VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

INTERPRET VYSOKOÚROVŇOVÝCH PETRIHO SÍTÍ V PYTHONU

HIGH-LEVEL PETRI NETS INTERPRETER IN PYHON

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DANIL GRIGOREV**
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 bakalářské práce

Student: **Grigorev Danil**
Program: Informační technologie
Název: **Interpret vysokoúrovňových Petriho sítí v Pythonu**
High-Level Petri Nets Interpreter in Python
Kategorie: Operační systémy

Zadání:

1. Seznamte se s problematikou vysokoúrovňových Petriho sítí.
2. Seznamte se s knihovnou SNAKES pro podporu implementace Petriho sítí v jazyce Python.
3. Navrhněte a realizujte interpret vhodně zvolené varianty vysokoúrovňových Petriho sítí s ohledem na možnost využití v distribuovaných řídicích systémech.
4. Proveďte testování za pomoci vhodné aplikace a vyhodnoťte dosažené výsledky.

Literatura:

- Jensen, K.: Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, Volume 1. Springer-Verlag, 1991
- RENEW. URL: http://www.renew.de/
- SNAKES. URL: https://www.ibisc.univ-evry.fr/~fpommereau/SNAKES/

Pro udělení zápočtu za první semestr 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: 15. května 2019
Datum schválení: 12. května 2019

Abstrakt

Tato práce je zaměřená na implementací interpretu vysokoúrovňových Petriho sítí v jayzce Python za použití knihovny SNAKES. Je schopná provadění a pokročilé vizualizace navyřených sítí, které jsou popsané v jayzce Python. Vysledný simulator odpovidá zasadům distribuovaného systému, a podporuje provádí v realném čase. Konečný užívatel bude mít příležitost vyzkoušet intuitivní API pro vytvoření a spouštění vysokoúrovňových Petriho sítí.

Abstract

This work is targeting a goal of implementing a high level Petri net interpret in Python. The implementation is based on SNAKES library. The final product is capable of executing and visualising Petri nets, created by user. The simulator is based on distributed systems theory and is executed in real-time. The end user is able to experience a simple and human-friendly API, made for creation and execution of High-Level Petri Nets.

Klíčová slova

Petriho síť, Python, HLPN, simulace, simulator, topení, interpret, real-time, SNAKES, graf

Keywords

Petri Net, Python, HLPN, simulation, interpret, real-time, SNAKES, graph

Citace

GRIGOREV, Danil. *Interpret vysokoúrovňových Petriho sítí v Pythonu.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Vladimír Janoušek, Ph.D.

Interpret vysokoúrovňových Petriho sítí v Pythonu

Prohlášení

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

. . . . . .
Danil Grigorev
15. května 2019

Poděkování

Děkuji Doc. Ing. Vladimíru Janouškovi za obrovskou pomoc a poradu pří psání této bakalářské práce.

Obsah

Kapitola 1

Úvod

Formalizmus Petriho sítí v současné době nemá tolik uznání, kolík si tento modelovací prostředek zaslouží. Cílem této bakalářské práce je ukázat čtenáři, kde všude se dá uplatnit matematický model Petriho sítí, a jaké výhody to přináší. Tato práce je zaměřena na docela populární téma distribuovaných systému, o kterých byste se po přečtení mohli dozvědět více.

Ve výsledku byste pochopili princip práce simulátoru v reálném čase, založeném na využití modelovacích technik Petriho sítí za použití knihovny SNAKES, který má předpoklady pro využití v distribuovaných systémech. Vzhledem k tomu, že takový systém je hodně závislý na komunikací se svými vzdálenými prvky, tak se v průběhu vysvětlí principy protokolu MQTT, který se stal základem implementace simulátoru a jeho komunikačních kanálů. Nakonec byste byli schopni odzkoušet simulátor v praxi, a podívat se podrobně na Petriho sítě, navržené pro ukázkovou aplikaci. Tato aplikace bude vytvořena pro demonstrační účely a založena na teorii distribuovaných systémů.

Výsledný simulační nástroj bude vytvořený tak, abyste mohli snadno získat představu o současném stavu implementací nebo běhu simulace, za využití uživatelsky zaměřeného vizualizačního modulu knihovny SNAKES. Generované obrázky tohoto modulu sestaví nemalou část obsahu této práce. Díky naimplementovaným rozšířením bude čtenář schopný snadno vytvořit sadu pro svůj model.

Vzhledem k existenci hotového nástroje pro modelování a simulaci, bude část této práce věnovaná popisu vytvoření aplikace za účelem usnadnění procesu modelování a seznámení čtenáře s hotovým produktem a postupem pro jeho nastavení.

Kapitola 2

Přehled teorie

V této kapitole jsou popsané teoretické základy Petriho sítí. Taky tato kapitola se soustředí na popis několika podtříd Petriho sítě, zejména na "Vysokoúrovňové Petriho sítě" 2.2.3 a "Značkované Petriho Sítě" 2.2.1 a včetně přehledu míst, kde se tento modelovací prostředek používá, a hraje významnou roli 2.3.

Dalším účelem této kapitoly je seznámení čtenáře s matematickými popisy Petriho síti (2.2.1, 2.2.3) a grafy Petriho sítí 2.2.5, které jsou nezbytné pro porozumění principu prace simulátoru 2.4.

Do obsahu patří popis protokolu MQTT 2.4.5, na kterém je založena externí komunikace simulátoru 4.2.

Taky se tady zmíním o simulační teorii, zaměřené na simulaci Petriho síti v reálném čase. Vyjasní se tím pojem distribuovaných systému 2.4.1, HWIL 2.4.4 a jejích vztah s Petriho sítěmi.

2.1 Petriho sítě

V současné době se Petriho sítě často používají jako modelovací prostředek pro modelování chování systému, za účelem pochopení jeho slabších stránek, ještě než se systém nasadí do provozu. Vyplňuje tím pádem mezeru mezi slovním popisem a návrhem (pomoci takových modelovacích prostředků jako UML), a implementací. Použití Petriho síti pro účel modelování je velice vhodné, jen velmi málo modelovacích prostředku dokáže popsat ne jenom systém jako celek, ale i zjistit slabší stránky jeho implementace, ještě před implementační fází. Stejně tak platí i pro testovací fází – některé chyby se dají objevit jenom po implementací, když Petriho sítě částečně řeší tento problém ještě ve fází návrhu.

Definice Petriho síť (PN) je čtyřce, $PN = (P, T, I, O)$ kde:

- $P = \{p_1, p_2, \cdots, p_n\}$ je konečná množina míst, $n >= 0$;
- $T = \{t_1, t_2, \cdots, t_m\}$ je konečná množina přechodů, $m >= 0$;
- $I = T \to P^\infty$ je vstupní funkce, propojující přechody s množinou vstupních míst;
- $O = P \to T^\infty$ je výstupní funkce, propojující výstupní místa s množinou přechodů;

P $\vee$ T = $\varnothing$, P $\wedge$ T != $\varnothing$.

2.2 Typy Petriho sítě

V této sekci budou popsané teoretické základy několika druhů Petriho sítí, značkovaných – 2.2.1 a Vysokoúrovňových Petriho sítí – 2.2.3 a jejich grafu – 2.2.5 a jak se v takovém grafu provádí přechod – 2.2.4. Taky bude něco málo zmíněno o Stochastických Petriho sítí – 2.2.2.

2.2.1 Značkované Petriho sítě

Značkované Petriho sítě (ZPS) zavadí takový pojem jako značka nebo "token". Značka je základní prvek v ZPS, umožnující její vykonávaní. Vykonávaní se provádí provedením přechodu, během kterého se vymaže odpovídající počet značek z množiny vstupních míst, a do výstupních míst se potřebný počet značek vloží. Tato operace se může vykonávat, dokud nezbude žádný povolený přechod. Na této myšlence je založena simulace provádění Petriho sítě.

Přechod v PS je povolen, dokud každé ze vstupních míst propojených s daným přechodem obsahuje alespoň jednu značku.

Pri vykonávaní sítě Petri vznikají dvě posloupnosti – posloupnost značek a posloupnost přechodu, které byly provedeny. Tyto dvě posloupnosti popisuji zcela vykonání Petriho sítě. [3, p.18–19]

Každá ze šipek v Petriho sítí má váhu. Váha reprezentuje číslo tokenů, které se musí nacházet ve vstupním místě pro to, aby přechod byl proveditelný. Při provedení váha určuje, kolík tokenů se vyjme ze vstupního místa, nebo kolík se jích vloží do místa výstupního.

Definice Značkovaná Petriho síť je šestice $PN = (P, T, I, O, W, M_0)$

- $P = \{p_1, p_2, \cdots, p_n\}$ je konečná množina míst, $n >= 0$;
- $T = \{t_1, t_2, \cdots, t_m\}$ je konečná množina přechodů, $m >= 0$;
- $I = T \to P^\infty$ je vstupní funkce, propojující přechody s množinou vstupních míst;
- $O = P \to T^\infty$ je výstupní funkce, propojující výstupní místa s množinou přechodů;
- $W = F \to \{1, 2, 3, \cdots\}$ je váhová funkce;
- M_0 je multimnožina počátečních značek;

P $\vee$ T = $\varnothing$, P $\wedge$ T != $\varnothing$.

Díky možnosti vykonávat Petriho síť vzniká příležitost, vytvořit simulační nastroj, který je založen na zmíněných teoretických základech. Podrobný popis simulátoru viz. 4.

2.2.2 Stochastické Petriho sítě

Podtřída Stochastických Petriho sítí [Stochastic Petri Net] rozšiřuje pojem značkovaných Petriho sítí o jednu důležitou zásadu – přechod může být označen časovou funkcí. Aby se takový přechod provedl, ten čas musí uplynout, až potom se vyjmou tokeny ze vstupních míst a vloží se do výstupních. Časová funkce nemusí být konstantní, může být závislá na stochastickém procesu, ale v rámci této práce se použilo jenom konstantní čekání na přechod, jelikož chování simulátoru musí být deterministické. Více o implementačních detailech viz. 4.3.1.

2.2.3 Vysokoúrovňové Petriho sítě

Vysokoúrovňové Petriho sítě (High Level Petri Nets) jsou dalším rozšířením Petriho sítí, které umožňují jednodušší modelování takových procesů, pro které jsou důležité výpočty a matematické výrazy. Celkově je tento druh Petriho sítě pohodlnější pro modelování toku řízení kódu nějakého programu a ve své podstatě připomíná grafické programování.

Definice [2, p.11–12]
High Level Petri Net (HLPN) je struktura $HLPN = (P; T; D; T_{ype}; P_{re}; P_{ost}; M_0)$ kde:

- P je konečná množina míst.
- T je konečná množina přechodu, pro kterou platí $(P \cup T = \varnothing)$.
- D je konečná neprázdná množina domén, kde každý element domény se jmenuje typ.
- $T_{ype} : P \cap T \to D$ je funkce pro přidání typu do míst a určeni typu přechodu.
- P_{re}, $P_{ost} : TRANS \to \mu PLACE$ jsou pre a post kondici pro které platí:

$$TRANS = \{(t, m) \mid t \in T, m \in T_{ype}(t)\}$$
$$PLACE = \{(p, g) \mid p \in P, g \in T_{ype}(p)\}$$

- M_0 je $\mu PLACE$ je multimnožina počátečních značek.

2.2.4 Pravidlo pro provedení přechodu

Pokud je dané, že konečná multimnožina T_μ je zapnuta pro značku M, tedy přechod v $HLPN$ muže byt proveden pro značku M' následujícím způsobem [2, p. 12]:

$$M' = M - Pre(T_\mu) + Post(T_\mu)$$

2.2.5 *HLPN* Graf

Graf *HLPN* má následující vlastnosti:

- **Graf site** je sestavený ze dvou prvků – **mist** a **přechodů**, které jsou propojeny mezi sebou šipkami.
- **Typy mist**: Každé místo musí mít přiřazeny jeden typ. Množina typu nesmí být prázdná.
- **Značkování mist**: je kolekce prvků, které musí odpovídat typu místa, ve kterém se nacházejí. Prvky se jmenuji *tokeny* a můžou se opakovat.
- **Šipkové anotaci**: kazda šipka je anotovaná výrazem sestavujícím se z konstant, proměn (x, y) nebo funkcí$(f(x))$. Každá proměna je tipovaná a výrazy se provádějí přiřazením hodnot ke každé z proměn. Vyhodnocení výrazu produkuje kolekci prvku převzatých ze vstupního místa. Kolekce se může opakovat.
- **Podmínky pro přechod**: jsou booleové výrazy ve tvaru $x < y$, které jsou zapsané v anotaci přechodu.
- **Deklaraci**: Vytvářejí definici typu míst, funkci a proměn.

2.3 Aplikace Petriho sítí

Tato sekce popisuje použití Petriho sítí v praxi.

2.3.1 Značkovaná Petriho sít

Rozsah míst, kde se dají aplikovat Petriho sítě je velmi široký. Jako modelovací prostředek můžou sloužit například pro reprezentací vzorů interakce a vznikajících konfliktů mezí člověkem a autopilotem letadla, jak je popsané v následujícím článku.

Příkladem použití značkování Petriho sítí může také sloužit implementace "Dining philosophers problem" [3, p.65–67], což je problém poprvé představený profesorem Dijkstrou v roce 1965. Je to obecně známý problém paralelismu, který názorně zobrazuje dva případy chyb, vznikajících při paralelním výpočtu – vyhladovění a uvíznutí [6].

Obecně značkovaná Petriho síť je dobrá pro ukázku vztahů mezi prvky modelovaného systému. Její rozšíření, jako Stochastické Petriho sítě, přidávají vizualizací časové závislost mezi prvky.

2.3.2 Vysokoúrovňová Petriho sit

Díky svým vlastnostem, HLPN dokážou modelovat případy, kdy nestačí použití jiné varianty Petriho sítí. Například, je důležité zachytit podstatnou část modelovaného procesu, která je v reálném modelu reprezentovaná nedělitelným prvkem a používá ve svém základu fyzické nebo matematické zákony, které nelze vyloučit, bez újmy na použitelností prvku. Obecně pro implementací takového případu slouží programovací zásady, které jsou součásti definice HLPN 2.2.3 a jejího grafu 2.2.5.

V našem případě vysokoúrovňová Petriho síť modeluje a následně simuluje chování systému řízení topení – 5.

2.4 Teoretické základy simulátoru

Cílem simulátoru je provádění Petriho sítí v diskretizovaném 2.4.2 reálném čase 2.4.3, s některými vylepšeními umožňujícími podporu implementací distribuovaných systémů 2.4.1 a simulaci HWIL 2.4.4. Tato podpora je založena na použití MQTT 2.4.5 protokolu komunikace.

2.4.1 Definice distribuovaného systému.

Distribuovaný systém je kolekce na sobě nezávislých počítačů, který konečný uživatel vnímá jako celek.

Distribuované systémy musejí používat decentralizované algoritmy, které požadují následující [5, p.20]:

1. Žádný prvek nesmí vědět informaci o celkovém stavu systému.
2. Prvek uvažuje jenom na základe svého stavu.
3. Chyba v žádném z prvku neovlivní chování algoritmu.
4. Neprovádí se synchronizace podle globálních hodin.

Z definice taky vyplývá, ze distribuovaný systém se obecně sestavuje z více se opakujících prvků, které fungují nezávisle na sobě a nemají přístup ke sdílené paměti, neboli každý prvek má svoji vlastni. Sdíleni stavu v případe distribuovaného systému se provádí pomoci zasílaní zprav.

Tento koncept je velice dobře použitelný pro modelování. Prvky budou reprezentovány Petriho sítěmi, které budou provázané komunikačními kanály. Jelikož každá Petriho síť je prvek nezávislý na vnějších podmínkách, ale jen na současném stavu mist a přechodu a taky jejich kombinaci, tak se to dá pojmenovat o samotě decentralizovaným algoritmem. Petriho síť si můžeme představit jako nezávislý blok, kdežto pro komunikaci mezi těmito bloky se dá použít protokol MQTT 2.4.5, který díky svým vlastnostem dokaže obsloužit neomezené množství klientů najednou. Případ implementace řízeni topení 5.1 názorně ukazuje použití dané myšlenky v praxi.

2.4.2 Diskrétní simulace

Diskrétní simulace je založena ve svém základu na algoritmu známém pod názvem “NextEvent”. Tento algoritmus se řídí následujícím pseudokodem:

Algorithm 1 Diskrétní simulace

```
1: nainicializuj simulaci, čas a plánovač udalosti
2: while dokud je naplánovaná událost do
3:     vyjmi prvni udalost ze seznamu
4:     if cas_události >= T_END then return
5:     nastav čas simulace na čas události
6:     proveď popis chováni události
```

2.4.3 Simulace v reálném čase

Simulace v reálném čase se liší od diskrétní simulace v jedné vlastnosti. Běh takové simulace musí být synchronizován s reálným časem.

Simulace v reálném čase upravena pro účely tohoto projektu je založena na výše zmíněném algoritmu "Next Event" s jedinou změnou, která přidává čekání na synchronizaci simulačního času s reálným, nebo na příchod nové události. Algoritmus v tom případě vypadá následujícím způsobem:

Algorithm 2 Real-time simulace

```
1: nainicializuj simulaci, čas a plánovač událostí
2: while dokud je naplánovaná událost do
3:   podivej se na cas dalsi udalosti
4:   if cas_událostí >= T_END then return
5:   pockej na cas udalosti nebo na prichod nove udalosti
6:   if nova_událost then:
7:     Continue
8:   proveď popis chovaní události
```

Tento algoritmus nachází svoje použití v mnoha simulačních případech. Běžný příklad použití – počítačové hry, kde uživatele zaujme herní prostředí, které simuluje reálný svět, včetně časové roviny. Simulace v reálném čase se také používá, když obyčejné diskrétní simulace nestačí a čas není zanedbatelný. Uživatel může chtít sledovat systém v průběhu simulace za účelem zjištěni anomálii v chovaní jeho prvků nebo zvýšení kvality systému před jeho nasazením do provozu. Takový přístup se používá v případě simulace "Hardware-in-the-loop" – 2.4.4.

2.4.4 Hardware-in-the-loop(HWIL)

Hardware-in-the-loop nebo HIL je technika používaná pro vývoj a testování řídících částí komplexních systémů, díky níž se fyzické části systému nahrazují jejích simulaci a umožňují plynulé testování ještě před nasazením do provozu [7].

Vezmeme si například řídící systém auta. Pokud se to auto zkonstruuje hned, co se dokončí návrh, tak je velká pravděpodobnost výskytu chyby, která může stát někoho život. Samozřejmě před nasazením do provozu se nový výrobek poradně testuje, ale často to objeví chybu, která je následkem jiné chyby. Pokud testujeme systém jako celek, přijít na takovou chybu je poměrně těžké, a časově a zdrojově náročné.

Pro takový případ není špatné si rozdělit celý systém na části a testovat každou zvlášť. Ale je těžké představit si auto, které má karosérii oddělenou zvlášť od kol, ale přesto jezdí. HWIL simulace takový problém vyrušuje tak, že kolo "si myslí" že jede jako součást celého auta, tím že všechny jeho vztahy s ostatními částmi auta modelují simulaci.

Simulátor řízení topení 5.1 není sice věc, která by v případě běžné poruchy stála někoho život, ale pro demonstraci implikace HWIL stačí. V tomto případě navíc jde o druh simulace, zvané "Reality-in-the-loop" nebo RIL. Realita je v tom případě součástí takových simulovaných prvků jako 7.1.

2.4.5 MQTT protokol

MQTT je protokol určeny pro komunikaci v případech nízké propustnosti síti nebo její vysoké nespolehlivosti. V současné době svoje využití nachází v IoT. MQTT zavádí takové pojmy jako klient, broker, zpráva a téma. Dále následují popisy důležitých částí protokolu MQTT z daného zdroje [1].

MQTT Klient Klient je tenká aplikace, schopna navázání spojení s brokerem. K jeho vlastnostem patří podpora odebírání zpráv pomoci filtrování na základě parametru, zvaného "topic". MQTT klient muže byt použit pro zařízení, jejichž výkon je bod srazu, díky lehkosti MQTT protokolu.

MQTT Broker Broker je server obsluhující klienty, navazuje spojeni s klientem a provádí přeposílání zpráv typu `PUBLISH`, které odebírají téma, nacházející se ve hlavičce zprávy. Každý klient se připojí k brokeru tak, že otevře sezení, během kterého může se serverem komunikovat. Pro tento účel klient musí předem vědět IP adresu brokeru, která pro něj bude přístupná.

Téma Každá zpráva posílaná klientem se nepředává přimo dalšímu klientovi, ale obsluhuje se brokerem. Zpráva musí mít specifikované téma, podle které broker určí, kterým klientům tu zprávu musí přeposlat. Takové téma je obvykle unikátní textový řetězec, který bude následně odebírat jeden či více klientů.

Zvláštní význam má, v případě MQTT tématu, znak "#" (U+0023). Téma se obecně skládá z jednoho až několika řetězců, které jsou oddělené znakem "/" (U+002F). Pokud takové téma končí značkou "#", tak je výsledek totožný s "wildcard" znakem "*" (U+002A) z regulárních výrazů.

Příklad:

`net1/#` – přijímá zprávy na téma `net1`, `net1/port1`, `net1/port1/log` atd.
`net2/portX` – přijímá zprávy jenom na téma `net2/portX`.

Zpráva Zprávy musejí mit specifikovaný obsah, a parametr QoS. Obsah zprávy je tělo samotné zprávy, které dostane klient a na základě ní se bude jednat dál. Obsah se posílá v bytech, a pro případ zprávy typu `PUBLISH`, může být prázdný.

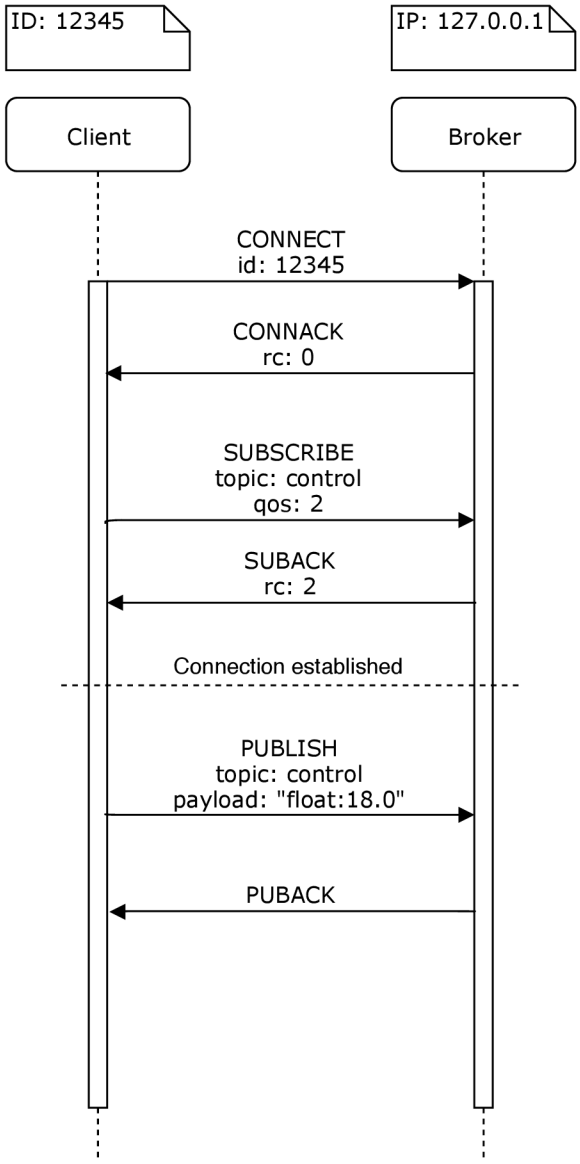
QoS muže byt 3 typu:

- QoS_0 – Nanejvýš jedno dodaní.
- QoS_1 – Alespoň jedno dodaní.
- QoS_2 – Přesně jedno dodaní.

Podle hodnoty tohoto parametru se určí, jak důkladně se bude obsluhovat zpráva.

Tok nastavení Pro navázání spojení s brokerem po nastavení síťového připojení s adresou brokeru, klient musí nejdříve poslat zprávu `CONNECT`, obsahující vlastní identifikátor. Na tu zprávu server musí odpovědět zprávou, typu `CONNACK` potvrzující navázání spojení a nulovým návratovým kódem. Potom klient je povinen poslat zprávy typu `SUBSCRIBE` s tématem, na které chce dostávat zprávy a hodnotu `QoS`. Server odpovídá zprávou `SUBACK`,

návratová hodnota je v případě úspěchu menší nebo se rovná hodnotě `QoS`. Potom už zpráva typu `PUBLISH` s určitým obsahem od klienta, který provedl všechny minulé kroky, bude zpracovaná brokerem.



Obrázek 2.1: Ukázka komunikace Klientu a Brokeru.

Kapitola 3

Návrh simulátoru

V této kapitole se popisuje návrh vytvořeného simulátoru Petriho sítí, založeného na použití knihovny SNAKES a simulující Petriho sítí v reálném čase.

3.1 Obecný návrh

Simulace funguje na základě opakování dvou simulačních cyklu, jak je to popsané v algoritmu 2. První čeká na příchod nové události od plánovače, když ten druhý čeká na čas spouštění nejdřívější události ze seznamu plánovače.

Simulator se tedy sestavuje ze třech částí:

- Plánovač události
- Simulace Petriho sítí
- Komunikace (MQTT)

Obrázek 3.1 vizualizuje strukturu částí simulátoru.

3.2 Plánovač události

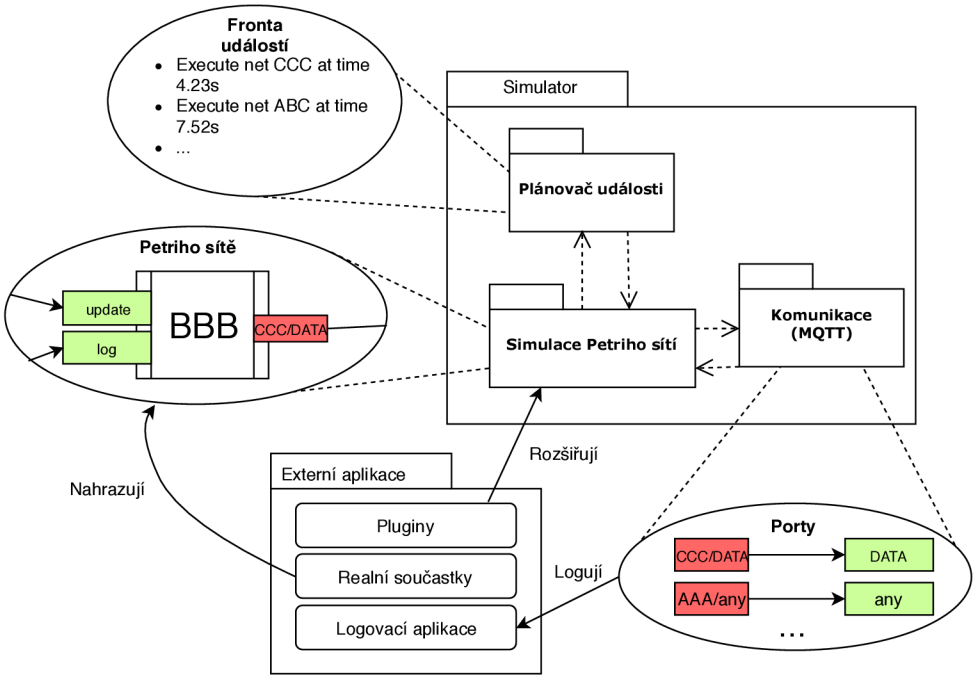
Plánovač události je základem celého simulátoru. Na něm spočívá všechna činnost spojená s plánováním spouštění sítí, blokováním a odblokováním simulační smyčky ve případě změn obsahu jeho plánovací fronty. To umožní provádění simulace v reálném čase, a taky zabraní zbytečnému využití procesorového času ve chvíli když simulátor není obsazen vykonáváním nějaké úlohy.

Naplánované události se musejí ukládat do prioritní fronty v deterministickém pořadí, a obsahovat časovou značku, podle které se určí, která událost se vykoná první.

3.3 Simulace Petriho sítí

Součásti simulátoru musí byt logická část, zaměřena na provádění Petriho sítí. Provádění se musí řídit plánovačem události, který spouští jejích provedení. Taky simulator musí mít možnost v libovolný okamžik běhu naplánovat spouštěni Petrího sítí do plánovače události.

Taková simulace musí byt rozšířitelná a nahraditelná externími aplikacemi a rozšířeními pro účel nahrazení simulace realitou, a pozorovaní stavu běžících částí.



Obrázek 3.1: Architektura simulátoru.

3.4 Komunikační protokol

Pro podporu simulací distribuovaných systému je potřeba použít protokol komunikaci, vhodný pro takové účely. Vzhledem k zaměření tohoto simulátoru na vestavěné systémy, by implementace neměla být náročná na technické vybavení počítače. MQTT protokol je v tom případě dobrou volbou.

Jeho cílem je obsloužení portů Petriho sítě, registrovaní a odhlášení instancí simulátoru, a zajištění stability dynamické podstaty simulatoru.

Kapitola 4

Implementace simulátoru

V této kapitole se popíše implementace simulátoru podle návrhu z kapitoly 3.

4.1 Implementace simulace Petriho sítí.

Tato sekce popisuje implementací simulátoru Petriho sítí.

4.1.1 Knihovna `SNAKES`

`SNAKES` – knihovna v jazyce Python, která nabízí všechno potřebné pro definici a spouštěni několika variant Petriho sítí. Cílem knihovny `SNAKES` je nabídnout badatelům možnost rychle namodelovat nový nápad. Zvláštní vlastnosti knihovny `SNAKES` je možnost vytvářet barevné Petriho sítě s použitím výrazů v jazyce Python pro anotaci přechodu či vstupních nebo výstupních šipek. [4]

Zvláštní vlastností `SNAKES` je možnost implementace vlastních rozšíření pro jednotlivé částí Petriho sítí, nebo přidání nových vlastností pro modelování jiných podtypu Petriho sítí. Tato vlastnost byla zvlášť použita v této prací pro přidání rozšíření 4.3, podporující vytváření Petriho sítí určených pro modelovaní distribuovaných systému. Použití těchto rozšíření je povinné pro správnou práci všech častí simulátoru.

Tato knihovna leží v základu implementace simulátoru, a poskytuje implementací takových prvků, jako `PetriNet`, `Place`, `Transition` a metody pro přenos tokenů pří provedení přechodu.

4.1.2 Simulační cyklus

Plánovač událostí se používá pro plánovaní spouštění sítě do budoucna. Jsou tří případy pro vložení záznamu pro spouštění sítě do plánovače. Může to být uživatelská akce, příchod tokenu do nějakého ze vstupních portů sítě, nebo událost od rozšíření, takových jako `timed_pl` – 4.3.1.

Vložení událostí spouštění sítě do plánovače se nejčastěji provádí ve chvíli příchodu tokenů do nějakého ze vstupních portů. Pro tento účel komunikační část implementace propaguje tu událost do simulátoru, který ve své řádě ji zpracovává a vkládá záznam pro spouštění sítě do plánovače.

Provádění Petriho sítě je záležitost metody `execute_net`. Táto metoda seřazuje zapnuté přechody do skupin podle priority, a následně je spouští. Taky se provádí seřazení uvnitř

každé skupiny pro deterministické chovaní simulátoru. Provádění Petriho sítě je ukončené odesláním shromážděných tokenů z výstupních portů podle popisu v 4.2.4.

Čekání na čas spouštění sítě z algoritmu 2 je naimplementovano metodou `sleep`.

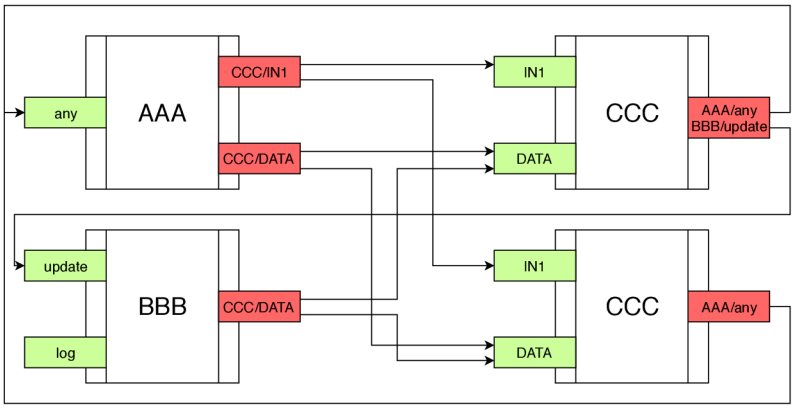
Původní návrh simulátoru předpokládal že simulace bude běžet nějakou omezenou časovou dobu, ale to se ukázalo byt nepoužitelné pro případ využití v teorii distribuovaných systémů. Takže současný stav simulátoru očekává ukončení simulaci příjetím signálu `SIGINT` nebo `SIGTERM` od uživatele.

4.2 Implementace komunikace

Pro implementací MQTT klientu v simulátoru byla použitá knihovna `paho-mqtt`. Tato knihovna obsahuje všechno potřebné pro nastavení komunikačních kanálů mezi porty vzdálených Petriho sítí. Na základě použití této knihovny konečný uživatel může nastavit zvolené místo v Petriho sítí jako vstupní nebo výstupní port pomoci 1. Jejích vzhled a nastavení je popsáné v subsekci 4.2.1.

Směřování

Na jeden vstupní port může být připojeno 0 až několik výstupních portů, nebo jinými slovy vstupní port přijímá všechny zprávy na svoje téma.



Obrázek 4.1: Ukázka směřování zpráv.

Na výše dané obrázku 4.1 můžete vidět ukázku směrování zpráv mezi porty. `AAA`, `BBB` a `CCC` jsou identifikátory Petriho sítí v rámci jediného brokeru. V případě, že se dvojice síť–port vyskytuje vícekrát, zprávy se přeposílají všem instancím. To si můžete všimnout na příkladu sítě `CCC`, která je na obrázku 4.1 dvakrát.

Na obrázku je 6 vstupních portů a 5 výstupních. Vstupní porty jsou označené zeleně, výstupní červeně. Jak je vidět na příkladě dvojic `CCC/IN1` a `CCC/DATA`, dvojice nemusí být unikátní. Počet připojených výstupních portu k jednomu vstupnímu není nijak omezen. Je to dosažené použitím vlastností MQTT klientu popsané v 2.4.5. Stejně tak, počet vstupních portu přípojených k jednomu výstupnímu, může být libovolný. Ukázkou je výstupní port jedné z instanci sítí `CCC`, který je napojený hned na dva vstupní – `AAA/any` a `BBB/update`. To umožňuje posílaní tokenů hned několika příjemcům.

Na příkladě vstupního portu `log` můžete vidět, že není povinné, aby port byl zapojený. V případě simulátoru se chování vstupního portu s takovým nastavením neliší od už připo-

jeného – pokračuje v naslouchání na svoje "téma". Pokud ale nějaký z výstupních portů nebude mít žádnou registrovanou dvojicí, "`NET/TOPIC`", tak jenom zahazuje tokeny namísto odesílání. Víc o použití takového nastavení viz. 4.2.3.

Ve případě že vzdálené porty jsou součástí Petriho sítě, nacházející se uvnitř stejné instanci, tak přenos zpráv se provádí lokálně, bez použití protokolu MQTT.

Více se o nastavení portů viz 4.3.4 a 4.2.1.

4.2.1 Nastavení vstupních a výstupních portů

Na rozdíl od ostatních míst sítě, se vstupní a výstupní porty označují barvou a dodatečným popisem.

Je předem definované, že každá simulační instance odebírá zprávy z tématu `control`. Je to ekvivalent broadkastové adresy pro nastavení. Pak každá instance simulátoru má svůj vlastní instanci MQTT klientu, id kterého se používá pro osobní zprávy určené instanci. Téma kterou odebírá instance je "`private/<MQTT-CLIENT-ID>`". Nastavení ostatních tématu je práce pro uživatele a je popsána v dale.

Nastavení vstupního portu Vstupní porty se označují zelenou barvou. Běžná činnost pro vstupní port – naslouchání na téma uvedené v vizualizací portu. Seznam takových témat se nachází pod navěštím `Listening to`, poslední název z toho seznamu patří místu. Výsledek je na obrázku 4.2.

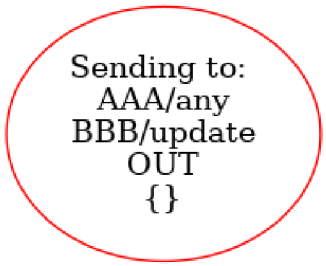
```
1  # Vytvoreni site, mista a pridani ho do site
2  n =PetriNet('CCC')
3  p =Place('DATA')
4  n.add_place(p)
5  # Nastaveni portu 'data' jako vstupni port z~'CCC/DATA'.
6  # Port DATA ze site CCC bude nastaven jako vystupni
7  n.add_remote_input(p, 'CCC/DATA')
```

Obrázek 4.2: Příklad vstupního portu

Nastavení výstupního portu Činnost výstupního portu – odesílaní všech tokenů z napojeného místa po ukončení provedení sítě do témat ze seznamu uvedenému pod návěštím `Sending to`. Výstupní port se označuje červeným rámcem, a taky ve své reprezentací obsahuje seznám míst, kam se odešlou tokeny po ukončení provádění sítě. Výsledek nastavení je na obrázku 4.3.

```
1  # Vytvoreni site, mista a pridani ho do site
2  n =PetriNet('CCC')
3  p =Place('OUT')
```

```
4  n.add_place(p)
5  # Nastaveni portu 'OUT' jako vystupni port do 'AAA/any' a 'BBB/update'.
6  # Porty 'any' a 'update' ze site 'AAA' a 'BBB' budou nastaveny jako vstupni porty.
7  n.add_remote_output(p, 'AAA/any')
8  n.add_remote_output(p, 'BBB/update')
```



Obrázek 4.3: Příklad výstupního portu

Implementace zevnitř vypadá podobně reprezentací samotných portů. Síť, pří volání metody `add_remote_input` nastaví typ místa na vstupní, pak pošle zprávu na téma `control` obsahující dvojicí `[jméno sítě, jméno místa]` a uloží zprávu do čekací fronty. Pak, když se k brokeru přípojí nějaká simulační instance s požadovanou dvojicí síť–místo, zpráva se přepošle všem simulačním instancím. To se zase provede zasíláním zprávy na téma `control`. Hned po přijetí takové zprávy, port cílové sítě se nastaví na výstupní a začne vykonávat svou činnost.

Nastavení portů je tedy oboustranní, a může se opakovat. Platí jenom, že vstupní a výstupní porty se navzájem vylučují.

Přeposílání zpráv se provádí buď lokálně, když příjemce se nachází v rámci jedné simulace, nebo za použití MQTT protokolu.

4.2.2 Vlastní protokol komunikace

Proto, aby simulátor měl podporu přidání vzdálených portů zmíněných v 4.2 a registrací externích aplikací v 4.2.3, vznikla potřeba vymyslet vlastní protokol výměny informací mezi instancemi. Tento protokol musí mít seznam zpráv pro takové simulační případy, kdy port musí předat nebo přijmout tokeny nebo když se objeví instance Petriho síti v nějakém z již už existujících či nově vytvořených simulačních uzlů. Návod pro vytvoření takového je v 4.3.4.

Následující tabulka 4.1 obsahuje přehled typů zpráv, a jejich formátu pro naplnění:

Tabulka 4.1: Tabulka typů zpráv

Typ	Zkrátka	Formát
Update	U	`U_ACTION, MQTT_ID, NEW_NET_NAME`
Request	R	`R_ACTION, TARG_NET/TARG_PLACE, SRC_NET/SRC_PLACE`
Success	S	`REQUEST_COPY`
Failure	F	`REQUEST_COPY, CUSTOM_INFO`

Update

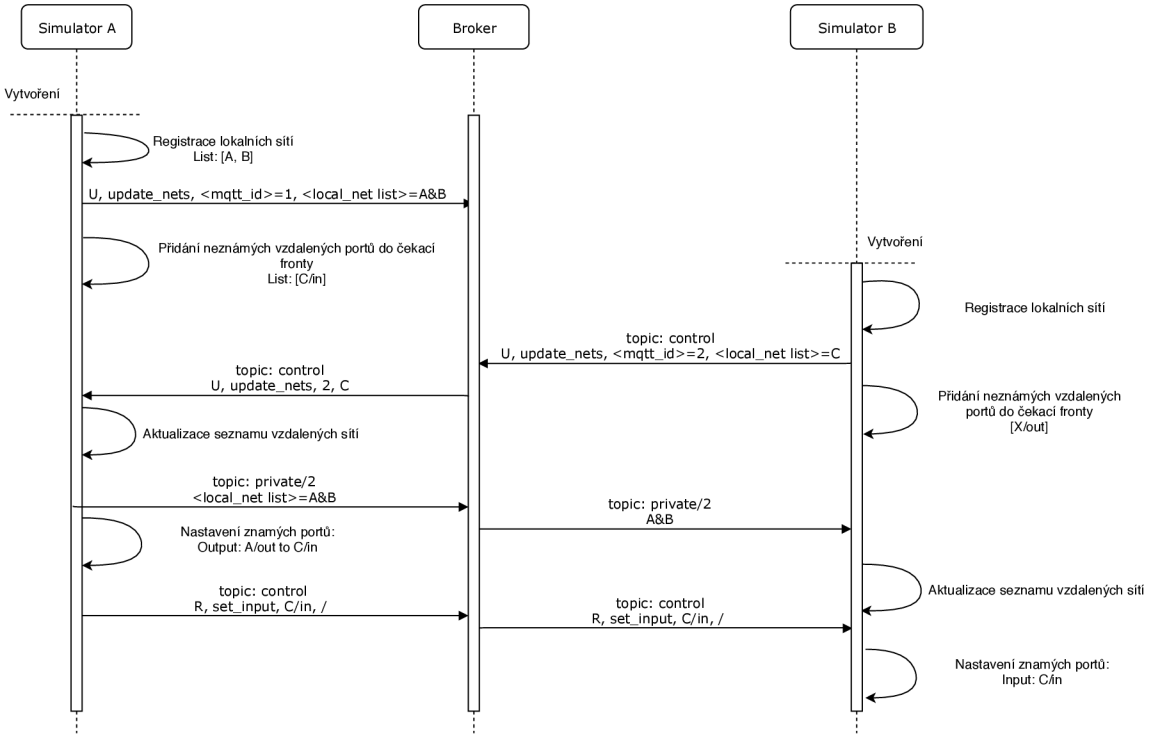
Zprávy tohoto typu oznamují instancím simulátoru, jestli se v jejich okolí neobjevila nová Petriho síť nebo naslouchající aplikace. Pomocí tohoto typu zpráv se provádí oznámení jiným simulačním instancím, že byla ukončena existence už registrovaných Petriho sítí. Takový případ může nastat, když byla nějaká simulace vypnuta zásahem uživatele.

Vytvoření simulace V případě dodání zprávy typu "Update" obsahující `U_ACTION` nastavenou na hodnotu "`update_nets`", taková zpráva bude sloužit k oznámení vzniku nové instance simulace. Proto od té chvíle každá síť bude muset aktualizovat seznám vzdálených Petriho sítí a aktivních portů. V případě, že nějaký port čeká na nastavení, tak dalším krokem se odešle požadavek na jeho nastavení. Poslední akcí je aktualizace tabulky známých vzdálených sítí u nové instance simulací, předané zprávou na téma `private/<mqtt_id>`, což je aproximací osobního kanálu spojení.

Příklad:

```
"U, update_nets, 1, A&B"
```

Hodnota "`MQTT_ID`" v 4.2.2 reprezentuje unikátní identifikační číslo instanci MQTT klientu. Takové číslo je důležité pro zpětnou aktualizaci seznamů známých Petriho sítí na stráně odesílatele. "`NEW_NET_NAME`" jsou názvy Petriho sítí, oddělené znakem "`&`". Výsledný tok nastavení můžete vidět na obrázku 4.4.



Obrázek 4.4: Vznik nové instancí simulace v okolí brokeru.

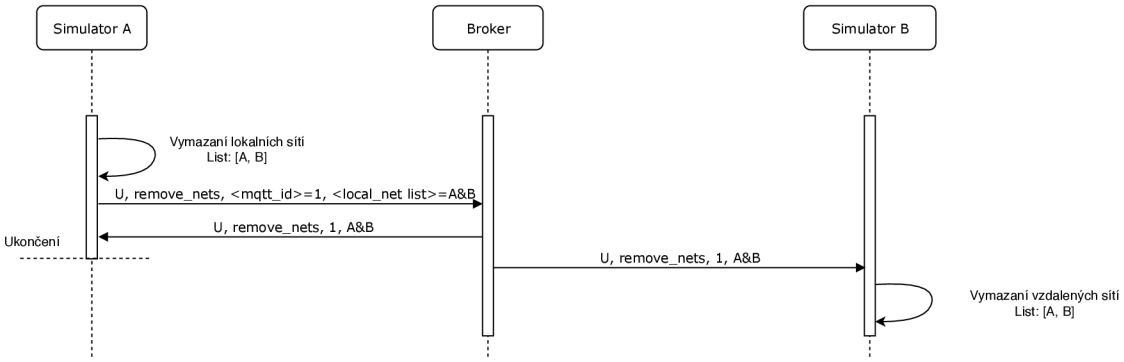
Stojí za zmínku, že `<mqtt_id>` se nastaví pro simulaci ve chvíli, kdy bude vytvořena, a skládá se z hašované hodnoty názvů lokálních Petriho sítí, které jsou součásti instance. Takže pokud nějaká simulace byla spuštěna vícekrát v jeden okamžik, tak jejich chování bude zcela identické.

Ukončení simulace Když se provádí ukončení simulace, tak je důležité oznámit ostatním simulacím, že Petriho sítě, patřící do instance, budou taky vymazány. Takové chování zamezí zbytečnému odeslání tokenů na už neexistující porty, ale nezruší jejich nastavení. Ve chvíli, kdy takový výstupní port nebude mít kam odesílat tokeny, tak je začne zahazovat, aby se nehromadily ve výstupním místě. Více o takovém nastavení viz. 4.2.

Příklad:

```
"U, remove_nets, 1, A&B"
```

Hodnota "`U_ACTION`" je ve případě 4.2.2 nastavená na "`remove_nets`", což po zpracování takové zprávy vymaže sítě "`A`" a "`B`", ze seznamů vzdálených sítí na straně příjemce. Tok provedení této operací je zobrazen v 4.5.



Obrázek 4.5: Ukončení existence simulační instance.

Request

Tento typ zpráv nastavuje vstupní a výstupní porty. Parametr "`R_ACTION`" může nabývat hodnoty `set_input` pro nastavení cílového místa "`TARG_PLACE`" jako vstupní port pro síť "`TARG_NET`". Parametr "`SRC_NET`" a "`SRC_PLACE`" není v tom případě potřeba, stačí uvést rozdělovač.

Příklad:

```
"R, set_input, net1/port1, /"
```

Pro nastavení výstupního portu stačí nastavit parametr "`R_ACTION`" na `set_output` a dodat zdrojové místo a síť nastavením parametrů "`SRC_NET`" a "`SRC_PLACE`".

Příklad:

```
"R, set_output, net1/port1, net2/port2"
```

Success

Tato zpráva se odesílá ve případě úspěšného zpracování nějakého požadavku z 4.2.2. Parametr "`REQUEST_COPY`" musí být přesnou kopií příchozího požadavku.

Příklad:

```
"S, R, set_output, net1/port1, net2/port2"
```

Přijetí, odesílající stranou potvrzovací zprávy Success, znamená že nastavení bylo úspěšné alespoň v jednom případě, což má za následek nastavení místa s jménem `port1` sítě `net1` jako výstupní port do `net2/port2`.

Failure

Během zpracování nějakého požadavku z 4.2.2 může nastat chyba, o které pomocí tohoto typu zprávy se dozví všechny instance simulace. Taková chyba ukončí všechny instance simulací a vypíše chybové hlášení. Parametr "`REQUEST_COPY`" musí být přesnou kopii příchozího požadavku. Parametr "`CUSTOM_INFO`" může obsahovat libovolnou zprávu.

Příklad:

```
F, R, set_output, net1/port1, net2/port2,
Error: place "port1" was not found in net "net1"
```

Běžně se takové chyby vyskytují jenom ve fázi vývoje aplikace, když uživatel zkouší přidávat nové Petriho sítě, takže nehrozí, že výsledná aplikace, nasazená do provozu, najednou spadne s podobným chybovým hlášením.

4.2.3 Využití protokolu pro registrací externích aplikaci.

Vzhledem k tomu, že simulace bez pozorování její stavu nemá smysl, celá předchozí sekce 4.2.2 je navržena tak, aby libovolná externí aplikace byla schopna aplikovat popsané postupy pro navázaní spojení s simulačními instancemi. Jediný požadavek v tom případě je to, že aplikace se musí tvářit jako Petriho síť, která má vstupní nebo výstupní port. Tedy postup pro navázaní spojení je následující:

- Připojit se k MQTT brokeru, se kterým komunikuje pozorovaná skupina simulačních instanci.
- Poslat zprávu typu "`Update`" na téma "`control`", obsahující seznám názvu sítí, které z pohledu simulačních instancí bude obsluhovat navržená aplikace. Příklad takové zprávy je registrace žurnálová aplikace: `"U, update_nets, 1234, logger"`, kde "`logger`" je její název. Z pohledu simulací se jedna o Petriho síť.
- Nastavit porty simulací pro přijetí nebo odeslání tokenů pomoci zprávy typu "`Request`" na téma "`control`": `"R, set_output, kitchen-temperature-updater/Inside update, logger/kitchen"`.

Po uplatnění předchozích kroků, aplikace bude schopná přijímat tokeny z místa "`Inside update`" sítě "`kitchen-temperature-updater`". Tokeny mají gramatiku popsanou v 4.2.4. Takové tokeny se budou posílat na téma "`logger/kitchen`" po každém provedení sítě. Více o takové aplikaci viz. A.4.

4.2.4 Formát přenášených tokenů

Simulator přeposílá tokeny ve tvaru textových řetězců "`type:value`" oddělených znakem &. Ve případě seznamů nebo množin, každá z podmnožin takového prvku bude zakódovaná podobně jednotlivému tokenu, ale struktura seznamu bude zachovaná. Každá externí aplikace by měla dodržovat tuto syntaxi, aby simulator byl schopny zpracovat příchozí tokeny.

Syntaxe v BNF je následující:

1. $< TOKENLIST >:==< SUBLIST > \mid \epsilon$;
2. $< SUBLIST >:==< TOKEN > \ \& \ < SUBLIST > \mid < TOKEN >$
3. $< TOKEN >:==< TYPE >:< VALUE >$;
4. $< TYPE >:== int \mid float \mid string \mid bool \mid list \mid set$;
5. $< VALUE >:== string$;

4.3 Implementace rozšíření

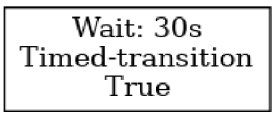
Tento oddíl popisuje implementaci různých rozšíření pro knihovnu `SNAKES`, které znesnadňují návrh Petriho sítí a jejího modelování. Každý z níže popsaných pluginů se dá použít po jejich importu podle tohoto návodu.

4.3.1 Plugin `timed_pl`

Toto rozšíreni používá plánovač události pro spolehlivé plánovaní spouštěni přechodu do budoucna. Toto rozšíření používá jednoduchou logiku pro provedení přechodu – ve chvíli, kdy přechod bude spustitelný, tak se zapne časovač a zkonzumuje odpovídající tokeny ze vstupních míst. Čekání se neresetuje přidáním dalších tokenů do vstupních míst. V případě této implementace přechod vezme nově příchozí tokeny, které se objeví ve vstupních místech a tím zapnou přechod. Po ukončení čekání, bude nová skupina tokenů vložena do výstupních míst s následujícími úpravami, které mohou nastat podle definice HLPN 2.2.3.

```
1  # Vytvoreni prechodu s~casovym zpozdenim
2  pr1=Transition("Timed-transition", timeout=30)
```

Pro vytváření takového přechodu stačí zadat parametr `timeout` do klauzule pro vytvoření přechodu – `Transition`, jak je ukázáno na řádku 2. Hodnota `timeout` se uvádí v sekundách. Výsledný tvar následujícího přechodu je na obrázku 4.6. Ukázku použití ve funkčním příkladu můžete vidět dále.



Obrázek 4.6: Příklad časovaného přechodu

4.3.2 Plugin `prob_pl`

Toto rozšíření přidává možnost spojit několik přechodů do jedné skupiny, a tudíž rovnoměrně rozdělit pravděpodobnost spuštění každé z nich. Pro spojené přechody platí, že musí mít stejnou sadu vstupních míst. Pak se při provedení jednoho z nich rozhodne na základě hodnoty jeho pravděpodobnost, jestli se tento přechod musí provést, nebo se provede některý z jeho sousedů. Součet pravděpodobnosti pro takovou skupinu přechodů musí být vždycky roven 1.

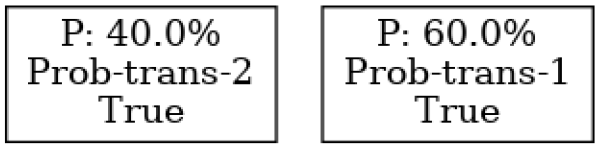
Takový přechod se definuje ve dvou krocích. Nejdříve se vytvoří prvek `Transition` s parametrem `prob`, jak je ukázáno na řádku 2. Příklad použití v obecném případě viz. řádek 12.

```
1  # Krok 1: Vytvoreni prechodu z~parametrem prob
2  pr1=Transition("Prob-trans-1", prob=0.6)
3  pr2=Transition("Prob-trans-2", prob=0.4)
```

Dále stačí spojit pravděpodobnosti přechodu do jedné skupin pomocí klauzule `add_neighbour_transition`. Viz. řádek 2 nebo řádek 16 z kompletního příkladu.

```
1  # Krok 2: Vytvoreni prechodu s~parametrem prob
2  pr1.add_neighbour_transition(pr2)
```

Tato sekvence vytváří následující zobrazení – 4.7.

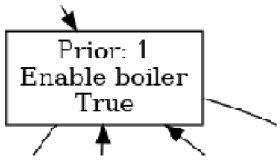


Obrázek 4.7: Příklad pravděpodobnostního přechodu

4.3.3 Plugin `prior_pl`

Tento plugin je určený pro přidání prioritních pochodů do Petriho sítě. Podle nastavené priority se během simulace bude rozhodovat, který přechod se vyhodnotí a provede jako první. Priority mohou být v rozsahu 0 – 100, a řadí se sestupně. Pro nastavení priority element `Transition` má dodatečný a nepovinný parametr `prior` – 2.

```
1  # Ukazka prioritniho prechodu
2  tr=Transition('Enable boiler', prior=1)
```



Obrázek 4.8: Příklad prioritního přechodu s prioritou 1

4.3.4 Plugin `sim_pl`

Na implementací tohoto pluginu je založená práce ostatních pluginů ze skupiny 4.3. Je ve své podstatě spojovacím prvkem mezi simulátorem, *MQTT* klientem a Petriho sítí. Obsahuje přetěžovaní metod z knihovny `SNAKES` pro vykreslování prvků, které jsou rozšířené pluginy. Taky přidává zásadní metody pro nastavení Petriho sítě, umožňující použití simulátoru a provádění spouštění sítě 3.

```
simulation-root
└── net-drawings
    ├── all-in-one
    │   ├── starting
    │   │   └── net1.png
    │   └── current
    │       └── net1.png
    └── server
        ├── starting
        └── current
```

Obrázek 4.9: Struktura uložení zobrazení Petriho sítí.

```
1  sim=PNSim()
2  n=PetriNet("Sample net")
3  n.add_simulator(sim)
```

Stejně tak podporuje nastavení vzdálených portů pro předání tokenů jiným instancím simulátoru připojeným na stejnou adresu `MQTT` brokeru 1, 2.

```
1  n.add_remote_input(it, "temp_gen/Measurement")
2  n.add_remote_output(hen, "boiler-logic/Sensory_input")
```

Vykreslování stávu simulace

Každá instance simulátoru dokáže vykreslit svůj původní a aktuální stav Petriho sítí do následující struktury složek – 4.9.

Původní stav: "`net-drawings/<název simulace>/starting/<net-name>.png`"
Současný stav: "`net-drawings/<název simulace>/current/<net-name>.png`".

4.3.5 Návod pro import rozšíření

Vzhledem k odlišnému postupu pří přidání rozšíření do SNAKES, pro import simulačního nástroje a dalších rozšíření ze skupiny, uživatel musí dodržovat správné pořadí při importu knihoven. Konvence probíhá následujícím způsobem.

```
1  from snakes.nets import *# Site, mista, prechody...
2  from simul import PNSim # Simulacni knihovna
3  snakes.plugins.load(
4  ["gv", "timed_pl", "sim_pl", "prob_pl", "prior_pl"],
5  "snakes.nets",
6  "plugins") # Seznam rozsireni pro import
7  from plugins import *# Redefinovane metody z~rozsireni
```

1. Přidání knihovny snakes a jejich prvků 1.
2. Import simulační knihovny, která poskytne základ pro rozšíření 2.
3. Přidání rozšíření podle požadovaných vlastností sítě 3. Ze skupiny doporučených pluginů stojí za zmínku plugin "gv". Jeho implementace používá nástroj `GraphViz` pro vytvoření vizuální reprezentace Petriho sítě. Více o jeho použití v rámci simulátoru viz. 7.1. Zbytek je popsán v sekcích 4.3.1–4.3.4.

4. Pro redefinici a přidání nových metod z rozšíření slouží řádek 7. Od dané chvíle použití knihovny SNAKES je stejné jako v návodu a taky 7.

4.4 Plánovač události

Plánovač události je naimplementován jako prioritní fronta za využiti vestavěné knihovny `heapq` v jazyce Python. Api naznačuje, že pro přidání a výběr prvků se používají metody `heappush` a `heappop`. Priorita v takové frontě se určuje číselnou hodnotou, která se vkládá do fronty. Tato hodnota může být prvním prvkem v seznamu elementů. Druhým parametrem pro určení priority je pořadí, ve kterém byl prvek vložen. Tato sada pravidel umožňuje vkládat záznamy, obsahující čas spuštění jako první prvek, a podle toho se zařadí na příslušné místo ve řadě. Metoda `heappop` vybere prvek ze seznamu s nejvyšší prioritou, což bude element s nejnižší hodnotou času.

Pro účel plánování události se používají dvě funkce – `schedule_at` 5 a `schedule` 3. Jedna slouží pro vkládání událostí na čas od začátku simulace, když ta druhá pracuje se současným časem simulace. Funkce jsou určeny pro počáteční spouštění sítí po přidání do simulátoru. Pro případ že by uživatel potřeboval provést operaci hned po vložení, bez porušení pořadí předchozích operací, je definovaná konstanta `PNSim.NOW`.

Záznamy musí odpovídat určitým zásadám, aby simulátor byl schopný takový záznam zpracovat. Každý záznam obsahuje ve své struktuře referenci na zvolenou funkci jako první prvek. Následující prvky seznamu jsou argumenty, které budou předány této funkci ve chvíli její vyhodnocení.

Ukázku takového záznamu můžete vidět na řádku 1.

```
1  ev =[simul.execute_net, net.name] # Udalost a jeji argumenty
2  # Planovani udalosti na soucasny okamzik.
3  sim.schedule(event=ev, tm=PNSim.NOW, prior=0)
4  # Planovani udalosti na cas 15s od zacatku simulace.
5  sim.schedule_at(event=ev, tm=15, prior=0)
```

Pro účely vývoje simulátor vypisuje logy z plánovače události do souboru `./planner.log'` ve složce, kde byl spuštěn.

Kapitola 5

Aplikace – návrh

V této kapitole je popsán návrh a implementace zvolené aplikace simulátoru – chovaní řízení topení.

5.1 Řízeni topení

Pro demonstraci práce simulátoru bylo rozhodnuto namodelovat distribuovaný systém modelující chování topení, který pravděpodobně používáte u sebe doma. Takový systém se sestavuje z několika částí. Základním prvkem je plynový spotřebič nebo bojler. Ten ohřívá vodu a uvádí ji do pohybu po potrubích. Pak každý z pokojů ma na starosti tepelné čidlo a řízení, které udržuje v pokoji nastavenou teplotu podle kalendáře teplot pro daný čas. Každou hodinu se z tabulky teplot vezme aktuální teplota pro současnou hodinu a tento pokoj, pak řízení rozhodne jestli má zapnout topení nebo ne. Pokud žádný z pokojů v současné době nemá zapnuté topení, tak se karma musí vypnout za účelem šetření plynu. Toto řízení provádí vestavěný systém v bojleru, protože je zcela mechanický. Každý z těch prvků bude dále namodelován Petriho sítěmi. Pro pochopení struktury viz. 5.1.

5.2 Struktura aplikace

Návrh 5.1 se sestavuje z 5 částí:

- Řízení
- Sklep
- Obývací pokoj
- Kuchyně
- Vnější prostředí

Každá z částí návrhu je zvláštní instance simulátoru. Taková instance může být spuštěna nebo pozastavena v různou dobu a zcela nezávisle na jiných instancích simulátoru nebo jiných podmínkách.

Takové rozdělení se vysvětluje snahou napodobovat skutečné rozložení pokojů. Ve sklepě se nachází bojler, což je reprezentováno Petriho sítí zvané “`Boiler`” a stav teploty toho pokoje se řídí svým vlastním mikropočítačem a jeho reprezentací je síť “`Sklep: simulace prostředí`”. Skutečnou síť pro bojler můžete vidět na obrázku 6.7, a její popis je v 6.4.

Pak každý z pokojů disponuje svou vlastní instancí simulátoru, která simuluje teplotní změny uvnitř pokoje.

Poslední, ale přesto nejdůležitější částí návrhu 5.1, je řízení pokojů. V reálných podmínkách takové řízení se provádí na zvláštním serveru, což reprezentuje instance simulace, zvaná "`Řízení`". Její součásti jsou Petriho sítě, simulující chování čidel a ventilu radiátoru v pokoji, jako "`Kuchyně: řízení ventilu`", a taky tabulky očekávaných teplot pro daný čas "`Tabulka teplot`".

Aby tento návrh, co nejpřesněji odpovídal realitě, bylo rozhodnuto, namodelovat chování teploty v pokoji. Pro tento účel, například slouží síť ze simulační instance "`Sklep`" pod názvem "`Sklep: simulace prostředí`".

Pro splnění tohoto úkolu nejdříve musíme vědět, jak výkonné je topení v pokoji. Běžné hodnoty pro tři druhy pokojů můžete vidět v následující tabulce 5.2. Dále nás zajímají tepelné ztráty a taky množství energie potřebné pro ohřáti pokoje. Pro výpočet tohoto parametru použijeme následující vzoreček. Základním parametrem, rozhodujícím o tepelné kapacitě pokoje, je plocha vnitřního povrchu pokoje V, respektive celková plocha povrchu stropu, podlahy a stěn. Pro ně je důležité vědět koeficient ztrát tepla K. Zvlášť se počítají okna a dveře, vzhledem k jejich větší vodivosti tepla. Pak zbývá jenom zjistit rozdíl současné teploty uvnitř pokoje a očekávané teploty pro tuto hodinu – ΔT. To všechno dosadíme do vzorečku a zjistíme celkové množství energie, potřebné pro jeho ohřáti. [8]

$$Q[kW/h] = \frac{V * \Delta T * K}{860}$$

Pak koeficienty vypadají následovně:

Tabulka 5.1: Tabulka parametrů pokojů

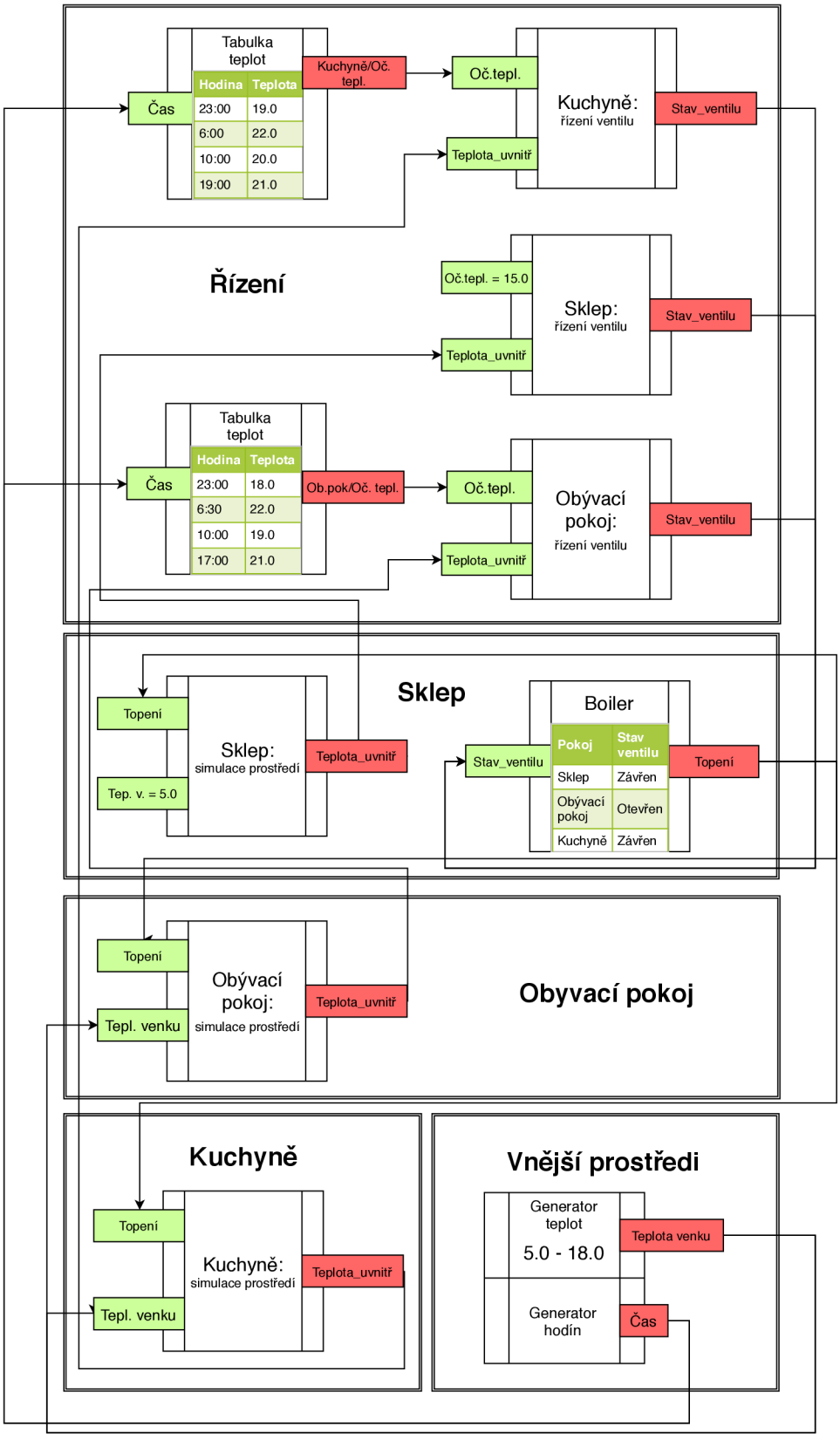
Typ pokoje	Koeficient tepelných ztrát	Objem $[m^3]$	Očekávaný výkon topení $[W]$
Kuchyně	0.5	$4 * 3 * 2.5$	1200
Obývací pokoj	0.6	$6 * 4 * 2.5$	2300
Sklep	0.8	$3 * 3 * 2.2$	300

Na výpočet tepelných ztrát pokoje jsem použil tento nastroj. Vychází to zhruba následovně:

Tabulka 5.2: Tabulka tepelných ztrát pokojů

Typ pokoje	Ztráty $[kW/h]$
Kuchyně	0.7
Obývací pokoj	1
Sklep	0.4

Celé chování systému odpovídá zákonu zachování energie, a tak můžeme odvodit rovnici pro výpočet jejího potřebného množství.



Obrázek 5.1: Architektura aplikace

Kapitola 6

Aplikace – implementační detaily

V této kapitole jsou detailně popsaný obrázky Petriho sítí, vygenerované během simulace, které umožňují simulaci senzorů teplotního řízení pokojů 6.3, ohřívaní pokoje a převod energii na teplotu pokoje – 6.2, a jiné.

6.1 Simulace vnějšího prostředí

Tato část aplikace je vyhrazena do zvláštních simulačních instanci, vzhledem k tomu že není závislá na žádných jiných částech aplikace. Obsahuje jenom jednu síť ve svém zapouzdření – Simulace počasí a času 6.1. Tato síť vytváří dvě podstatné věci pro tuto aplikaci, čas a teplotu pro tento čas. Podrobnější návrh a implementace je v 7.1.

6.2 Simulace pokojů

V rámci této simulace se popisuje chování řídícího systému senzoru a simuluje se chování teploty pro každý z pokojů. Implementace kódu v jazyce Python se dá najít tady.

6.2.1 Výměna energii se vnějším prostředím

Tato Petriho síť je jednou ze součásti simulace pokoje, a může být vyměněna za podrobnější a rozsáhlejší její variantu ve případě že přesnost simulace je bodem srázu. Její cílem je modelovat chování změny teploty v pokoji během výměny teploty se vnějším prostředím. Pro tento účel slouží srovnání současné hodnoty teploty venku a vevnitř pokoje. První hodnota je uložena pro lokální využití v místě s názvem "`Temperature outside`". O její aktualizaci se stará vstupní port "`Outside update`". Pro aktualizaci vnitřní teploty je vstupní port "`Inside temp update`" a lokální hodnota se nachází v "`Temperature inside`".

Výměna teploty se vnějším prostředím se provádí na základě pollingu každých 30 vteřin, což se řídí časovanými přechody "`Temperature gain`" a "`Temperature loss`". Výsledek provádění takového přechodu je vzorec, který spočítá kolík [W] energii bylo ztráceno nebo získáno během těch 30 vteřin – $q\_lps * exch\_time$. Hodnota q_lps je definovaná jako globální proměna v prostředí Petriho sítě. Kód této sítě se nachází v souboru `temp_sensor.py`.

Výsledek můžete vidět na obrázku 6.4.

6.2.2 Převod změn energii na teplotu

Tato síť spojuje hned několik častí simulace řízení teploty v pokoji, a složí pro převod změn energii do stupnici Celsia. Každý z pokojů má uloženou svou hodnotu v místě "`Q per C`". Pokud do vstupního portu "`Q change`" přichází změna energii ze sítí 6.2.1 nebo 6.2.3, ta hodnota bude převedena do stupnici Celsia a přidá se do aktuální hodnoty teploty v pokojí, uloženou v místě "`Temperature inside`". Tato změna bude oznámena takovým častém simulací jako řízení – 6.3.1 přes výstupní port "`Inside update`".

Výsledná síť je v 6.2.

6.2.3 Simulace ohřívání pokoje

Tato část aplikace je taky vyhrazena do zvláštní instance Petriho sítě, a její cílem, simulace ohřívaní pokoje pří zapnutém bojleru a otevřeném ventilu v pokoji. Informace o stavu ventilu přichází do vstupního portu "`Valve update`". Stav bojleru přichází do "`Heater update`". Jejích lokální stavy se ukládají do míst "`Heater state`" a "`Valve state`".

Samotné ohřívaní pokoje se provádí časovaným přechodem "`Heating`", který vkládá množství energii, vyprodukované radiátorem za dobu "`heat_time`" podle vzorečku $q\_gps * heat\_time$. Hodnota q_gps je definovaná pro každý z pokojů zvlášť jako globální deklarace v prostředí Petriho sítě. Výsledek můžete vidět v 6.3.

6.3 Řízení pokojů

Následující Petriho sítě provádějí obsluhu systému pokojů. K nim patří, například řízení senzoru 6.3.1 a tabulka očekávaných teplot v pokoji 6.3.2 během dané hodiny.

6.3.1 Řízení

Řízení pokojů je vizualizované na obrázku 6.6. Funkce senzoru je velice jednoduchá, a spočívá v oznámení bojleru že pokoj potřebuje zapnout ohřívaní. To se provádí odesíláním dvojici "`("jméno pokoje", "ventil je otevřen")`" z portu "`Valve state`". Pro splnění tohoto úkolu, síť potřebuje porovnat hodnoty očekávané teploty pro daný pokoj na tento čas, a současné teploty v pokoji. Informací o poslední přichází do portu "`Temperature inside`". Takové srovnání se musí provádět jen tehdy, kdy se stane nějaká změna v současné teplotě pokoje. Pro oznámení změn teploty uvnitř, slouží port "`Temperature update`". Současný stav očekávané teploty se ukládá do místa "`Expected temperature`".

6.3.2 Tabulka očekávaných teplot

Účelem této Petrího sítí je dynamická změna očekávané teploty v pokoji podle příchozí hodnoty času. Tu, Petriho síť na obrázku 6.5 obdrží ze vstupního portu pod názvem "`Time update`". Následující obsah tabulky je vytvořen dynamicky. Jsou to místa pod názvem Time-HH:MM, obsahující číselný token, reprezentující počet vteřin z počátku dne. Té hodnoty se použijí pro zjišťování, do kterého časového rozmezí spadá nová hodnota času. Podle něj určí teplota v pokoji por danou dobu, která se vezme z výstupní šipky přechodu s názvy "`Enable: HH:MM-HH:MM`". Výsledek bude uložen do místa "`Expected update`".

Pokud nová hodnota očekávané teploty se liší od předchozí, která je uložena v místě "`Expected temperature`", tak ta změna bude předaná pro odesílaní výstupnímu portu "`Temperature update`".

6.4 Simulace bojleru

Součásti simulace je taky popis chování bojleru. Jeho cílem je sledování stavu ventilů v pokojích. Sémantika je jednoduchá – pokud je nějaký z pokojů má otevřený ventil, on to oznámí vstupnímu portu "`Sensory input`" dvojicí tokenů `("name", "enabled")`. Když alespoň jeden pokoj má otevřený ventil, tak bojler musí začít pracovat na jeho ohřáti.

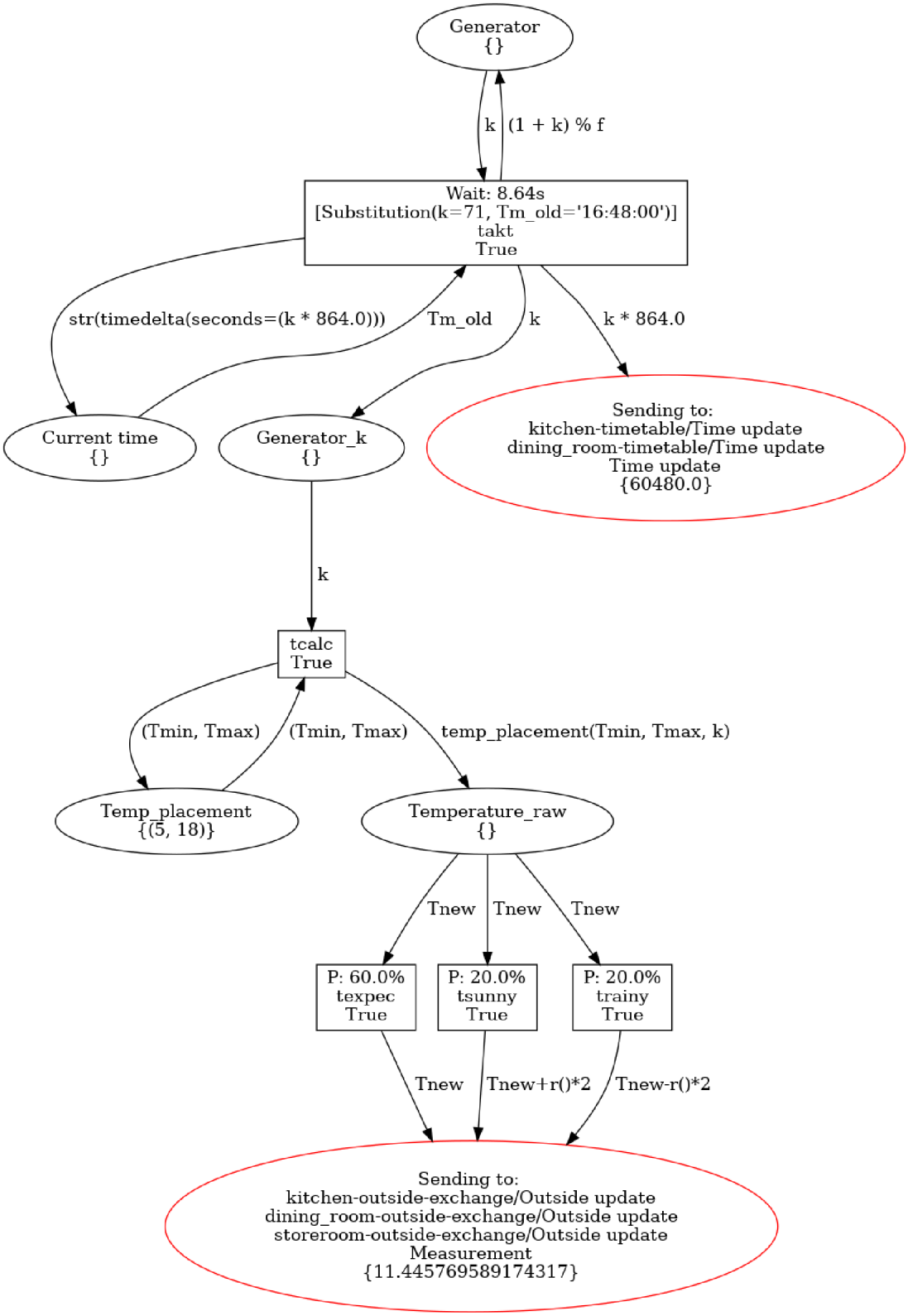
Celá síť obsahuje dvá základní řídící místa – "`Table`" a "`Active Table`". První je tabulka registrace pokojů. Tam se ukládá aktuální stav ventilu pokoje, dokud nebyla změna jeho stavu zpracovaná aktivní tabulkou. V aktivní tabulce se nacházejí pokoje, které v danou chvílí požadují zapnuté topení. Pokud je v ní alespoň jeden takový pokoj, tak bojler se přepne do zapnutého stavu. Současný stav bojleru je uložen v místě "`Boiler Enabled`", a změny jeho stavu se oznamují na výstupní port "`Boiler state`".

Výsledná síť je na na obrázku 6.7.

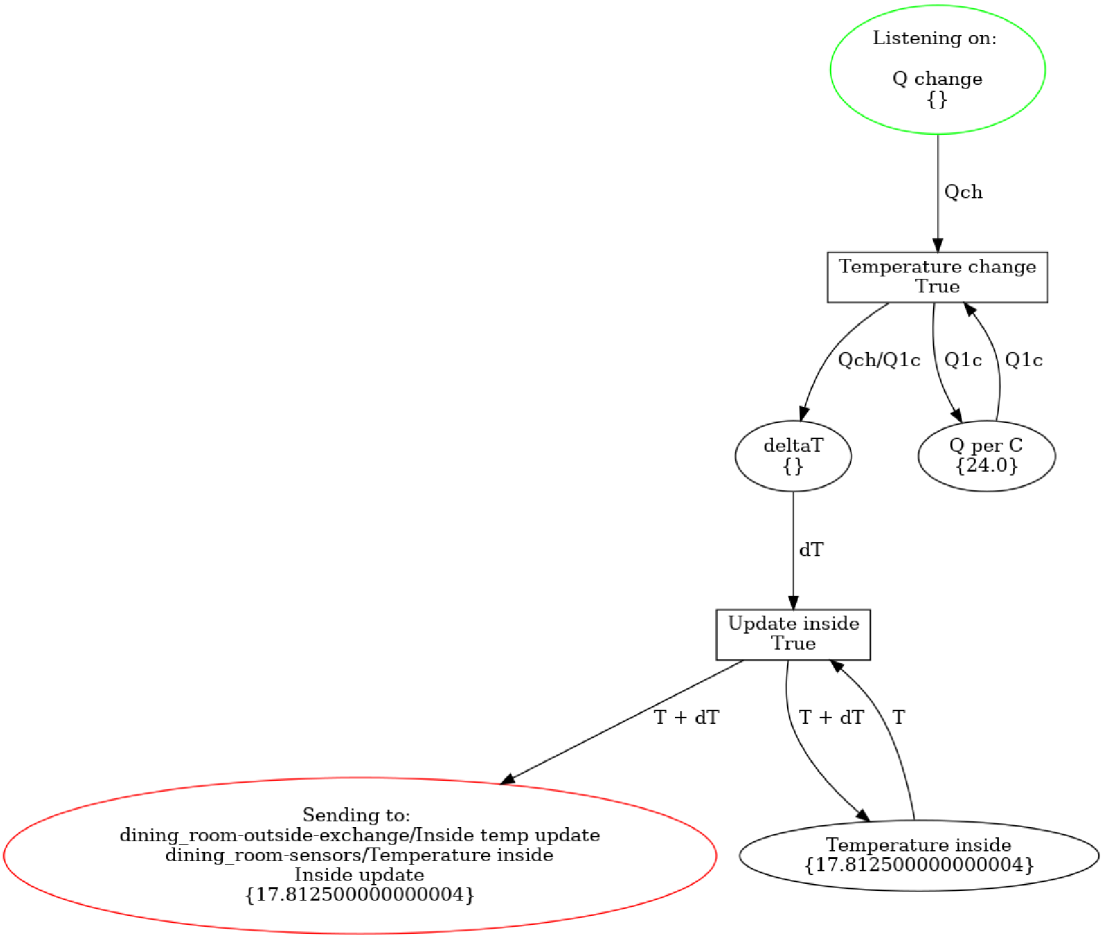
6.5 Nahrazení simulace realitou

Stojí za to se zmínit, že účelem této aplikace není zcela přesná simulace chování topení, ale její zjednodušená kopie. Do úvahy se nebere například to, že rychlost výměny tepla s vnějším prostředím není konstantní, jak je uvedeno v tabulce 5.2, ale je závislá na spoustě parametrů, jako teplota vnějšího prostředí, materiál a tloušťka stěn pokoje atd. To se nepočítá v současném návrhu, ale může být přidáno ve případě potřeby. Účelem této aplikace je vizualizace práce vytvořeného simulátoru a demonstrace využití v distribuovaných systémech.

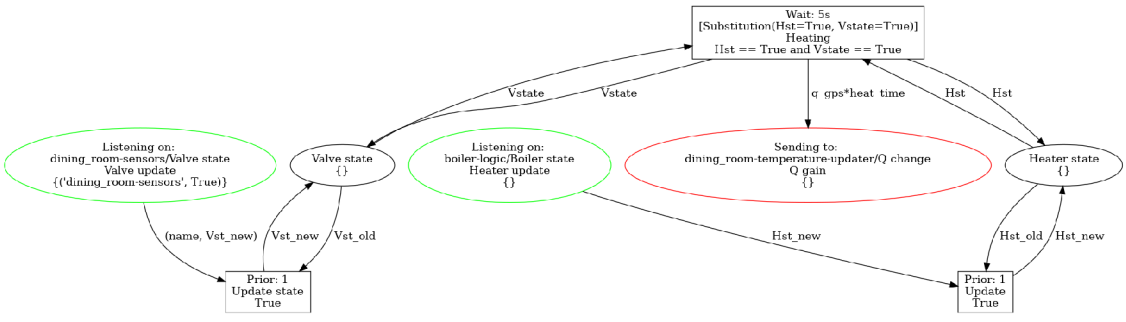
Každou Petriho síť z popsané aplikace by bylo možné vyměnit za reální součástku, zcela nebo částečně. Ve případě že by uživatel měl už existující system topení, ale by chtěl odzkoušet nová čidla pro měření teploty v pokoji, a jejich řízení, tak by se dalo popsat chování čidel Petriho sítí, a provázat síť vstupními a výstupními porty s už existujícími součástkami. Požaduje se od tech součástek jenom podpora MQTT a dodržení zásad vložení externí aplikace z podsekci 4.2.2.



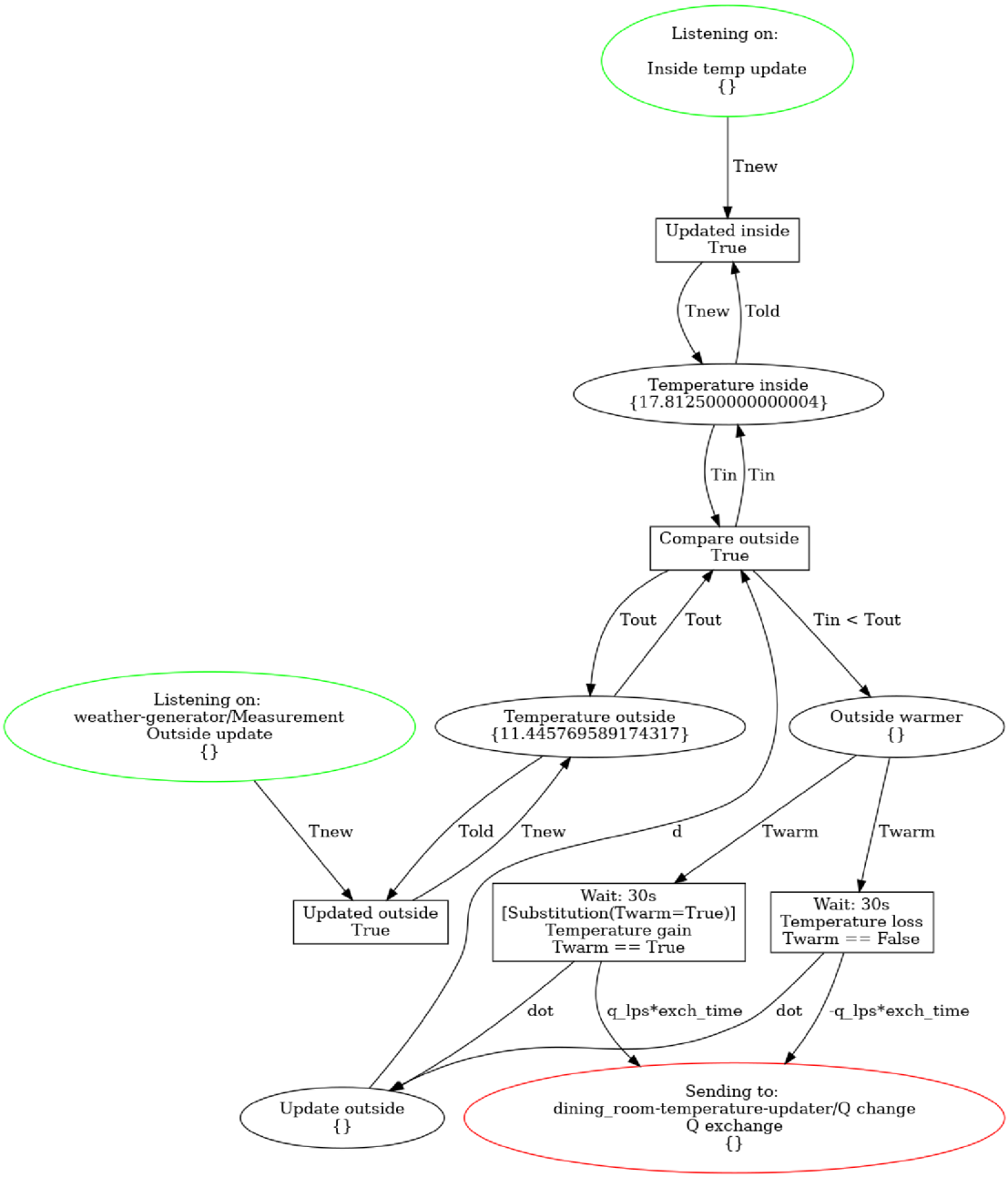
Obrázek 6.1: Simulace počasí a času.



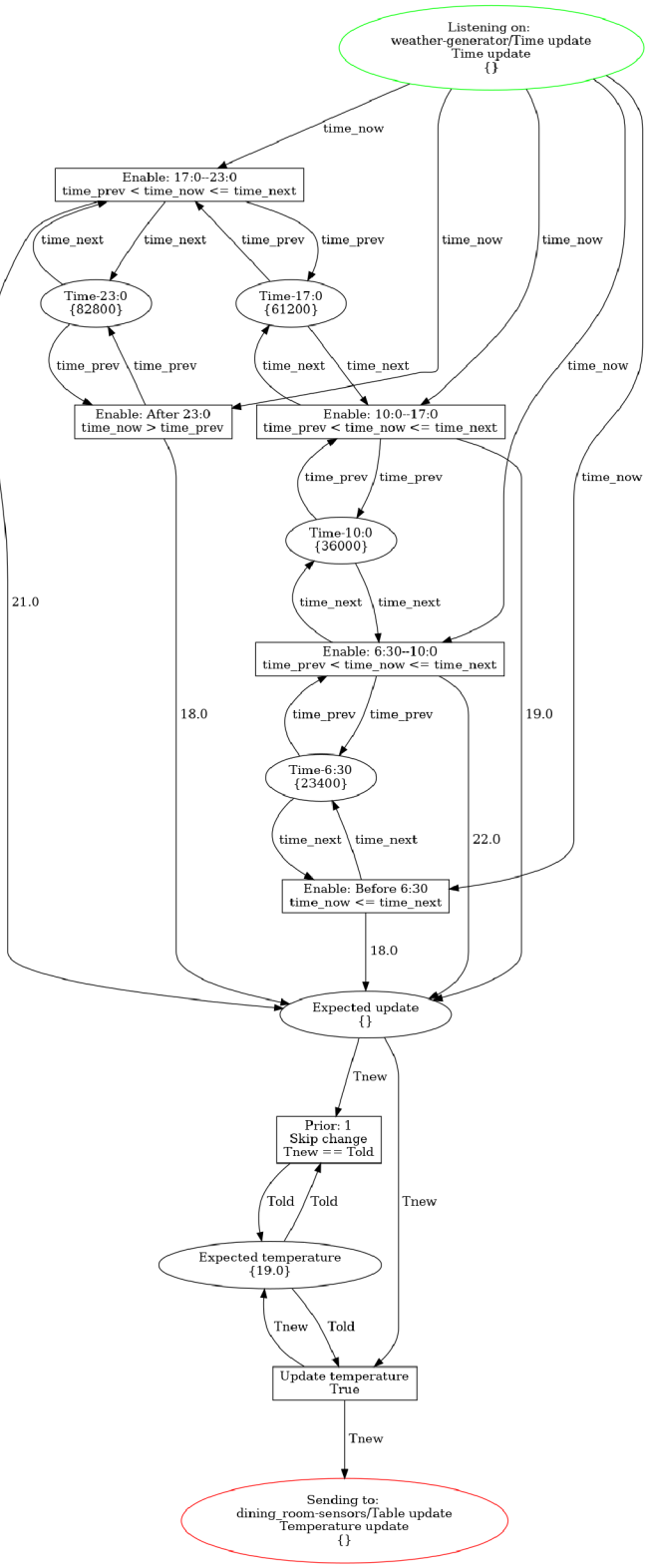
Obrázek 6.2: Převod energii na stupnici celsia.



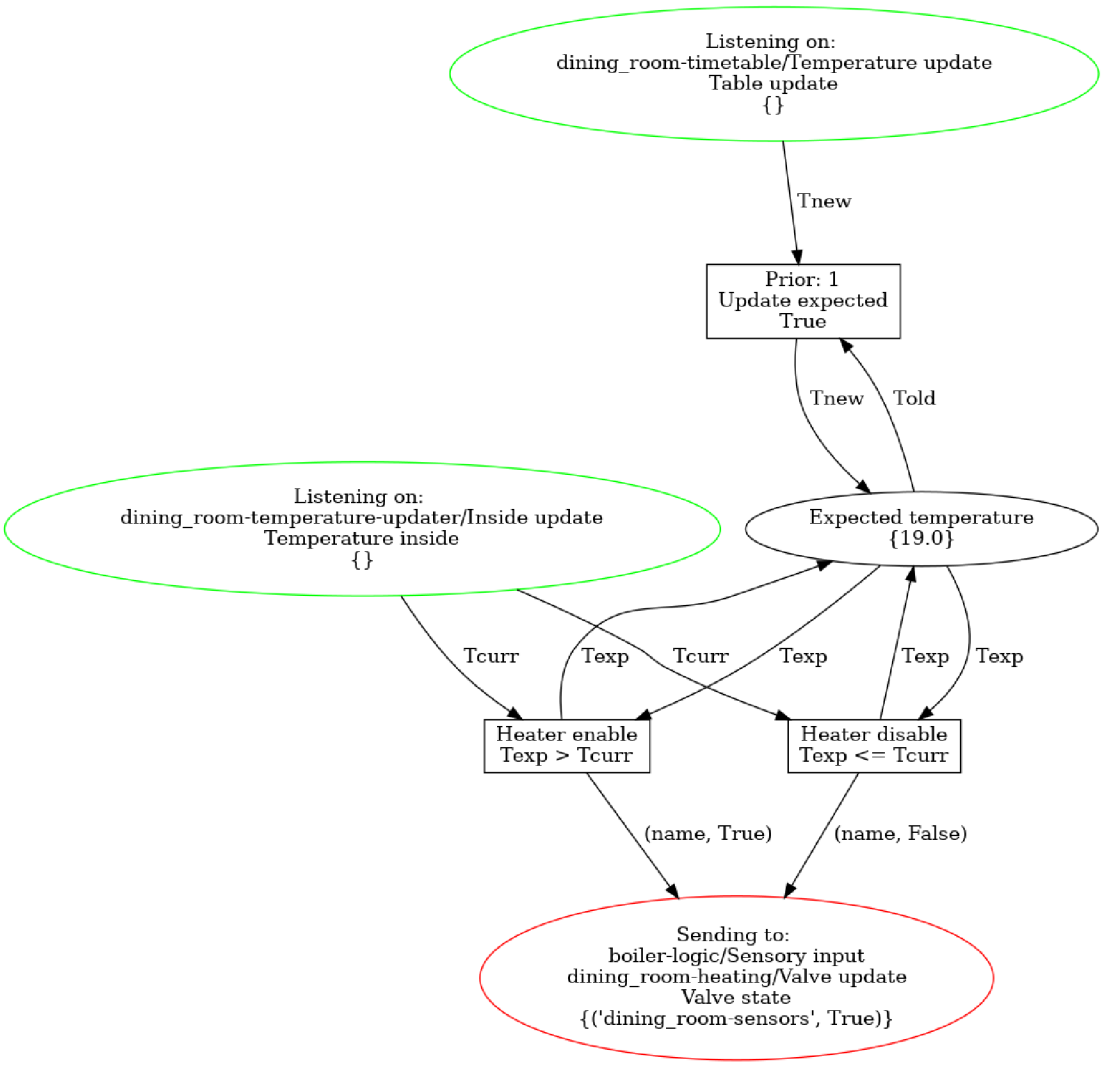
Obrázek 6.3: Simulace ohřívání pokojů.



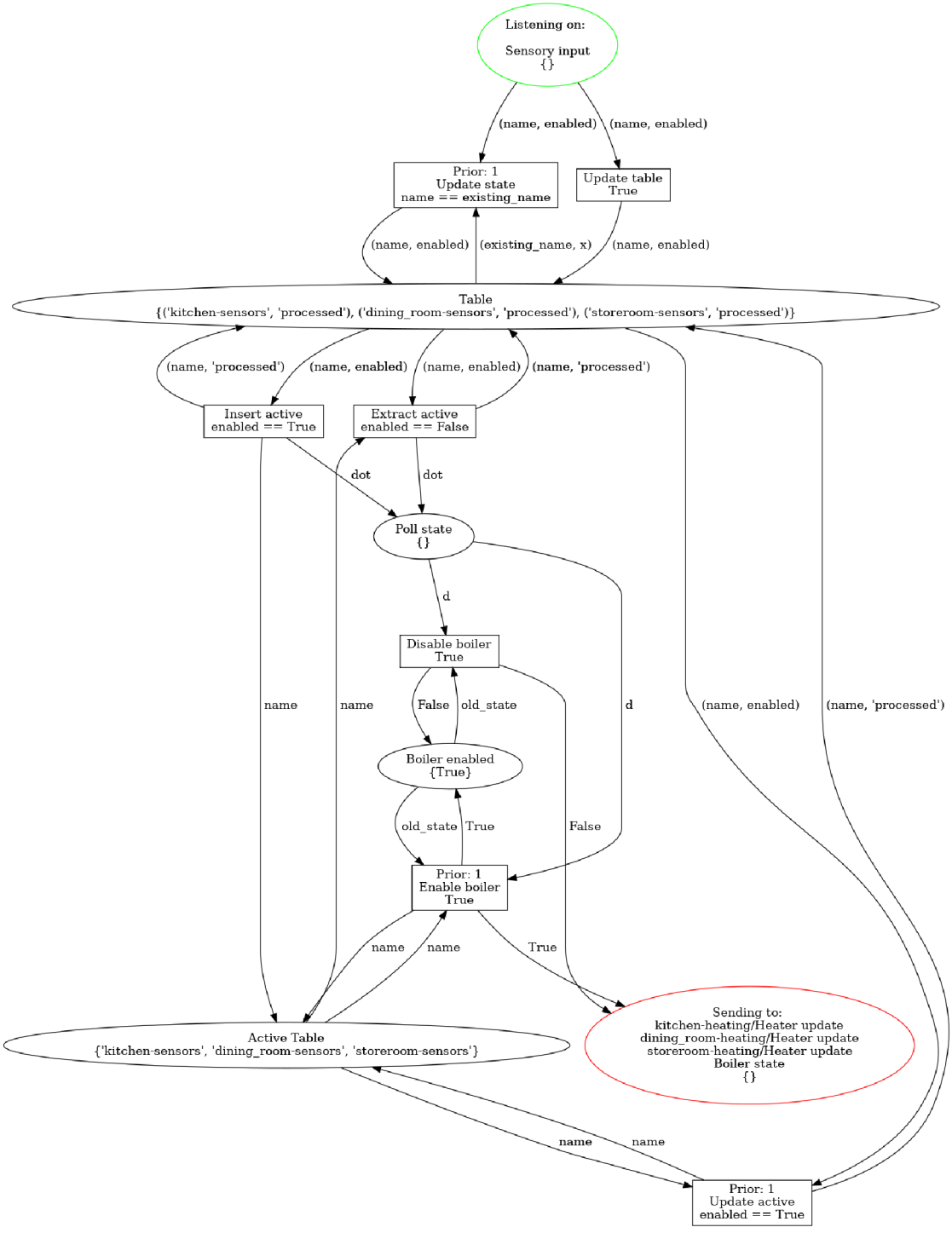
Obrázek 6.4: Simulace výměny teploty se vnějším prostředím.



Obrázek 6.5: Tabulka teplot pro pokoj.



Obrázek 6.6: Simulace senzorů a stavu ventilů v pokoji.



Obrázek 6.7: Simulace bojleru.

Kapitola 7

Detailní popis implementace Petriho sítě v rámci simulátoru.

V této kapitole je detailní popis vytvoření Petriho sítě, simulující chování počasí během dne 6.1.

7.1 Měření teploty venku

Modelování chování teploty venku je stochasticky proces, který není možné spočítat přesně. Ale ve snaze její simulace byla navržena Petriho síť, využívající pravděpodobnostní přechody. Má na vstupu rozsah teplot den – noc a podle vybrané doby dne, za použití funkce *cos* modeluje změnu aktuální teploty. Rozhodnutí použití funkce *cos* vychází z jednoduchého pozorování průběhu teplot během dne a periodicita změny teploty se jí trochu podobá za předpokladu, že průběh funkce začíná v nejteplejší hodinu během dne, což je počáteční bod funkce *cos*. Ale základní cíl použité funkce je modelování spojitého průběh teplotních změn.

Pravděpodobnostní přechody modelují náhodné jevy jako oteplení nebo ochlazení kvůli východu slunce a jiné. Ukázku implementace můžete vidět v 7.3.

Vizuální reprezentací celé sítě můžete vidět na obrázku 6.1.

Obrázek 6.1 byl vygenerován v průběhu práci simulátoru, díky použití rozšíření "gv". Toto rozšíření, spolu s menšími vizuálními změnami jiných rozšíření, zaměřených na jednotlivé prvky sítě, dokáže spolehlivě vygenerovat obrázek Petriho sítě, podobný 6.1, v jakýkoliv okamžik simulace. Pro vytvoření obrázku je specifikován jednoduchý příkaz `draw("file.png")`.

Tato síť je poměrně jednoduchá na pochopení, a zároveň názorně ukazuje použití většiny naimplementovaných rozšíření. Celý postup pro vytvoření 6.1 se sestavuje z následujících kroků – 7.2.

7.2 Kroky postupů

Krok 1: Je důležité si nejdříve naimportovat rozšíření a knihovnu SNAKES podle 1. Každá síť si vytváří klauzuli `PetriNet('nazev site')`. Dále se do sítě musí přidat místa, přechody, a případně vstupní a výstupní porty, při použití nastavení podle 4.2.1.

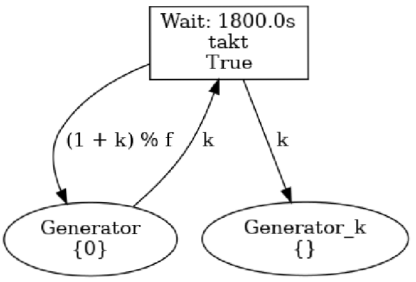
V případě potřeby se dá vytvořená síť vykreslit pomocí příkazu `draw(net_name+'.png')`

```
1 # Import pluginu, knihovny SNAKES, viz 1
```

```
net_name="Thermometr"
n=PetriNet(net_name)
... # Specifikace prechodu a mist: 7.1, 7.2 a 7.3
n.draw(f"{path_to_img}/{net_name}.png")
```

Krok 2: Dále, abychom si namodelovali chování teploty během dne, musíme vytvořit generátor, který rozdělí den častí. V případě této implementace je to 48 – 2. Tedy každých 1800s se vytvoří nová teplota – 16, což je použití rozšíření `timed_pl` z 4.3.1. Výsledek je na obrázku 7.1.

```
# Sit 7.1
f=48 # Frekvence mereni behem dne
T=24*60*60/f # Perioda zmen teploty
k=0 # Citac pro generator
timeout=T
 # Deklarace globalnich promenych v~ramci site
n.declare(f"f={f}")
 # Vytvoreni mist
gen=Place("Generator", [k], check=tInteger)
gen_k=Place("Generator_k", [], check=tInteger)
# Pridani mist do site
n.add_place(gen)
n.add_place(gen_k)

# Vytvoreni prechodu
takt=Transition("takt", timeout=timeout)
takt.add_input(gen, Variable("k"))
takt.add_output(gen, Expression("(1 + k) % f"))
takt.add_output(gen_k, Variable("k"))
# A~pridani do site
n.add_transition(takt)
```



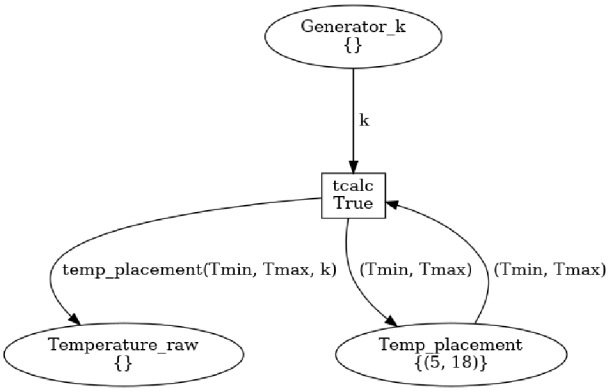
Obrázek 7.1: Generator doby dne.

Krok 3: Pro vytvoření teploty pro daný okamžik musíme provést mapování na teplotní funkci 6. To se provádí na výstupu z `Transition("tcalc")`, a je povolené podle pravidel anotaci výstupních šípek. Funkce vrací číselnou hodnotu v rozmezí $5 - 18\,[C°]$, která se vloží do místa `Place("Temp_placement", [(low, high)], check=tTuple)` z 15.

Výsledná síť je na obrázku 7.2.

```
# Sit 7.2
# Globalni promene
```

```
3  low, high=5, 18
4  # Deklarace globalni knihovny a funkce pro vypocet teploty
5  n.declare("from math import cos, pi")
6  n.declare("def temp_placement(Tmin, Tmax, k):"
7  "\n\tif k~== 0:"
8  "\n\t\treturn float(format(Tmax, \".1f\"))"
9  "\n\telse:"
10 "\n\t\treturn float(format("
11 "Tmin + (Tmax-Tmin)*(cos(2*pi*k/f)/2 + 1/2),"
12 "\".1f\"))")
13
14 # Vytvoreni mist
15 temp_pl=Place("Temp_placement", [(low, high)], check=tTuple)
16 traw=Place("Temperature_raw", [], check=tFloat)
17 n.add_place(temp_pl)
18 n.add_place(traw)
19
20 # Pridani prechodu
21 tcalc=Transition("tcalc")
22 tcalc.add_input(gen_k, Variable("k"))
23 tcalc.add_output(traw, Expression("temp_placement(Tmin, Tmax, k)"))
24 tcalc.add_input(temp_pl, Tuple((
25   Variable("Tmin"), Variable("Tmax"))))
26 tcalc.add_output(temp_pl, Tuple((
27   Variable("Tmin"), Variable("Tmax"))))
28 n.add_transition(tcalc)
```



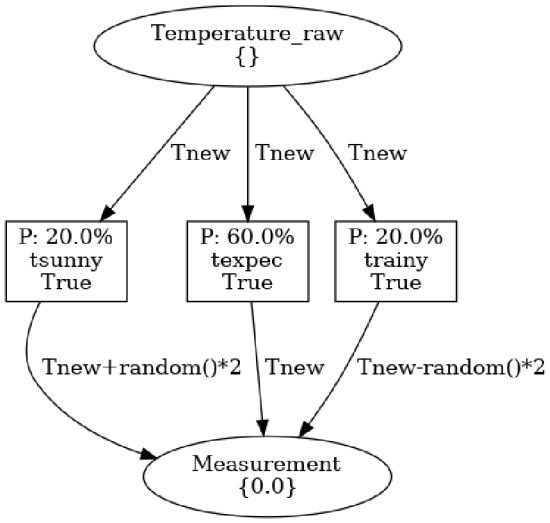
Obrázek 7.2: Výpočet teploty pro danou dobu dne.

Krok 4: Tato síť je zaměřena na simulaci počasí z pohledu zdí budovy, která se čelí vnějšímu prostředí. Simulace je v tomto případě hrubou, ale dostačující aproximaci reálného světa.

Pro simulaci různých náhodných jevů slouží 2. Pro případ, že teplota přesně odpovídá očekávané, je přechod `texpec=Transition("texpec", prob=0.6)` z 14. Ten má pravděpodobnost 0.6. Pak jsou stejné pravděpodobnosti 0.2 pro případ deště nebo slunce, které svítí na zeď a jsou to řádky 13 a 14. Součet pravděpodobnosti musí být vždy 1. Pro seskupení takových přechodů slouží řádek 16 a 17. Přechody musí mít stejnou sadu vstupních míst, jak je to popsané v 4.3.2.

Síť vypadá následovně: 7.3.

```
# Sit 7.3
# Import globalnich knihoven
n.declare("from random import random")

# Pridani vystupnich mist
meas=Place("Measurement", [0.], check=tFloat)
n.add_place(meas)

# Nastaveni mista jako vystupni port
n.add_remote_output(meas, "temp_sens/Input temp")

trainy=Transition("trainy", prob=0.2)
tsunny=Transition("tsunny", prob=0.2)
texpec=Transition("texpec", prob=0.6)
# Spojeni prechodu do skupiny
trainy.add_neighbour_transition(texpec)
trainy.add_neighbour_transition(tsunny)

# Nahodna zmena teploty vedouci k~ochlazeni
trainy.add_input(traw, Variable("Tnew"))
trainy.add_output(meas, Expression("Tnew-random()*2"))
n.add_transition(trainy)

# Teplota podle ocekavani
tsunny.add_input(traw, Variable("Tnew"))
tsunny.add_output(meas, Expression("Tnew+random()*2"))
n.add_transition(tsunny)

# Zvyseni teploty na porovnani s~ocekavanou
texpec.add_input(traw, Variable("Tnew"))
texpec.add_output(meas, Variable("Tnew"))
n.add_transition(texpec)
```



Obrázek 7.3: Modelování náhodných změn teplot.

Kapitola 8

Závěr

Během čtení této bakalářské práce byla čtenářovi nabídnuta možnost pochopit teoretické základy Petriho sítí, zjistit rozdíl mezí značkovanými a vysokoúrovňovými Petriho sítěmi a nahlédnout do případů, kde se tento modelovací prostředek úspěšně používá nebo má předpoklady pro uplatnění. Byly taky ukázaný teoretické základy Grafů Vysokoúrovňových Petriho sítí, které našli využití v dalších kapitolách, spojených s vizualizací navržené aplikace simulatoru.

Byla vyjasněna problematika Distribuovaných systémů, a taky kde a proč se takové systémy používají. Byl vysvětlen pojem Diskrétní simulace a simulace v reálném čase, včetně jejich běžných případů využití v praxi. V sekci spojené s teoretickými základy simulátoru bylo taky řečeno o jeho využití v HWIL. Ve stejné sekci byl podrobně popsán MQTT protokol komunikace, který našel svoje využití v mnoha místech, včetně implementací simulátoru.

Další kapitola vyjasnila struktura simulatoru, využívající MQTT protokol. Značná část této kapitoly byla zaměřená na vysvětlení práce vlastního protokolu pro nastavení a využití simulátoru, který se dá uplatnit pro napsaní externích aplikací pro výsledný simulátor. Stejně tak zde byla velice podrobně popsána implementace několika rozšíření pro knihovnu SNAKES, bez kterých využití simulačního nástroje není možné. Z toho důvodu byl zdůrazněn postup pro import těchto rozšíření do knihovny SNAKES.

Stejně jako bez rozšíření, simulátor Petriho sítí v reálném čase by nebyl možný bez implementace plánovače událostí, informace, která byla popsána v další sekci.

Následující kapitola měla za úkol vyjasnit návrh aplikace, demonstrující využití simulátoru pro modelování a simulaci distribuovaných systémů na příkladě chování tepelného řízení. Spolu s kapitolou 6 se zcela podrobně popsala práce navržené aplikace.

Poslední kapitolou této práce byla ukázka implementaci jedné z Petriho sítí, patřící do aplikace, která demonstrovala využití všech navržených rozšíření a měla za účel naučit uživatele správně zacházet s knihovnou SNAKES, a tedy implementovat svoje vlastní nápady. Pro tento účel slouží appendix A, který obsahuje instalační manuál a popisuje nástroje, usnadňující nastavení simulátoru do 2 řádku kódu.

Svoji práci tímto ukončuji a považuji za zcela hotovou. Byly splněny všechny body zadání, naimplementované a otestované všechny nástroje a dodrženy všechny pravidla strukturování textu, patřící technické zprávě, podle standardů VUT.

Literatura

[1] Banks, A.; Gupta, R.; Cohn, R. J.; aj.: MQTT Version 3.1.1. 2014, [Online; accessed 28-April-2019].
URL
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/csprd02/mqtt-v3.1.1-csprd02.pdf

[2] High-level Petri Nets - Concepts, Definitions and Graphical Notation. Standard, International Organization for Standardization, 1214 Vernier, Geneva, Switzerland, October 2000.

[3] Peterson, J. L.: *Petri Net Theory and The Modeling of systems*. Prentice-Hall, Inc., 1981, ISBN 0-13-661983-5.

[4] Pommereau, F.: The SNAKES toolkit. 2019, [Online; navštíveno 28. 04. 2019].
URL https://www.ibisc.univ-evry.fr/~fpommereau/SNAKES/

[5] S.Tanenbaum, A.; Steen, M. V.: *Distributed systems: Principles and Paradigms*. Pearson Education, Inc., 2007, ISBN 0-13-239227-5.

[6] Wikipedia contributors: Dining philosophers problem — Wikipedia, The Free Encyclopedia. 2019, [Online; accessed 28-April-2019].
URL https://en.wikipedia.org/w/index.php?title=Dining_philosophers_problem&oldid=890834797

[7] Wikipedia contributors: Hardware-in-the-loop simulation — Wikipedia, The Free Encyclopedia. 2019, [Online; accessed 28-April-2019].
URL https://en.wikipedia.org/w/index.php?title=Hardware-in-the-loop_simulation&oldid=894087685

[8] www.komplektacya.ru: Výpočet tepelného výkonu. 2018, [Online; navštíveno 21. 04. 2019].
URL https://www.komplektacya.ru/spravochnik/teplovoe-oborudovanie1/raschet-teplovoj-moschnosti

Příloha A

Instalační manuál

A.1 Instalace simulátoru

Pokud byste potřebovali odzkoušet simulační nástroj z této práci, tak bych doporučoval provést této kroky instalace.

1. Nainstalovat si balíčky `python3.6`, `graphviz`, `python3-pip`, `git`.
2. Nainstalovat knihovnu SNAKES podle nějakého z postupu v návodu. Doporučený postup, nainstalovat to přes pip:

   `pip3 install git+git://github.com/fpom/snakes --user`.
3. Nainstalovat této knihovny přes `pip3 install xxx --user` pro podporu MQTT, kde `xxx` je:

   - `paho-mqtt`
   - `hbmqtt`
4. Naklonovat git repositář simulační knihovny do libovolného místa na disku:

   `https://github.com/Danil-Grigorev/rt-sim-petry-net`.
5. Provést potřebné úpravy pro rozšíření "gv" v poslední verzí knihovny SNAKES, a nakopírovat jiné rozšíření do složky pro rozšíření. Tá se obecně po kroku 2 nachází v `~/.local/lib/python3.6/site-packages/snakes/plugins`, ale může se to lišit v závislosti od nastavení balíčku Python. Seznám rozšíření pro kopírovaní/nahrazení ze složky `rt-sim-petry-net/plugins` je následující:

   - `prior_pl.py`
   - `prob_pl.py`
   - `sim_pl.py`
   - `timed_pl.py`
   - `gv.py`

Alternativní postup je nainstalovat si závislosti z `pip3` pomoci pythsetuptools. Stačí spustit příkaz `python3 -m pip install -r requirements.txt --user`. Pří použítí `python3-venv`, nemusíte uvádět `--user` po každém z příkazů.

A.2 MQTT broker

V mém repositáři se taky nachází konfigurační soubor pro případ, že byste neměli k dispozici MQTT broker. Je to soubor hbmqtt.conf, a abyste takový server spustili na lokálním počítači, stačí k tomu příkaz `hbmqtt -c hbmqtt.conf`.

Pro instalaci tohoto nástroje, stačí spustit příkaz `pip3 install hbmqtt`.

A.3 Šablona pro simulator

Běžný způsob použití simulátoru předpokládá využití třídy `PNSim`, ale pro zjednodušení jsem se rozhodl vytvořit šablonu pro spuštění simulátoru. Použití je snadné: 1.

```
1  # Import metod ze sablony
2  from template import *
3  nets =[]# Seznam Petriho siti pro simulaci
4  ... # Implementace Petriho siti a pridani instanci do seznamu
5  execute_nets(nets, sim_id='server') # Spusteni simulacni instance pod nazvem 'server'
```

A.4 Ukázka externí aplikace

Postup pro vytvoření takové aplikace se nachází v kapitole 4.2.3, výsledný kód z komentáři je součástí tohoto souboru.

A.5 Zdrojové kódy pro aplikaci

V kapitole 5 a 6 je detailní popis navržené aplikace. Zdrojové kódy se nacházejí ve složce "`sample_nets`".

Pro import a následující použití stačí provést příkaz `from sample_nets import *`.

Jsou dvě možností pro spouštění aplikace – buď distribuovaně podle návrhu z 5.1, kdě aplikace je rozdělena na častí v do souborů `dining-room.py`, `storeroom.py`, `surroundings.py`, `kitchen.py` a `server.py`, nebo pro demonstrační účely slouží varianta `all-in-one.py`.