace prostřednictvím MQTT zpráv, přednastavený topic je `pn`. Pro použití jiného topicu je možné jej přepsat.

broadcast_everything Parametr sloužící k nastavení odesílání všech zpráv také prostřednictvím zpráv typu broadcast, například pro ladící účely. Může nabývat hodnot `0` - vypnuto nebo 1 - zapnuto.

settemp1, settemp2 Formát pro ukládání zpráv uživatelskými aplikacemi. Aplikace běžící pod názvy `settemp1` a `settemp2` si uložily hodnotu s parametrem `setPoint`.

5.7 Zjednodušení nahrávání

Aby nebylo potřeba po každém restartu systému posílat do uzlu zprávy se zdrojovými kódy aplikací a s Petriho sítěmi, provádí se po úspěšném přijetí uložení uživatelských aplikací do složky `apps`. Po restartu tedy není potřeba opět nahrávat aplikace do systému pomocí zpráv `load` a `loadpnml`, jelikož se systém pokusí všechny aplikace načíst z uložených souborů. Neukládají se ale informace o aktivních aplikacích a neukládá se ani směrovací tabulka. Pro zjednodušení načtení příkazů `activate` a `addroute` se ale může použít soubor `routings.txt` obsahující zprávy oddělené čárkou a koncem řádku. Po startu se tedy zkontroluje, zda tento soubor existuje a v případě nalezení se provede načtení a vykonání zpráv uložených v souboru.

5.8 Testování

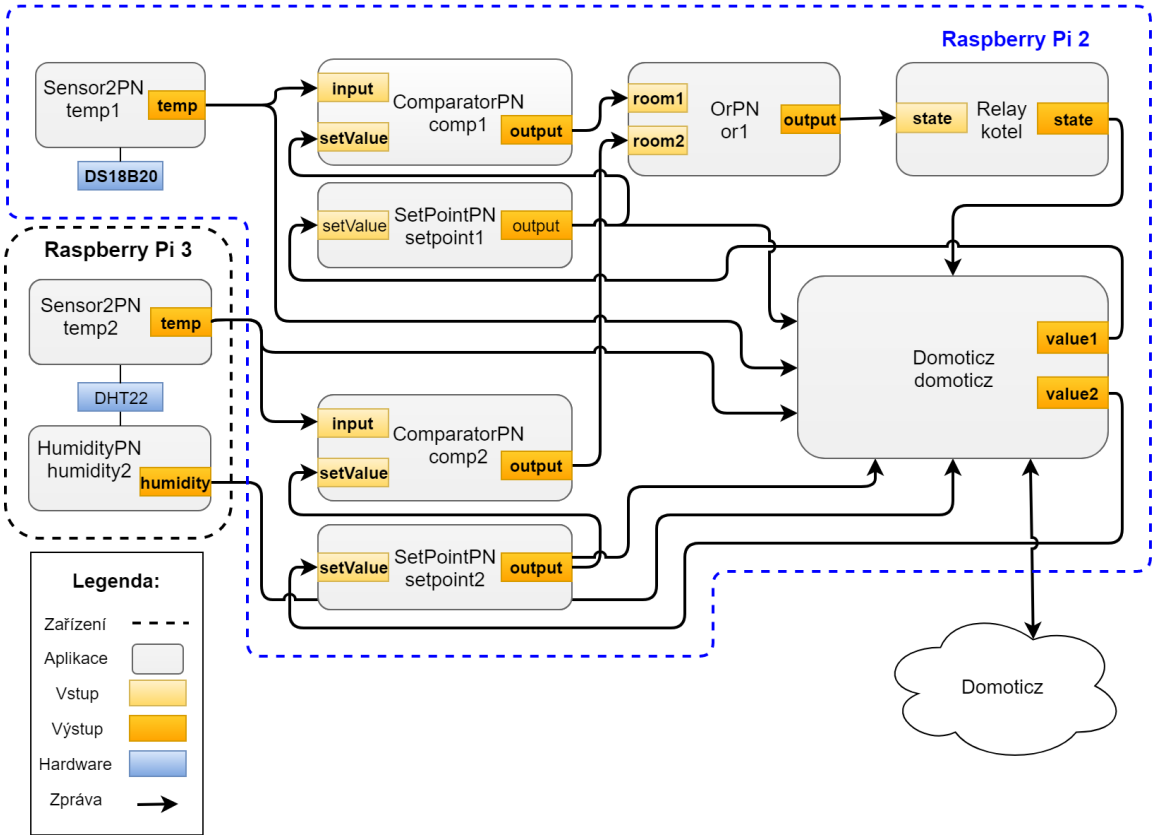
Na obou zařízeních Raspberry Pi 2 a Raspberry Pi 3 byl nainstalován Python 3.5 s potřebnými knihovnami. Uživatelské aplikace byly záměrně vytvářeny tak, aby bylo možné použít stejný kód vícekrát, někdy i s mírně upravenou funkčností. Schéma testovaného systému se nachází na obrázku 5.2. Na prvním zařízení běžel implementovaný systém, s aplikací pro čtení teploty z jednoho teploměru a přijímal teplotu a vlhkost z druhého zařízení. Tyto teploty porovnával s nastavenými teplotami na virtuálních programovatelných termostatických hlavicích a na základě získaných výsledků spínal nebo vypínal kotel. Uživatelské aplikace běžící na zařízeních byly napsány jak v jazyce Python, tak navrženy a popsány pomocí Petriho sítí v aplikaci Renew, které jsou následně interpretovány. Na Raspberry Pi 2 zároveň s implementovaným systémem běžel systém pro automatizaci domácnosti Domoticz, pomocí kterého bylo možné vytvořený systém spravovat. V případě potřeby lze uživatelské aplikace které nepracují s hardware, po úpravě směrování, libovolně přemisťovat mezi zařízeními či přidávat nová zařízení pro rozšíření funkčnosti.

Automatické spuštění implementovaného systému po startu lze nastavit například použitím nástroje `/etc/crontab`, který spustí spouštěcí skript umístěný ve složce s implementovaným systémem pod uživatelem `user`. První řádek z výpisu 5.8 umístíme do `/etc/crontab` a ostatní řádky do skriptu `launcher.sh` s potřebnými právy.

```
@reboot user sh /home/user/../slozka/script.sh &
```

```
#!/usr/bin/env bash
sleep 10
cd /home/user/../slozka/
./system.py &
```

Výpis 5.8: Automatické spuštění implementovaného systému



Obrázek 5.2: Schéma zapojení a komunikace vytvořeného systému

Raspberry Pi 2

```
Název: rpi_obyvak
Umístění: obývací pokoj
IP adresa: 192.168.0.151
Hardware: senzor DS18B20
Aplikace:
        temp1 - Sensor2PN
        comp1 - ComparatorPN
        comp2 - ComparatorPN
        setpoint1 - SetPointPN
        setpoint2 - SetPointPN
        or1 - OrPN
        kotel - Relay
        domoticz - Domoticz
```

Zařízení Raspberry Pi 2 slouží ke čtení teploty ze senzoru DS18B20 pomocí aplikace `Sensor2PN`, která byla navržena pomocí Petriho sítě a spouští se s názvem `temp1` a dvěma parametry určujícími typ použitého senzoru a dobu čekání. Aktivace se provádí s jedním parametrem, který slouží k určení použitého teploměru. Pro čtení ze senzoru DS18B20 se používá knihovna `W1ThermSensor`. Čtení se provádí každých 60 sekund, a výstup je poté odeslán na port temp. Aplikace temp1 se nahraje a aktivuje zprávami 5.7.

Po odeslání souboru s Petriho sítí se vytvoří soubor ve formátu pnml ve složce `pnml` s názvem `Sensor2PN.pnml`. Přijatý soubor se následně převede na aplikaci v jazyce Python a vytvoří se její soubor `apps/Sensor2PN.py`. U dalších souborů se provede převod obdobně.

```
{".",{"192.168.0.151","loadpnml"},{"""pnml_soubor""","Sensor2PN"}}
{".",{"192.168.0.151","activate"},{"Sensor2PN","temp1","DS18B20","60"}}
{".",{"192.168.0.151","addroute"},{{{"192.168.0.151","temp1","temp"},
{{"192.168.0.151","comp1","input"},{"192.168.0.151","domoticz","9"}}}}}
```

Př. 5.7: Zpráva pro nahrání Petriho sítě ve formátu PNML-RefNet s jedním parametrem značícím název, zpráva pro aktivaci aplikace temp1 s parametrem určujícím použitý teploměr a zpráva pro nastavení směrování.

Poté je do zařízení nahrána Petriho síť `ComparatorPN` a jsou aktivovány dvě aplikace `comp1` a `comp2`. Aplikace očekávají vstupní hodnoty na portech input a setValue. Na portu input se očekávají hodnoty získané z teploměru a na portu setValue nastavené hodnoty z aplikací `setpoint1` a `setpoint2`. Hodnota z portu input snížená o požadovanou hodnotu hysterze se porovnává s poslední přijatou hodnotou setValue a pokud je větší vrací hodnotu `1`, v opačném případě vrací hodnotu `0`.

```
{".",{"192.168.0.151","loadpnml"},{"""pnml_soubor""","ComparatorPN"}}
{".",{"192.168.0.151","activate"},{"ComparatorPN","comp1","22","0.5"}}
{".",{"192.168.0.151","activate"},{"ComparatorPN","comp2","20.5","0.5"}}
{".",{"192.168.0.151","addroute"},{{{"192.168.0.151","comp1","output"},
                {{"192.168.0.151","or1","room1"}}}}}
{".",{"192.168.0.151","addroute"},{{{"192.168.0.151","comp2","output"},
                {{"192.168.0.151","or1","room2"}}}}}
```

Př. 5.8: Zpráva pro nahrání Petriho sítě ve formátu PNML-RefNet s jedním parametrem značícím název, zprávy pro aktivaci aplikací comp1 a comp2 s parametry sloužícími pro inicializaci hodnoty setPoint a hysterezi a zprávy pro nastavení směrování.

Po aplikacích comp1 a comp2 se nahraje Petriho síť `SetPointPN` a jsou aktivovány aplikace `setpoint1` a `setpoint2`. Aplikace se spouští s parametrem sloužícím k inicializaci hodnoty setPoint, který použije nepodaří-li se načíst uloženou hodnotu ze souboru `settings.json`. Aplikace mají jeden vstup a přijímají hodnoty na portu setValue. V aktuálním případě přijímají hodnoty z `Thermostat Setpoint` ze systému Domoticz, čímž by se nastavovala teplota na programovatelných termostatických hlavicích. V případě změny hodnoty setPoint je změněná hodnota uložena do souboru a odeslána na port `output`.

```
{".",{"192.168.0.151","loadpnml"},{"""pnml_soubor""","SetPointPN"}}
{".",{"192.168.0.151","activate"},{"SetPointPN","setpoint1","22"}}
{".",{"192.168.0.151","activate"},{"SetPointPN","setpoint2","20.5"}}
{".",{"192.168.0.151","addroute"},{{{"192.168.0.151","setpoint1","output"},
{{"192.168.0.151","comp1","setValue"},{"192.168.0.151","domoticz","7"}}}}}
{".",{"192.168.0.151","addroute"},{{{"192.168.0.151","setpoint2","output"},
{{"192.168.0.151","comp2","setValue"},{"192.168.0.151","domoticz","8"}}}}}
```

Př. 5.9: Zpráva pro nahrání Petriho sítě ve formátu PNML-RefNet s jedním parametrem značícím název, zprávy pro aktivaci aplikací setpoint1 a setpoint2 s parametrem sloužícím pro inicializaci hodnoty setPoint a zprávy pro nastavení směrování.

Dále je nahrána Petriho síť `OrPN` a aktivována aplikace `or1`. Aplikace se nespouští s žádnými parametry. Přijímá hodnoty na dvou portech `room1` a `room2` a porovnává poslední přijaté hodnoty. Po přijetí hodnoty provede funkci a odešle výstup na port `output`, který se použije pro změnu stavu kotle.

```
{".",{"192.168.0.151","loadpnml"},{"""pnml_soubor""","OrPN"}}
      {".",{"192.168.0.151","activate"},{"OrPN","or1"}}
{".",{"192.168.0.151","addroute"},{{{"192.168.0.151","or1","output"},
              {{"192.168.0.151","kotel","state"}}}}}
```

Př. 5.10: Zpráva pro nahrání Petriho sítě ve formátu PNML-RefNet s jedním parametrem značícím název, zpráva pro aktivaci aplikace or1 bez parametrů a zpráva pro nastavení směrování.

Potom je nahrán zdrojový kód `Relay` a aktivována aplikace `kotel` s jedním parametrem nastavujícím dobu provádění cyklu, ve kterém ovládá relé a odesílá aktuální stav na výstup `output`.

```
        {".",{"192.168.0.151","load"},{"""zdroj_kod_Relay"""}}
          {".",{"192.168.0.151","activate"},{"kotel","200"}}
{".",{"192.168.0.151","addroute"},{{{"192.168.0.151","kotel","state"},
                {{"192.168.0.151","domoticz","3"}}}}}
```

Př. 5.11: Zpráva pro nahrání zdrojového kódu aplikace Relay, zpráva pro aktivaci aplikace kotel s parametrem značícím dobu provádění v sekundách a zpráva pro nastavení směrování.

Poslední je nahrán zdrojový kód `Domoticz` a aktivována aplikace s názvem `domoticz` s povinným parametrem, kterým je IP adresa MQTT brokeru a volitelnými parametry pro uživatelské jméno a heslo k připojení. Aplikace tvoří rozhraní mezi implementovaným systémem a systémem Domoticz. Komunikace se systémem Domoticz probíha prostřednictvím MQTT zpráv ve formátu JSON. Aplikace přijímá vstupy od uživatelských aplikací na portech odpovídajících `idx` objektů vytvořených v systému Domoticz. Zprávy ze systému Domoticz přijímá na topicu `domoticz/out` a nastavovací zprávy odesílá na topic `domoticz/in`. Formát pro příjem zpráv je jednotný a pro zpracování typů `Switch` a `SetPoint` stačí porovnat parametry `idx` a `stype`. Hodnoty setPoint se posílají na porty `value1` a `value2`, hodnota stavu kotle se aktuálně nepoužívá. Formát podporovaných odesílaných zpráv je následující:

teploměr(temperature) `{"idx":id,"svalue":"hodnota"}`

kotel(switch) `{"command":"switchlight","idx":id,"switchcmd":"On/Off"}`

vlhkoměr(humidity) `{"idx":id,"nvalue":"hodnota","svalue":"0/1/2/3"}`

termostatická hlavice(setPoint) `{"idx":id,"svalue":"hodnota"}`

```
{".",{"192.168.0.151","load"},{"""zdroj_kod_Domoticz"""}}
{"Domoticz","domoticz","""{"1":"temperature","2":"switch",...}""",
"192.168.0.8","username","password"}}
{".",{"192.168.0.151","addroute"},{{{"192.168.0.151","domoticz","value1"},
{{"192.168.0.151","setpoint1","setValue"}}}}}
{".",{"192.168.0.151","addroute"},{{{"192.168.0.151","domoticz","value2"},
{{"192.168.0.151","setpoint2","setValue"}}}}}
```

Př. 5.12: Zpráva pro nahrání zdrojového kódu aplikace Domoticz, zpráva pro aktivaci aplikace domoticz s parametry: IP adresa MQTT brokeru, volitelné: uživ. jméno, heslo a zprávy pro nastavení směrování.

Raspberry Pi 3

```
Název: rpi_loznice
Umístění: ložnice
IP adresa: 192.168.0.8
Hardware: senzor DHT22
Aplikace:
       temp2 - Sensor2PN
       humidity2 - HumidityPN
```

Zařízení Raspberry Pi 3 slouží ke čtení teploty ze senzoru DHT22 pomocí aplikace `Sensor2PN`, která byla navržena pomocí Petriho sítě a spouští se s názvem `temp2` a dvěma parametry určujícími typ senzoru a dobu čekání. Aktivace se provádí s jedním parametrem, který slouží k určení použitého teploměru. Pro čtení ze senzoru DHT22 se používá knihovna `Adafruit_Sensor` s vytvořeným modulem `DHT` upravujícím formát výstupu. Čtení teploty se provádí každých 60 sekund a výstup je poté odeslán na port `temp`. Aplikace se nahraje a aktivuje zprávami 5.13.

```
{".",{"192.168.0.151","loadpnml"},{"""pnml_soubor""","Sensor2PN"}}
{".",{"192.168.0.151","activate"},{"Sensor2PN","temp2","DHT22","60"}}
{".",{"192.168.0.8","addroute"},{{{"192.168.0.8","temp2","temp"},
{{"192.168.0.151","comp2","input"},{"192.168.0.151","domoticz","2"}}}}}
```

Př. 5.13: Zpráva pro nahrání Petriho sítě ve formátu PNML-RefNet s jedním parametrem značícím název, zpráva pro aktivaci aplikace temp2 s parametrem určujícím použitý teploměr a zpráva pro nastavení směrování.

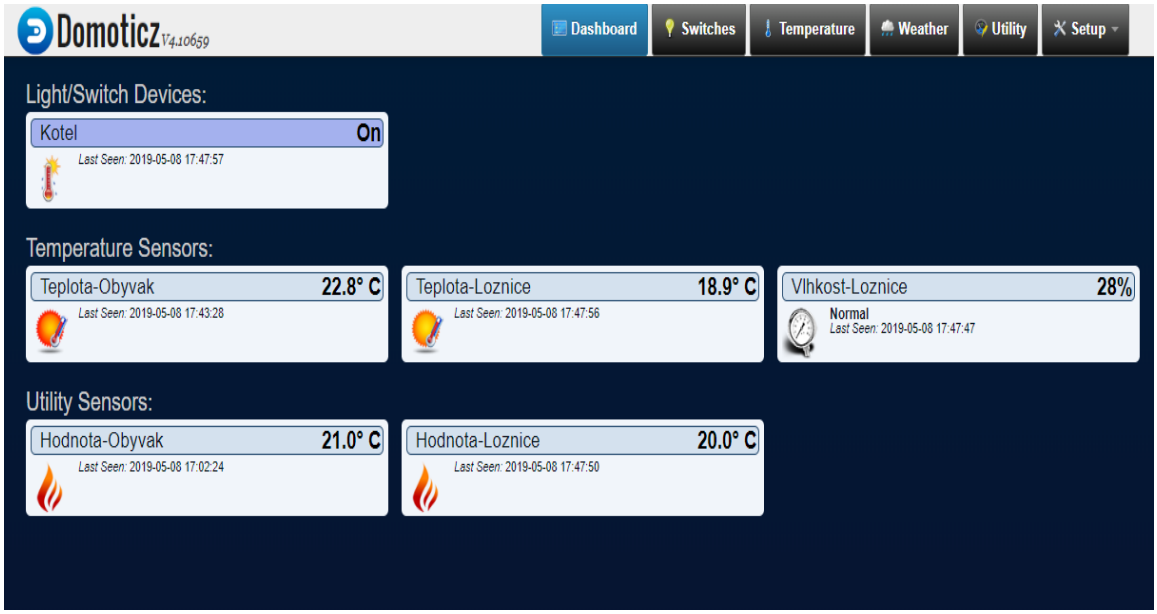
Poté se nahraje Petriho síť `HumidityPN` a aktivuje aplikace `humidity2` se dvěma parametri značící pin a dobu čekání. Aplikace slouží ke čtení vlhkosti ze senzoru DHT22. Pro čtení ze senzoru DHT22 se používá knihovna `Adafruit_Sensor` s vytvořeným modulem `DHT` upravujícím formát výstupu. Čtení vlhkosti se provádí každých 60 sekund a výstup je následně odeslán na port `humidity`. Aplikace se nahraje a aktivuje zprávami 5.14.

```
{".",{"192.168.0.151","loadpnml"},{"""pnml_soubor""","HumidityPN"}}
{".",{"192.168.0.151","activate"},{"HumidityPN","humidity2","4","60"}}
{".",{"192.168.0.8","addroute"},{{{"192.168.0.8","humidity2","humidity"},
{{"192.168.0.151","domoticz","6"}}}}}
```

Př. 5.14: Zpráva pro nahrání Petriho sítě ve formátu PNML-RefNet s jedním parametrem značícím název, zpráva pro aktivaci aplikace humidity2 s parametrem určujícím pin a zpráva pro nastavení směrování.

Domoticz

Ke sledování a nastavení implementovaného systému a pro ukládání hodnot se využívá systém pro automatizaci domácnosti Domoticz s nainstalovaným rozšířením MQTT. Pro připojení systému Domoticz k MQTT brokeru je potřeba přidat v záložce Hardware nové zařízení typu `MQTT Client Gateway with LAN interface` s vlastnostmi pro připojení a topicem `out`, což značí `domoticz/out`. Pro přidání dalších zařízení slouží zařízení typu `Dummy`. Zařízení typu Dummy totiž přidá tlačítko `Create Virtual Sensors`, kterým je lze přidávat další zařízení různých typů. Pro aktualizaci hodnoty „virtuálního sensoru“ stačí zaslat MQTT zprávu v požadovaném formátu na topic `domoticz/in` s hodnotou `idx` odpovídajícího zařízení. Pro ovládání systému byla použita zařízení typu Temp, Switch, Humidity a Thermostat. Kotel je zařízení typu Switch a slouží k zobrazení aktuálního stavu. Teplotní senzory jsou typu Temp a senzor vlhkosti typu Humidity se zobrazením slovního popisu stavu: Dry, Normal, Comfort, Wet. Pro nastavení požadovaných hodnot v místnostech slouží Thermostat SetPointy, přičemž pro každou místnost lze nastavit libovolný program. Změnu teploty lze podmínit časem a dny v týdnu, konkrétním datem, dobou svítání, dobou stmívání atd. Hodnota SetPointu lze dočasně změnit, do provedení další nastavené události. Zároveň jde systém použít k zasílání notifikací na mobil např. skrz aplikaci `IFTTT` nebo emailů použitím automatizačních skriptů. Toho lze využít v případě delší doby od posledního přijetí dat ze senzoru k restartu aplikace nebo zařízení apod.



Obrázek 5.3: Výsledný Dashboard systému Domoticz

Kapitola 6

Závěr

Cílem této práce bylo navrhnout a vytvořit systém umožňující interpretaci vysokoúrovňových Petriho sítí. Tyto sítě mohou být využity k popisu chování různých komponent systému, přičemž kód pro komponenty je možné generovat z vizuálního editoru. Závěrem bylo provedeno testování systému na vhodném příkladu.

Výsledkem této práce je rekonfigurovatelný systém vyvíjený a testovaný na platformě Raspberry Pi. Systém umožňuje dynamické nahrávání a správu běhu aplikací napsaných v jazyce Python nebo navržených pomocí vysokoúrovňových Petriho sítí a vychází z MPOS pro ESP8266 a ESP32 implementovaného v jazyce MicroPython. Ke komunikaci se používá protokol kompatibilní s MPOS, takže je možné posílat zprávy nejen mezi aplikacemi implementovaného systému, ale i do aplikací MPOS a naopak. Díky tomu mohou oba systémy pracovat společně na stejné síti. Po přijetí aplikací se provádí uložení získaných zdrojových kódů do souboru, takže je není potřeba při startu systému opět posílat. Pro automatizaci spouštění jsou existující aplikace automaticky načteny do systému. Aktivace aplikací a nahrání směrovací tabulky po startu systému může být také provedena automaticky vytvořením souboru obsahujícího zprávy, které se mají vykonat. Aplikace také mohou ukládat data do souboru, která mohou být použita pro uchování informací po vypnutí systému nebo pro inicializaci.

Návrh vysokoúrovňových Petriho sítí se provádí ve vizuálním nástroji Renew, jehož výstup ve formátu kompatibilním s PNML se dále používá pro nahrání vytvořených Petriho sítí do systému. Přijaté Petriho sítě jsou ukládány a převáděny systémem na aplikace systému v jazyce Python. Tímto způsobem vytvořené aplikace používají pro práci s Petriho sítěmi knihovnu SNAKES rozšířenou pomocí vytvořeného pluginu o fyzický čas a po spuštění interpretují přijatou síť.

Pro ovládání a kontrolu systému se používá systém Domoticz běžící taktéž na Raspberry Pi, se kterým vyvíjený systém komunikuje skrz MQTT prostřednictvím k tomu vyvinuté aplikace. Použití systému Domoticz také umožňuje nastavení vlastního programu s požadovanými teplotami.

Pro usnadnění návrhu Petriho sítí využitých pro popis aplikací by bylo vhodné mít vizuální nástroj s kontrolou syntaxe a simulací. K využití systému v reálné aplikaci by bylo potřeba přidat do vyvíjeného systému podporu komunikace se systémem pro ovládání programovatelných termostatických hlavic. Systému Domoticz lze také využít k zasílání upozornění prostřednictvím mobilní aplikace nebo emailu, a v případě delší doby bez odezvy některé z aplikací tím upozornit na závažnější chybu.

Literatura

[1] *MQTT Essentials Special*. 2015, [cit. 23.12.2018].
URL https://www.hivemq.com/taxonomy/mqtt-essentials/

[2] *Raspberry Pi - Teach, Learn, and Make with Raspberry Pi*. 2016, [cit. 26.12.2018].
URL https://www.raspberrypi.org/

[3] *Domoticz*. 2017, [cit. 27.12.2018].
URL https://domoticz.com/

[4] *Co je SCADA?* 2018, [cit. 12.12.2018].
URL https://www.promotic.eu/cz/pmdoc/WhatIsPromotic/WhatIsScada.htm

[5] *Eclipse Mosquitto*. 2018, [cit. 21.12.2018].
URL https://mosquitto.org/

[6] *Home Assistant*. 2018, [cit. 27.12.2018].
URL https://www.home-assistant.io/

[7] *openHAB*. 2018, [cit. 27.12.2018].
URL https://www.openhab.org/

[8] *Spolehlivé WiFi moduly a kity s ESP32*. 2018, [cit. 26.12.2018].
URL https://vyvoj.hw.cz/spolehlive-wifi-moduly-a-kity-s-esp32.html

[9] *What is SCADA?* 2018, [cit. 12.12.2018].
URL https://inductiveautomation.com/resources/article/what-is-scada

[10] Ashton, K.: *That 'Internet of Things' thing*. RFiD Journal (2009).
URL http://www.itrco.jp/libraries/RFIDjournal-That%20Internet%20of%20Things%20Thing.pdf

[11] Cabac, L.; Duvigneau, M.; Moldt, D.; aj.: Modeling Dynamic Architectures Using Nets-within-Nets. In *Applications and Theory of Petri Nets 2005. 26th International Conference, ICATPN 2005, Miami, USA, June 2005. Proceedings*, LNCS, 2005, ISBN 978-3-540-26301-2, s. 148–167, doi:10.1007/11494744_10.

[12] 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

[13] Evans, D.: *The Internet of Things - How the Next Evolution of the Internet Is Changing Everything*. 2011, [cit. 12.12.2018].

URL https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[14] Janoušek, V.: *Modelování objektů Petriho sítěmi*. Disertační práce, Brno: Fakulta elektrotechniky a informatiky VUT v Brně, 1998.

[15] Janoušek, V.: *Simulace a návrh vyvíjejících se systémů*. Brno: Fakulta informačních technologií VUT v Brně, 2011, ISBN 978-80-214-4414-0, 123 s.

[16] Koziorek, J.: *Distribuované systémy řízení: učební text*. Vysoká škola báňská - Technická univerzita, 2011, [cit. 10.12.2018], ISBN 978-80-248-2599-1.
URL http://www.person.vsb.cz/archivcd/FEI/DSR/Distribuovane%20systemy.pdf

[17] Malý, M.: *Protokol MQTT: komunikační standard pro IoT*. 2016, [cit. 21.12.2018].
URL https://www.root.cz/clanky/protokol-mqtt-komunikacni-standard-pro-iot/

[18] Markl, J.: *Učební texty k předmětu Petriho sítě I*. Ostrava: Fakulta elektrotechniky a informatiky VŠB - TUO, 2006.
URL http://www.cs.vsb.cz/markl/pn/index.html

[19] Pommereau, F.: SNAKES: a Flexible High-Level Petri Nets Library. In *Proceedings of PETRI NETS'15*, LNCS, Springer, 06 2015, s. 254–265.

[20] Rouse, M.: *What is internet of things (IoT)?* 2018, [cit. 29.12.2018].
URL https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

[21] Stehlík, P.: *ESP32 je tu. Co přinese nástupce ESP8266? - Root.cz*. 2016, [cit. 26.12.2018].
URL https://www.root.cz/clanky/esp32-je-tu-co-prinese-nastupce-esp8266/

[22] Taberner, T.: *The Difference Between M2M and IoT*. 2015, [cit. 29.12.2018].
URL http://advantech-bb.com/the-difference-between-m2m-and-iot/

[23] Vojáček, A.: *IoT MQTT prakticky v automatizaci - 1.díl - úvod*. 2017, [cit. 21.12.2018].
URL https://automatizace.hw.cz/iot-mqtt-prakticky-v-automatizaci-1dil-uvod.html

[24] Zandl, P.: *Internet věcí - Internet of Things*. 2009, [cit. 12.12.2018], ISSN 1213-0702.
URL https://www.lupa.cz/clanky/internet-veci-internet-of-things/

[25] Češka, M.; Marek, V.; Novosad, P.; aj.: *Petriho sítě - Studijní opora*. Brno: Fakulta informačních technologií VUT v Brně, 2009.

[26] Šec, D.: *Distribuované systémy*. Diplomová práce, Hradec Králové: Fakulta informatiky a managementu UHK, 2015.
URL https://theses.cz/id/wow3pb/STAG84353.pdf