VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MULTIAGENTNÍ SYSTÉM UČÍCÍ SE MAXIMALIZOVAT KOMFORT UŽIVATELŮ V RÁMCI SMART HOME

MULTI-AGENT SYSTEM LEARNING TO MAXIMISE USER COMFORT WITHIN SMART HOME

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. RADIM BEDNAŘÍK**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

141487

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Bednařík Radim, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Vestavěné systémy
Název:	**Multiagentní systém učící se maximalizovat komfort uživatelů v rámci Smart Home**
Kategorie:	Umělá inteligence
Akademický rok:	2022/23

Zadání:

1. Prostudujte problematiku multiagentních systémů a obklopující inteligence.
2. Prostudujte problematiku automatizace budov a Smart Home a existující přístupy, aplikující multiagentní systémy v řízení.
3. Ve spolupráci s vedoucím zvolte vhodný subsystém Smart Home a specifikujte požadavky na jeho inteligentní řízení. Navrhněte řešení s využitím multiagentního paradigmatu a vhodného middleware tak, aby se průběžně na základě senzorických dat, interakcí uživatele se systémem a omezujících podmínek učil maximalizovat komfort uživatelů.
4. Navrženou řidicí aplikaci implementujte s využitím vybraného multiagentního middleware.
5. Ověřte funkčnost realizované řídicí aplikace s využitím vhodné kombinace reálných a simulovaných senzorů a aktuátorů. Ověřování proběhne nasazením systému v reálném prostředí. Vyhodnoťte dosažené výsledky.

Literatura:

1. Wooldridge, M. (2009). An Introduction to Multiagent Systems. Chichester, UK: Wiley. ISBN: 978-0-470-51946-2
2. Mirra, J., Silva, F., & Analide, C. (2018). Reinforcement learning based approach for smart homes. In *Intelligent Environments 2018* (pp. 38-47). IOS Press.

Při obhajobě semestrální části projektu je požadováno:
První 3 body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Janoušek Vladimír, doc. Ing., Ph.D.**
Vedoucí ústavu:	Hanáček Petr, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	3.11.2022

Abstrakt

Tato práce je zaměřena na tvorbu multiagentního systému pro subsystém chytré domácnosti vytápění, jež se snaží učit vzory uživatelů, pomocí algoritmu posilovaného učení. Práce dále popisuje tvorbu potřebných modulů, kterými jsou digitální termostatická hlavice, figurující v systému jako koncový agent, a modul pro detekci přítomnosti osob. Vytvořený systém byl nasazen v reálném prostředí a je funkční.

Abstract

This thesis is focused on creating a multi-agent system for a smart home heating subsystem that tries to learn user patterns, using a reinforcement learning algorithm. The thesis further describes the creation of the necessary modules, which are a digital thermostatic valve, which appears in the system as an end agent, and a module for detecting the presence of people. The created system was deployed in a real environment and is functional.

Klíčová slova

Chytrá domácnost, multiagentní systém, agent, JADE, posilované učení, Q-Learning

Keywords

Smart home, multi-agent system, agent, JADE, reinforcement learning, Q-Learning

Citace

BEDNAŘÍK, Radim. *Multiagentní systém učící se maximalizovat komfort uživatelů v rámci Smart Home*. Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Vladimír Janoušek, Ph.D.

Multiagentní systém učící se maximalizovat komfort uživatelů v rámci Smart Home

Prohlášení

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

.....
Radim Bednařík
16. května 2023

Poděkování

Zde bych rád poděkoval vedoucímu mé diplomové práce doc. Ing. Vladimíra Janouška, Ph.D. za odborné vedení a konzultace při řešení problémů se psaním této práce, tak při praktické tvorbě.

Obsah

Kapitola 1

Úvod

Vznik a stálý pokrok inteligentních budov je motivován zvyšováním komfortu, bezpečí a optimalizací spotřeby energie [2]. Při počátku inteligentních budov byl tento koncept spíše odkazován na automatizaci než-li inteligenci, toto se však mění a s postupem času je kladen stále větší důraz na inteligenci systémů. Rozdílem mezi těmito principy je, že automatizace využívá komponent systému k zajištění určitého chování, kdežto inteligence se snaží přizpůsobovat parametry systému za účelem předvídání preferencí uživatele a zvýšení jeho komfortu.

Tato práce se primárně zabývá vytvořením řešení pro malou inteligentní budovu (tzv. chytrou domácnost), které disponuje místo automatizace právě inteligencí.

Cílem této práce bylo vytvoření rozšiřitelného multiagentního systému pro subsystém vytápění chytré domácnosti, a jednotlivé potřebné moduly, který s podporou posilovaného učení bude schopen se učit uživatelským preferencím, a na jejich základě ovládat aktory v prostředí pro jejich dosažení.

Obsah práce je rozdělen do několika logických celků. Druhá kapitola se zabývá agentními a multiagentními systémy, jsou zde popsány jednotlivé část takového systému, různé architektury agentů, základní principy multiagentního systému, a je zde také stručně představen framework pro tvoření multiagentních systémů JADE. Další kapitola se věnuje základům posilovaného učení a představuje některé jeho přístupy. Čtvrtá kapitola představuje některé již existující řešení implementující multiagentní systém pro inteligentní budovy a chytré domácnosti. Následující dvě kapitoly pojednávají o návrhu a implementaci jednotlivých modulů, které byly v rámci práce vytvořeny. A předposlední kapitola popisuje způsob a výsledky ověřování jednotlivých modulů, tak i celého systému v reálném prostředí.

Kapitola 2

Agentní a multiagentní systémy

Velkou částí této diplomové práce je téma multiagentních systémů. V této kapitole se proto budu věnovat základnímu popisu a principů agentních a multiagentních systému [18][17].

Nejprve je popsána architektura agentního systému a jeho prvků. Následuje část popisující abstraktní architekturu inteligentního agenta. Dále je rozebrán popis a vlastnosti multiagentních systémů, a chování agentů v takovém systému. A poslední je zmíněn multiagentní framework JADE[1] pro programovací jazyk Java[2].

2.1 Agentní systém

Agentní systém se skládá ze dvou základních prvků, z prostředí a z agenta, jež je v prostředí situován a má dán určitý úkol, který mu byl zadán klientem (člověkem) a kterého se snaží za dobu svého chodu dosáhnout.

2.1.1 Agent

V tomto kontextu se používá výraz umělý agent, což je člověkem vytvořené dílo, jež je zasazeno do prostředí za účelem autonomního konání ve prospěch svého klienta. Důležitou vlastností agenta je tedy samostatné chování, toto je nejzásadnější rozdíl oproti objektům v objektově orientovaném programování.

Existuje několik definic, co racionální agent je, a jaké vlastnosti by měl mít. Jednou z těchto definic je Jenningsova klasifikace. Vlastnosti racionálního agenta definované dle Jenningsovy klasifikace je následovná:

- **Autonomita** – Agent koná autonomně, pokud koná samostatně a nezávisle (bez přímého vlivu z okolí) na základě svého rozhodnutí, za určitých okolností se agent může své autonomity vzdát.
- **Reaktivita** – Agent je schopen reagovat na změny v prostředí adekvátně a pohotově, tak aby dosáhl předepsaného cíle.
- **Proaktivita** – Agent je schopen projevit iniciativu a působit na prostředí způsobem, který jej přibližuje k jeho cílům.

[1] `https://jade.tilab.com`
[2] `https://www.java.com/en/`

- **Sociální schopnosti** – Agent má schopnost sociálního chování, komunikace ve skupině agentů. Agenti mohou spolupracovat za dosažením společného cíle, mohou řešit konflikty při protichůdných cílech, nebo tvořit skupiny, aby lépe hospodařili s kritickými zdroji v prostředí.

2.1.2 Prostředí

Prostředím chápeme část světa, ve které se agent nachází a vykonává v něm svou úlohu. Prostředí lze kategorizovat následovně:

- **Dostupné versus nedostupné** – V prostředí jež považujeme za dostupné, je agent schopen vnímat veškeré informace, které prostředí nabízí, dokáže získat úplný a přesný obraz o jeho stavu. Zatímco v nedostupném prostředí je agent schopen vnímat pouze část, podmnožinu prostředí, ve kterém je zasazen. Pokud pracujeme v reálném světě, tak se téměř vždy nacházíme v nedostupném prostředí, protože většinou existují externí vlivy, které nejsme schopni měřit.

- **Deterministické versus nedeterministické** – Deterministické prostředí je takové, ve kterém každá dvojice, akce agenta a stav prostředí, má pevně dán následující stav. V nedeterministickém prostředí existuje jakási náhoda, nejčastěji mluvíme o stochastickém prostředí, tedy že akce může vést do různých stavů s danými pravděpodobnostmi.

- **Statické versus dynamické** – Ve statickém prostředí se v čase jeho stav neměnní, pokud agent nacházející se v něm nevykoná nějakou akci, stav prostředí je plně závislý na agentovi. Dynamické prostředí je naopak nepředvídatelné bez modelu, který by jej popisoval, a v čase se může samovolně měnit.

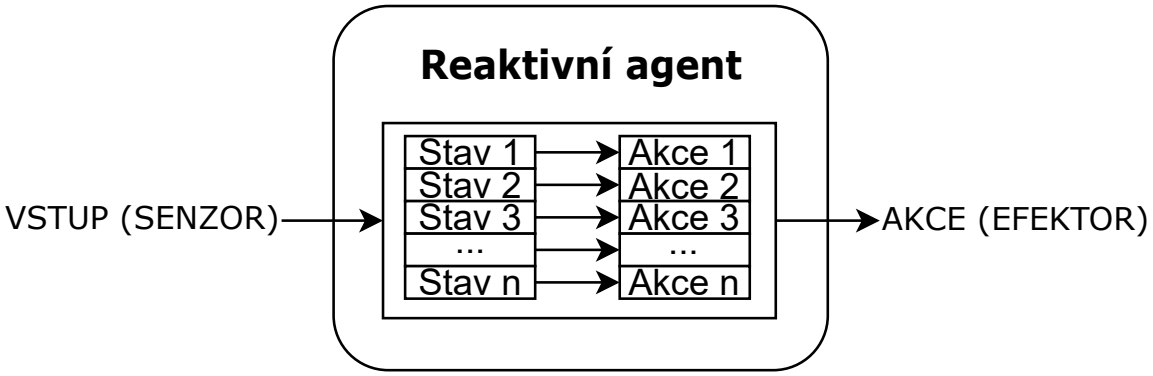
- **Diskrétní versus spojité** – V diskrétním prostředí existuje pouze konečná/spočetná množina stavů, vjemů a akcí. Naopak ve spojitém prostředí je tato množina nekonečná/nespočetná. Většina reálných prostředí se řadí do kategorie spojitých, avšak i do těchto prostředí se většinou zasazují agenti, kteří okolí vnímají diskrétně, protože navrhnout agenta, který by vnímal okolí jako spojité, by bylo velice náročné.

2.2 Architektury agentů

Jak bylo ukázáno v části o agentech 2.1.1, agent může nabývat různých vlastností, které definují jeho schopnosti a chování v systému. Představený racionální agent, ačkoliv disponuje nejvíce schopnostmi a jeví se jako nejlepší možnost pro veškerá možná použití, tak tomu tak v realitě není, protože vytvořit racionálního agenta, aby splňoval již definované vlastnosti a zároveň požadavky na chod v prostředí je velice obtížné. Mnohdy pro zadanou úlohu ani racionálního agenta nepotřebujeme, při práci v omezeném prostředí, které je statické, je vlastnost reaktivity k ničemu a vetší roli hraje schopnost plánování, jak nejefektivněji dosáhnout cíle. V některých případech jsou vlastnosti protichůdné, když chceme systém pracující v reálném čase, tak je třeba vyzdvihnout reaktivnost, a co nejvíce potlačit plánování, aby reakce agenta byla svižná a tedy odpověď/akce na daný podnět z prostředí přišla v čas, když je ještě smysluplná [3].

2.2.1 Reaktivní

Architektura reaktivního agenta je založena na přímém mapování stavů prostředí na akce agenta. Reaktivní architektura je realizována pomocí množiny senzorů, vnímající prostředí a množiny efektorů, schopných měnit prostředí, kde vnímání prostředí agenta je mapováno, pomocí agentovi množiny pravidel, na akci efektorů, jak lze vidět na obrázku 2.1.



Obrázek 2.1: Architektura reaktivního agenta

Nejznámější reaktivní architekturou je takzvaná sumbsumpční architektura[3], který bývá považována za nejlepší příklad čistě reaktivního návrhu agenta. Subsumpční architektura je realizována konečným stavovým automatem a je koncipována několika vrstvami, kde každá má na vstupu nějaký senzor a mapuje stav na nějakou akci, kde více akcí z různých vrstev je možné provádět zároveň.

Velikou výhodou reaktivní architektury je její snadný návrh, robustnost díky jednoduchosti architektury a snadná reprodukovatelnost chování (ladění, sledování chování atd). Architektura sebou nese i nevýhody, kterými mohou být neschopnost plánování, a tudíž pro některé složitější problémy je nepoužitelná.

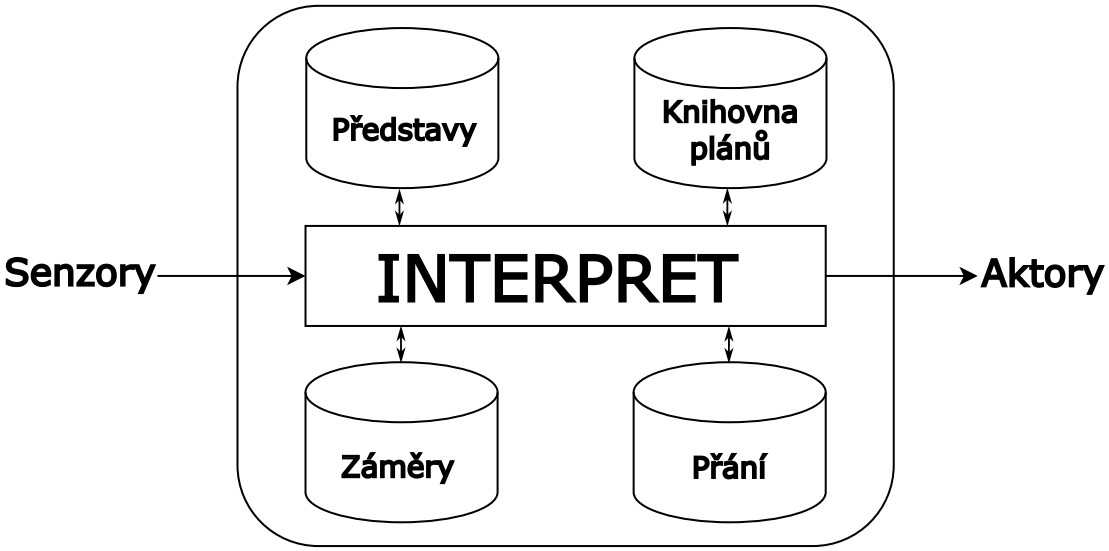
2.2.2 Deliberativní/BDI

Hlavním principem deliberativní architektury je praktické odvozování, což je akt, který slouží ke zjištění, co je potřeba udělat. Praktické odvozování se skládá ze dvou činností, ze zvažování (anglicky deliberation, proto deliberativní) a plánování. Úkolem zvažování je zvolit stav, do kterého je potřeba se dostat, a plánování rozhoduje, jakým způsobem se do toho stavu dostat.

BDI agenti se skládají ze tří logických komponent (viz. obrázek 2.2) označovaných jako mentální stavy, těmi jsou představy, přání a záměry (anglicky beliefs, desires, intentions). Představy jsou množina informací, které má agent o stavu prostředí v daném čase. Přání reprezentují stavy, kterých by chtěl agent dosáhnout. Záměr je výsledkem praktického odvozování, jedná se o přání, ke kterému se agent do jisté míry zavázal, že jej splní.

Výhodou deliberativní architektury je jasně definovaný a intuitivní návrh. Architektura je také schopná plnit složitější úlohy, díky schopnosti plánování. Pro správné fungování je však potřeba agenta implementovat způsobem, aby závazky nebral za absolutní, ale také sledoval, zda jsou jeho záměry stále validní a dosažitelné, a zároveň netrávil příliš času nad zvažováním svých záměrů, aby byl schopen daný záměr splnit.

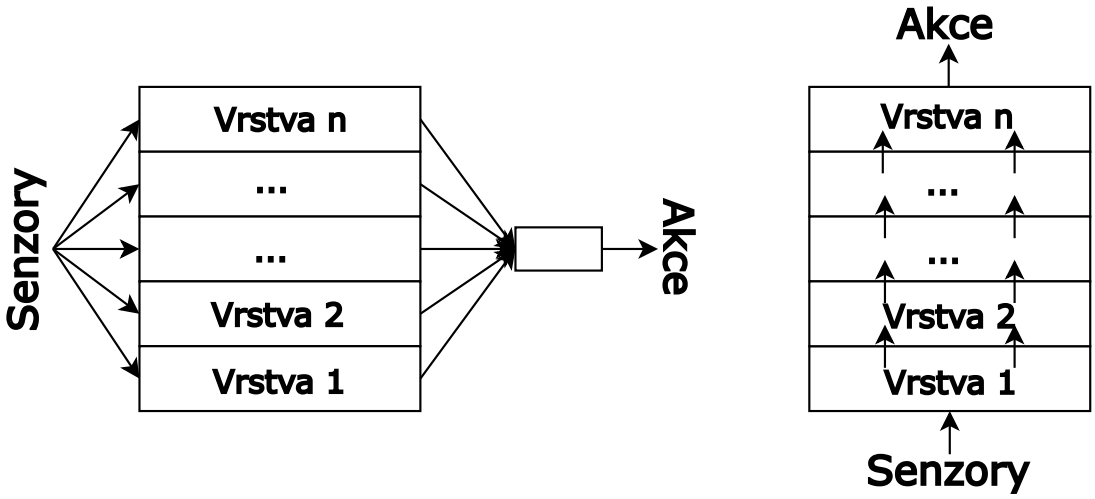
[3]Brooks, R.A., Intelligence without representation, Artificial Intelligence 47 (1991), 139–159. Dostupné z: `https://people.csail.mit.edu/brooks/papers/representation.pdf`



Obrázek 2.2: BDI architektura systému PRS[4]

2.2.3 Vrstvená/Hybridní

Vrstvená architektura je dekomponována do subsystémů, reprezentovaných vrstvami, které mohou být organizovány do hierarchické struktury, kde každá zodpovídá za určitý druh chování, umožňuje také kombinovat reaktivní a deliberativní chování agenta. Rozlišujeme dva základní typy vrstvených architektur, horizontální a vertikální.



Obrázek 2.3: Vlevo: horizontální vrstvená architektura, Vpravo: vertikální jednoprůchodová vrstvená architektura

V horizontální vrstvené architektuře je každá vrstva propojena se senzory a jejím výstupem je akce, vrstvy jsou explicitně rozděleny do částí, které pracují samostatně se svým vstupem a tvoří svůj výstup. Vertikální vrstvená architektura je hierarchicky seřazena, kde vstup z prostředí dostává pouze nejnižší úroveň a ten se postupně zpracovává všemi vrstvami

[4]Dostupné z: `https://ieeexplore.ieee.org/document/180407`

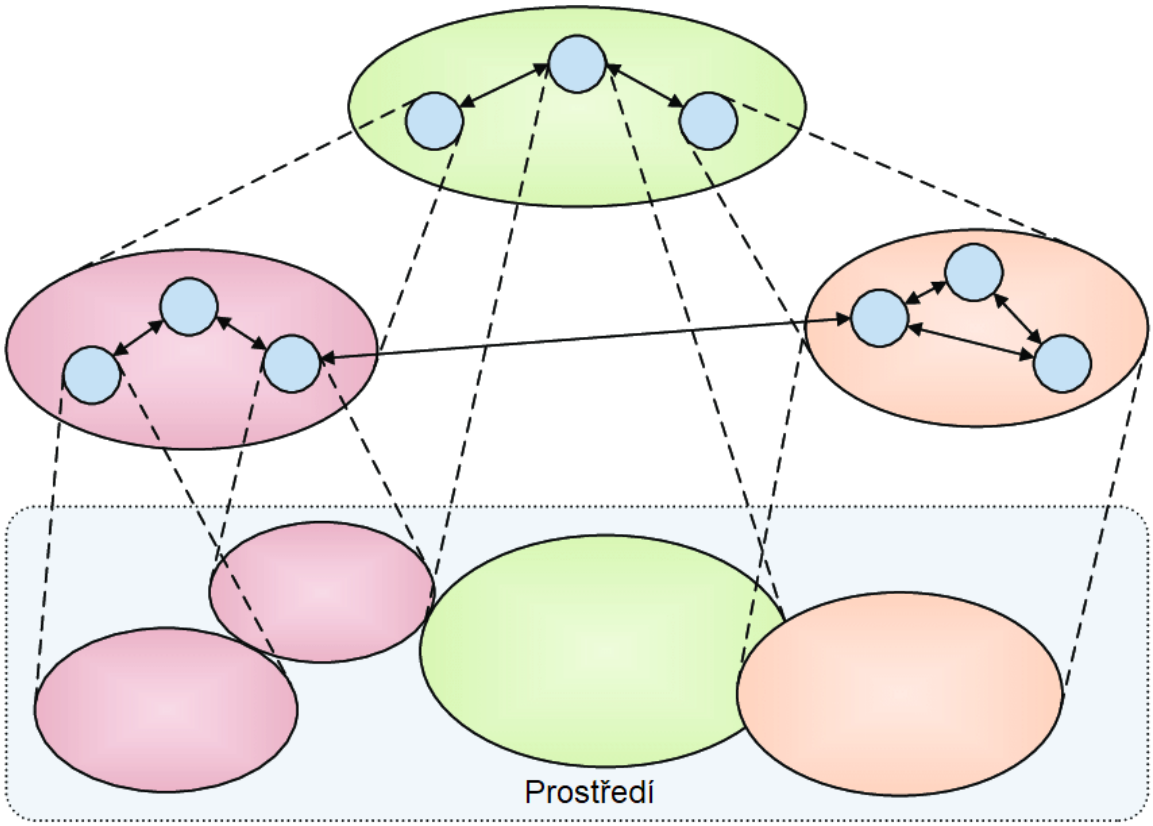
od nejnižší po nejvyšší, která z něj poté vytvoří výstup, pokud se jedná o jednoprůchodovou architekturu (viz obrázek 2.3). U dvouprůchodové architektury se data ze vstupu zpracovávají od nejnižší vrstvy po nejvyšší, stejně jako u jednoprůchodové, ale řídící tok poté jde směrem od nejvyšší po nejnižší, jejímž výstupem je poté akce.

Kombinací reaktivního a deliberativního chování je tato architektura schopna řešit velikou škálu problémů. Nevýhodou je však složitější návrh a zajištění robustnosti systému, protože systém je tak robustní, jako nejslabší jeho vrstva.

2.3 Multiagentní systém

Multiagentní systém je systém, ve kterém operuje větší množství (alespoň dva) autonomních agentů. V takovém systému má každý jednotlivý agent svou proměnlivou (na základě informací z prostředí a ostatních agentů) bázi informací a množinu definovaných cílových podmínek/stavů, kterých chce v prostředí dosáhnout. Agenti jsou schopni vnímat pouze část prostředí, ve kterém se nacházejí a také jsou schopni ovlivňovat jen pouze segment daného prostředí, tedy každý agent má danou sféru vlivu v prostředí (příklad multiagentního systému na obrázku 2.4).

V multiagentních systémech jsou agenti schopni, někdy i nuceni, mezi sebou komunikovat, ať už z důvodu výměny informací, delegace určitých činností nebo vyjednávání ohledně cesty ke společnému cíli.



Obrázek 2.4: Příklad multiagentního systému, komunikace a sfér vlivu agentů v prostředí

2.3.1 Interakce

V rámci multiagentního systému dochází mezi agenty k interakcím, tyto interakce se odehrávají z důvodu omezené vnímací schopnosti agenta, který ale potřebuje informace, ke kterým přístup nemá, je tedy potřeba, aby byl schopen o tyto informace požádat agenta, který je má. Dalším typ interakce je nutný z obdobného důvodu, tedy z omezené sféry vlivu na prostředí, a jedná se o delegaci cílů, nebo činností, tedy že agent potřebuje splnit nějakou podmínku/cíl, ale nemá k jeho splnění prostředky a proto požádá (v případě hierarchické struktury přikáže) jiného agenta o provedení nějaké činnosti, která vede do stavu, kde je cíl splněn.

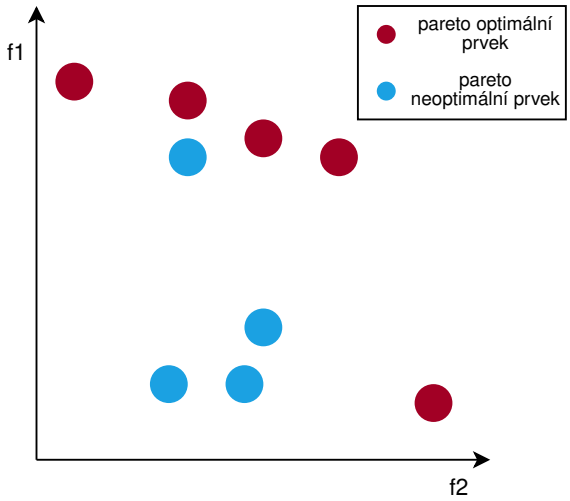
2.3.2 Vyjednávání/dohody

Protože se jedná o systém, kde se nachází několik autonomních agentů, kteří nemusí sdílet nějaký společný cíl, je nutné, aby byli schopni se mezi sebou dohodnout, jak pokračovat v jednání v prostředí, nebo jak budou v čase alokovat potřebné sdílené zdroje a další. Za tímto účelem existují různé návrhové mechanismy a protokoly, které umožňují agentům mezi sebou vyjednávat o daných problémech a docházet k dohodám, které splňují určité požadavky kladené na vyjednávací protokol.

Návrhové mechanismy

Návrhové mechanismy jsou pravidla při návrhu vyjednávacího protokolu, které vedou ke splnění podmínek, které by protokol měl na základě požadavků mít. Mezi tyto požadavky [10] můžou patřit například následující.

- **Maximalizace sociálního prospěchu** – Sociální prospěch je součet odměn všech agentů v rámci dosažené dohody, tedy měří, jak výhodná je dohoda pro multiagentní systém jako celek.
- **Paretovo optimum** - Dohoda x je pareto optimální, pokud neexistuje dohoda x' taková, že alespoň jeden agent dostane větší odměnu, než v případě x a zároveň žádný agent nedostane menší odměnu než v dohodě x. Paretovo optimum také měří celkovou výhodnost dohody vůči multiagentního systému jako sociální prospěch, avšak sociální prospěch je pouze podmnožinou paretova optima, protože zde patří pouze celkově nejlepší dohody, bez ohledu na prospěch jednotlivých účastníků, zatímco paretovo optimum bere v potaz všechna řešení, která jsou nějakým způsobem nejlepší, tedy i řešení, kde má jednotlivec největší zisk (viz. obrázek 2.5).
- **Individuální racionalita** - Jedná se o vlastnost protokolu, která udává, že pro agenta je výhodnější se zapojit do vyjednávání, než jednat samostatně. Kdyby protokol tuto vlastnost neměl, tak by agent neměl žádnou motivaci vyjednávat, a v systému by jednal vždy samostatně.
- **Stabilita** - Stabilita protokolu je vlastnost, udávající, že agent je motivován chovat se podle dosažené dohody, pokud stabilita není splněna, tak se agent bude chovat způsobem, kterým mu vzniká největší odměna, což může být chování, které je v rozporu s dosaženou dohodou.



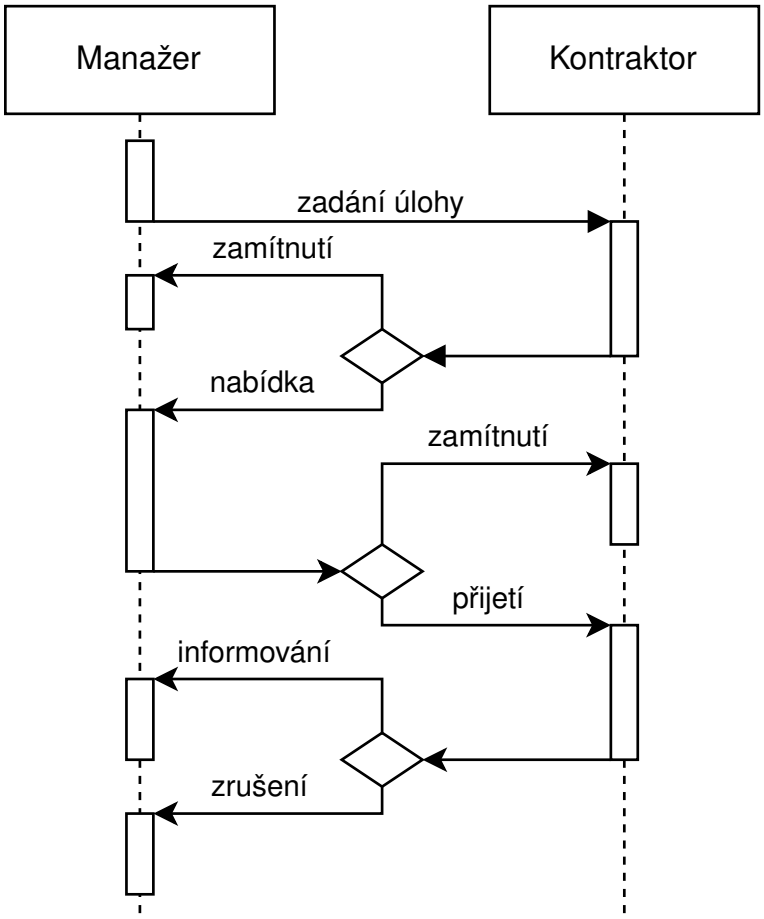
Obrázek 2.5: Příklad grafu paretova optima, kde je žádoucí oba atributy maximalizovat

Protokoly

Vyjednávací protokol je množina pravidel, které udávají jak může interakce v průběhu vyjednávání vypadat. Pravidla pokrývají povolené typy účastníků (vyjednavač, arbitr a další), stavy vyjednávání (přijímání nabídek, ukončení vyjednávání atd.), události způsobující přechody mezi stavy (nabídka přijata, žádná další nabídka) a povolené akce účastníků v daných stavech (jaký účastník může poslat zprávu v daném stavu, komu a jaký typ).

Jedním z nejběžnějších protokolů je kontrakční síť[5] (contract net protocol). V protokolu figurují dva typy účastníků, manažer a kontraktor. Nejčastějším cílem protokolu je, rozdělení práce mezi více agentů, nebo delegace práce. Kontrakční síť začíná manažer, který je pouze jeden, rozesláním zpráv s definovaným úkolem a dalšími požadavky kontraktorům, kterých je n. Kontraktor po přijetí zprávy odpoví buď s nabídkou s informacemi, na základě kterých se manažer bude rozhodovat, kterou nabídku přijme, nebo se zamítnutím. Manažer následně vybere, která nabídka je pro něj nejlepší a odešle zprávy o příjetí, nebo zamítnutí nabídky všem kontraktorům, kteří poslali nabídku. Další komunikace probíhá ze strany kontraktora, který manažera informuje o splnění úlohy a zašle případný výsledek, nebo mu zašle zrušení kontraktu, protože nebyl schopen úlohu splnit nebo dokončit (viz. obrázek 2.6).

[5]Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver,"in IEEE Transactions on Computers, vol. C-29, no. 12, pp. 1104-1113, Dec. 1980, doi: 10.1109/TC.1980.1675516.



Obrázek 2.6: Sekvenční diagram kontrakční sítě

2.3.3 Agentní komunikační jazyky

Komunikační jazyky udávají strukturu a význam (syntax a sémantiku) komunikace v multiagentních systémech, aby informace obsažené v komunikaci měly jednoznačný význam. V multiagentních systémech se používá jazyk ACL (Agent Communiation Language), nejběžněji používanými ACL jazyky jsou jazyk FIPA-ACL [1] a KQML.

KQML

KQML (The Knowledge Query and Manipulation Language) je komunikační jazyk založený na zprávách, tedy KQML definuje jednoznačný formát zpráv, je nezávislý na transportním mechanismu a není vázán na konkrétní jazyk. Zpráva jazyka KQML může být přirovnána k objektu, kde každá zpráva má performativ (lze chápat jako třídu, které je objekt instance) a několik parametrů (které lze chápat jako instanční atributy) (viz. tabulka 2.1).

Parametr	Význam
:content	Obsah zprávy
:sender	Adresa odesilatele
:receiver	Adresa příjemce
:language	Jazyk obsahu zprávy
:force	Typ zprávy
:ontology	Ontologie
:in-reply-to	Kód zprávy, na kterou tato odpovídá

Tabulka 2.1: Struktura zprávy jazyka KQML

FIPA-ACL

Jazyk FIPA-ACL (Foundation for Intelligent Physical Agents ACL) vychází z jazyka KQML, tedy se také jedná o jazyk založený na zprávách, ale má menší počet řečových aktů a větší počet parametrů zpráv (viz. tabulka 2.2). Jako příklad je uvedena jednoduchá dvou zprávová komunikace.

```
(query-if
    :sender i
    :receiver j
    :content
        (has
            (item y)
            (agent j))
    :ontology
z~:reply-with x
)
```

```
(inform
    :sender j
    :receiver i
    :content
        (not
            (has
                (item y)
                (agent j)))
    :ontology
z~:in-reply-to x
)
```

První zpráva má typ řečového aktu *query-if*, který slouží k dotazování na pravdivost nějakého tvrzení, v tomto příkladě se agent i dotazuje na tvrzení, zda agent j má ve vlastnictví předmět y. Agent j odpovídá řečovým aktem *inform*, kterým informuje o nepravdivosti dotazovaného tvrzení.

Parametr	Význam
Performative	Typ řečového aktu zprávy
Sender	Identifikátor odesilatele
Receiver	Identifikátor příjemce
Reply-to	Identifikace agenta, který obdržet odpověď na tuto zprávu
Content	Obsah zprávy
Language	Jazyk obsahu zprávy
Encoding	Specifikace kódování obsahu zprávy
Ontology	Ontologie
Protocol	Interakční protokol
Conversation-id	Identifikátor konverzace
Reply-with	Identifikátor, kterým má být označena odpověď na tuto zprávu
In-reply-to	Identifikátor odpovědi
Reply-by	Časový limit, do kdy agent čeká na odpověď na zprávu

Tabulka 2.2: Struktura zprávy jazyka FIPA-ACL

2.4 Java Agent DEvelopment Framework

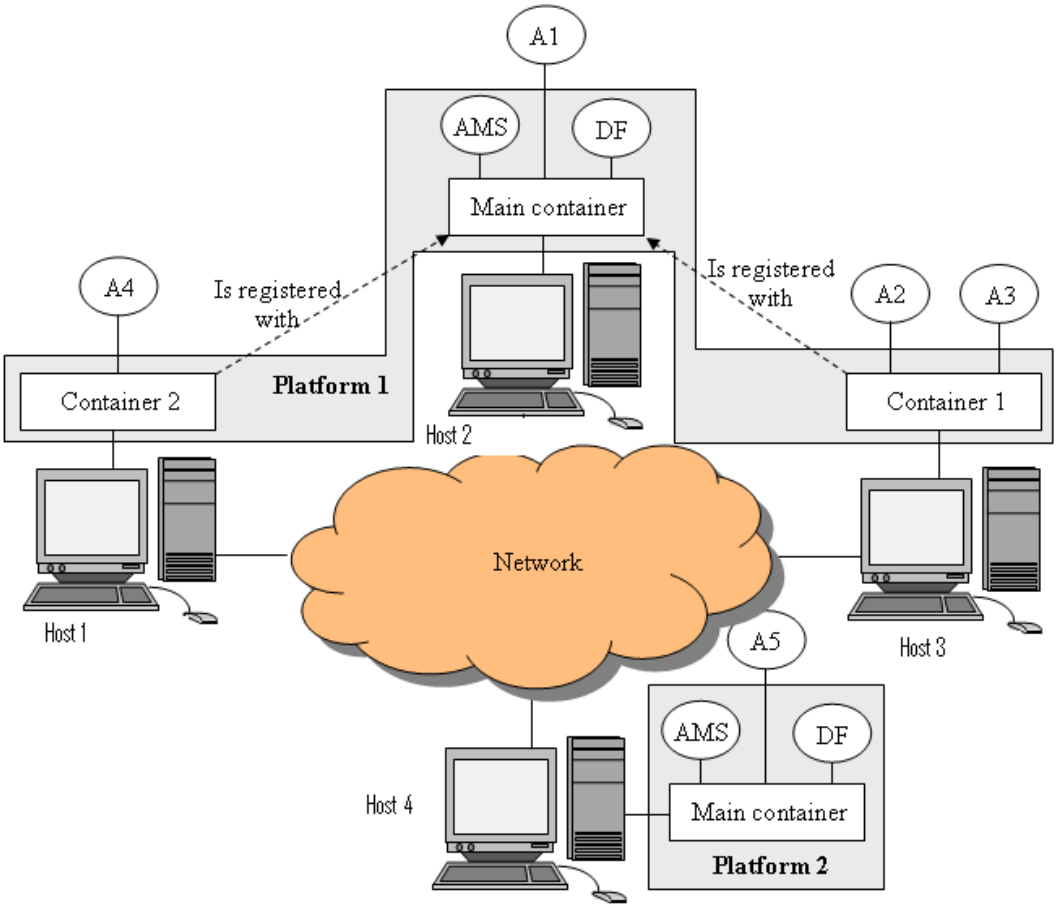
JADE je softwarová platforma (framework) [6] [5], která poskytuje základní funkcionalitu middlewarové vrstvy, která je nezávislá na aplikaci a zjednodušuje tvorbu distribuovaných aplikací, využívající abstrakci softwarových agentů. Jedním z důvodů popularity JADE platformy je její implementace v běžném objektově orientovaném jazyce Java, a tedy poskytuje jednoduché a známé API.

2.4.1 Architektura

Systém JADE obsahuje několik vrstev dekompozice, kde největší bloky tvoří platforma, platformy obsahují takzvané kontejnery, ve kterých už běží jednotlivý agenti (viz. obrázek 2.7).

Platforma je základní blok, který může být rozprostřen skrz síť a ve kterém mohou existovat kontejnery. Platformy nejsou oddělené celky, a tedy prvky v nich existující mohou mezi sebou komunikovat, ačkoliv se nacházejí v jiných platformách.

Kontejnery jsou část systému, která implementuje Java proces, který poskytuje *JADE run-time* a všechny další služby potřebné pro hostování a spouštění agentů. Na každé platformě existuje jeden speciální kontejner se jménem *Main-Container*, který je vždy vytvořen s vytvořením platformy. Všechny ostatní kontejnery se při vytvoření musí registrovat u *Main-Container*, který si uchovává tabulku všech existujících kontejnerů na platformě. Může se zdát, že modulárnost vyplývající z kontejnerů nemá žádný přínos, avšak kontejnery mohou běžet na jedné platformě a zároveň na různých strojích, tedy využívat rozdílné zdroje, což je veliká výhoda při větších a složitějších aplikacích.



Obrázek 2.7: Architektura systému JADE. Převzato z [`https://jade.tilab.com/doc/tutorials/JADEAdmin/jadeArchitecture.html`].

Agenti jsou nejmenší dílčí částí v JADE systému a jedná se o část, kterou už je třeba implementovat dle potřebné funkcionality. Avšak na každé platformě existují dva specifičtí agenti (v *Main-Container* kontejneru), kteří jsou vytvoření při spuštění platformy, jedná se o *AMS* (*Agent Management System*) a *DF* (*Directory Facilitator*).

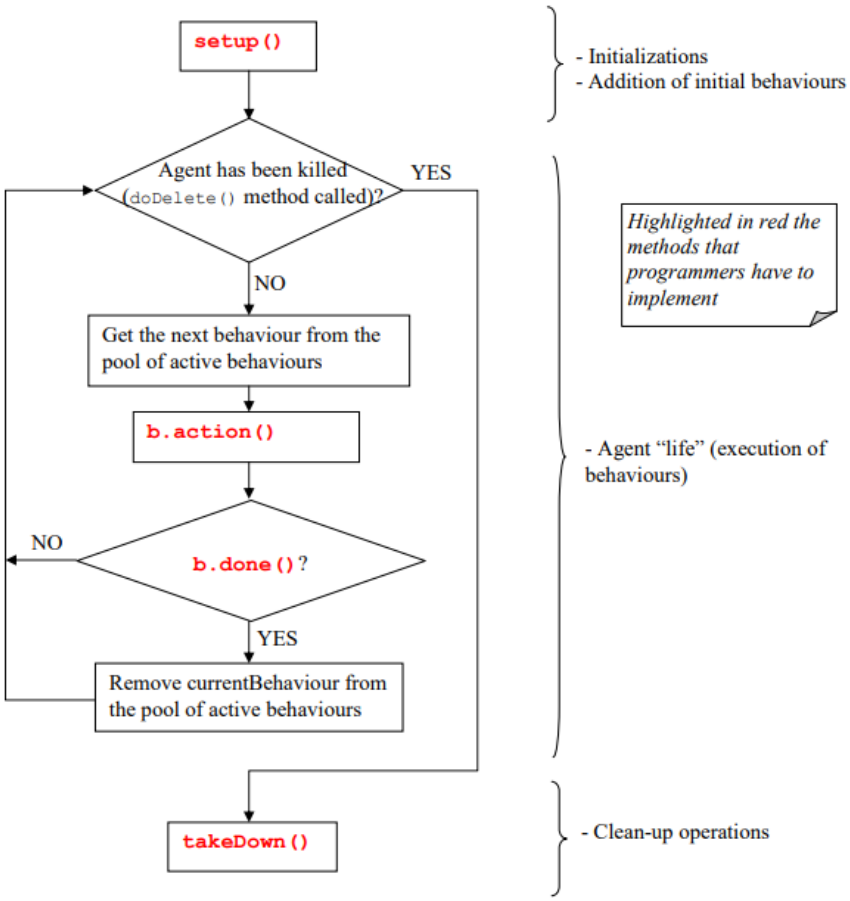
AMS je jediný agent, který je schopen provádět akce, kterými se spravuje platforma, na které se nachází. Mezi takové akce patří například vytvoření nového agenta, zabití existujícího agenta, nebo vypnutí celé platformy, ostatní agenti však mohou o provedené těchto operací u *AMS* zažádat.

DF je agent poskytující platformě službu takzvané *zlaté stránky*. Jedná se o službu, která umožňuje všem agentům dané platformy registrovat své služby a schopnosti u *DF*, a na tyto informace se také dotazovat, tedy zjistit přítomnost a identifikátor agenta s potřebnou funkcionalitou/schopností na platformě.

2.4.2 Agent

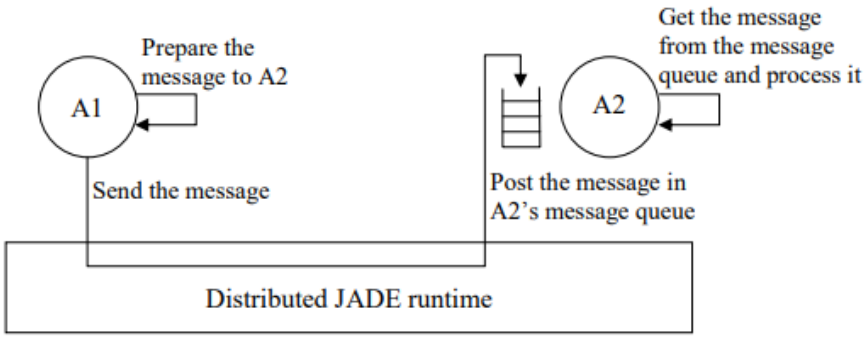
Agent je v JADE realizován jako obyčejná Java třída, u které je třeba implementovat alespoň metodu `setup`, která provede požadovanou inicializaci agenta.

Pro chování agenta JADE poskytuje třídu `Behaviour`, ve které lze definovat akce, které má agent provádět. Třídu lze upravit tak, že se definované chování odehrává cyklicky, nebo dokud nenastane určitá podmínka, a nebo že se začne odehrávat na základě nějakého podnětu (viz. vývojový diagram života agenta 2.8).



Obrázek 2.8: Cyklus života agenta v systému JADE. Převzato z [`https://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf`].

Agenti mezi sebou komunikují asynchronním zasíláním zpráv. Aby toto bylo možné, každý agent je označen unikátním identifikátorem, který se skládá ze dvou částí a to z lokálního identifikátoru agenta a identifikátoru platformy. Zprávy jsou směrovány pomocí těchto identifikátorů. Každý agent obsahuje frontu, do které se zprávy ukládají a je na agentovi, kdy si je vyzvedne (viz. obrázek 2.9).



Obrázek 2.9: Ilustrace asynchronního zasílání zpráv v JADE. Převzato z [`https://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf`].

Kapitola 3

Posilované učení

Tato kapitola se zaobírá přístupy v umělé inteligenci, které se nazývají posilované učení [16][13]. Jsou zde zmíněny přístupy, které jsou vhodné pro využití na problém učení se preferencím/komfortu uživatele v prostředí chytrého domu.

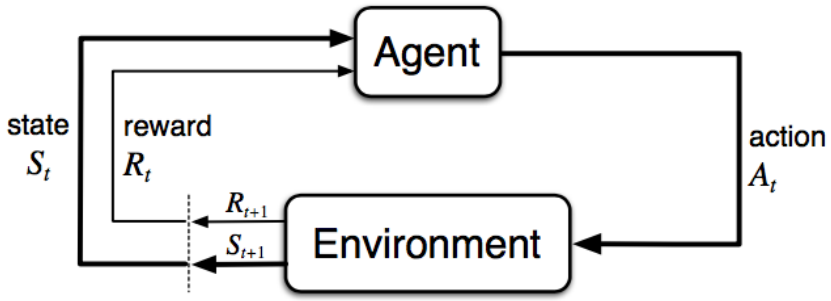
První jsou nastíněny základní koncepty posilovaného učení. Následuje popis algoritmu Q-learning. Dále je rozebráno posilované učení s neruonovými sítěmi. A na závěr je zmíněno, co je hluboké posilované učení.

3.1 Základní koncepty

Posilované učení je typ strojového učení, ve kterém se agent učí provádět akce v určitém prostředí, s cílem maximalizovat kumulativní odměnu. V posilovaném učení agent interaguje s prostředím v sérii kroků, kde v každém kroku je nucen provést akci, na základě pozorovaného stavu prostředí. Prostředí po provedení akce, přejde do dalšího stavu a agentovi poskytne odměnu, která slouží jako zpětná vazba udávající, jak dobrá provedená akce byla.

Cílem agenta je naučit se v každém stavu prostředí, vybrat nejlepší možnou akci pro provedení. Za tímto účelem agent náhodně prozkoumává zadaný prostor akcí, a za pomocí zpětné vazby se nejlepší akce učí.

Posilované učení je vhodné pro aplikace, kde není žádný existující dataset, nebo optimální řešení není předem známo.



Obrázek 3.1: Interakce agent s prostředím. Převzato z [https://lilianweng.github.io/posts/2018-02-19-rl-overview/].

3.2 Q-Learning

Q-learning je populární bez modelový algoritmus posilovaného učení, který se učí optimální funkci akce-hodnota pro dané prostředí. Q-hodnota představuje očekávanou předvídanou odměnu, kterou agent obdrží, když vykoná specifickou akci ve specifickém stavu. Q-learning algoritmus iterativně aktualizuje Q-tabulku, ve které má uložené očekávané odměny pro všechny páry stav-akce. Agent vybírá akci s nejvyšší Q-hodnotou v každém stavu, Q-hodnoty jsou aktualizovány na základě pozorovaných odměn. Algoritmus Q-learning může být popsán následovně:

1. Inicializace Q-tabulky s nulami pro všechny páry stav-akce.
2. Získat aktuální stav prostředí a vybrat akci na základě aktuální strategie. Strategie bývá jednoduchá ϵ-greedy strategie, ve které agent vybírá náhodnou akci s pravděpodobností ϵ ($\epsilon < 1$), a akci s nejvyšší Q-hodnotou s pravděpodobností $1 - \epsilon$.
3. Provedení vybrané akce a získání odměny a nového stavu.
4. Aktualizace Q-hodnoty pro daný pár stav-akce, pomocí použití Bellmanovy rovnice:

$$Q(s,a) = Q(s,a) + \alpha * (r + \gamma * max(Q(s',a')) - Q(s,a)) \tag{3.1}$$

   kde α je míra učení, γ je diskontní faktor, r je reálná odměna, s' je nový stav, a a' je akce s nejvyšší Q-hodnotou v novém stavu s'.
5. Opakování kroků dva až čtyři do konce zadané epizody.

Algoritmus Q-learning spolehlivě konverguje k optimální Q-hodnotě za určitých podmínek, které jsou konečný počet stavů a akcí, a dostatečně nízká míra učení. Avšak v realitě může být potřeba velkého množství iterací k dosáhnutí konvergence, obzvláště když je počet stavů a akcí velký. Aby se tento problém vyřešil, tak byly navrženy rozšíření Q-learningu, jako například hluboké Q-sítě, nebo Dvojitý Q-learning.

3.3 Metody Monte Carlo

Metody Monte Carlo jsou třída bez modelových algoritmů posilovaného učení, které využívají simulace pro odhad očekávané hodnoty odměny pro danou politiku.

Metody Monte Carlo se učí optimální politiku pouze pozorováním sekvencí stav-akce a odměna. Informace získává z celých dokončených epizod pomocí jedné dané politiky po celou epizodu. Po dokončení epizody si na základě získaných odměn v párech stav-akce aktualizuje politiku a opakuje běh další epizody. Tento proces se opakuje dokud učení nekonverguje.

Jedna z variant metod Monte Carlo je Monte Carlo první návštěvy (First-Visit), která aktualizuje funkci politiky jen při prvním výskytu (navštívení) daného páru stav-akce v každé epizodě. Další variantou je Monte Carlo každé navštívení (Every-Visit), tato metoda funkci politiky aktualizuje při každém výskytu daného páru stav-akce v epizodě.

Metody Monte Carlo někdy mohou velice pomalu konvergovat, protože potřebují veliké množství epizod, aby spolehlivě odhadly funkci akce-hodnota. Metody také nemusí být vhodné pro prostředí, kde jsou epizody velice dlouhé, nebo jsou odměny získávány málo.

3.4 Hluboké posilované učení

Hluboké posilované učení je skupina metod, které kombinují metody posilovaného učení a hlubokých neuronových sítí, za účelem naučení se komplexních úloh přímo ze vstupů z prostředí.

Použití hlubokého posilovaného učení je vhodné v prostředích, ve kterých se vyskytuje velice velké množství akcí nebo stavů (může se jednat i o nespočetné množství). Takové prostředí je klasickými metodami posilovaného učení nezpracovatelné.

3.4.1 Hluboké Q-sítě

Hluboké Q-sítě [12] se používají na obdobné problémy jako Q-learning, ale kde existuje veliké množství akcí a stavů, a tudíž Q-learning není možné použít, protože Q-tabulka by dosahovalo rozměrů, které by nebylo praktické, někdy ani možné mít fyzicky uložené a Q-hodnoty by konvergovali po velikém počtu vstupů, kterého v určitých aplikacích nelze dosáhnout. Proto hluboké Q-sítě používají neuronové sítě pro aproximaci Q-hodnot.

Oproti Q-learningu přinášejí hluboké Q-sítě i další změny než jen využití neuronových sítí. První z nich je přítomnost zpětné zkušenosti (anglicky *experience replay*), což je průběžné ukládání přechodů do paměti ve tvaru (s, a, r, s'), kde s je aktuální stav, a je vykonaná akce, r je získaná odměna a s' je následující stav. Při aktualizaci se náhodně vybírají tyto přechody z paměti a jsou použity k aktualizaci. Díky použití zpětné zkušenosti se snižuje korelace pozorovaných sekvencí a tedy by nemělo docházet k přetrénování neuronové sítě, a také se zvyšuje datová efektivita, protože se jedny data využijí několikrát. Druhou změnou je, že se používají dvě aproximační sítě, Q-síť a cílová síť. Q-síť má na výstupu Q-hodnoty, které se používají při aplikaci akcí v prostředí, a cílová síť produkuje cílové Q-hodnoty, pomocí kterých se vypočítá ztráta (anglicky *loss*) ztrátovou funkcí, popsanou rovnicí 3.2, kterou se následně aktualizuje Q-síť.

$$L(\theta) = ((r + \gamma max_{a'} Q(s', a'; \theta^{target})) - Q(s, a; \theta^{prediction}))^2 \tag{3.2}$$

Q-síť se aktualizuje po každé provedené akci, a cílová síť se aktualizuje periodicky po n krocích. Tyto dvě sítě se používají, aby se předcházelo nestabilitě způsobené tím, že by se jedna síť aktualizovala stále aktualizovanými daty.

Kapitola 4

Existující řešení/přístupy mutliagentních systémů pro Smart Home

Tato kapitola je zaměřena na existující řešení ovládání chytré domácnosti, případně chytrých budov, využitím multiagentního systému.

4.1 MavHome

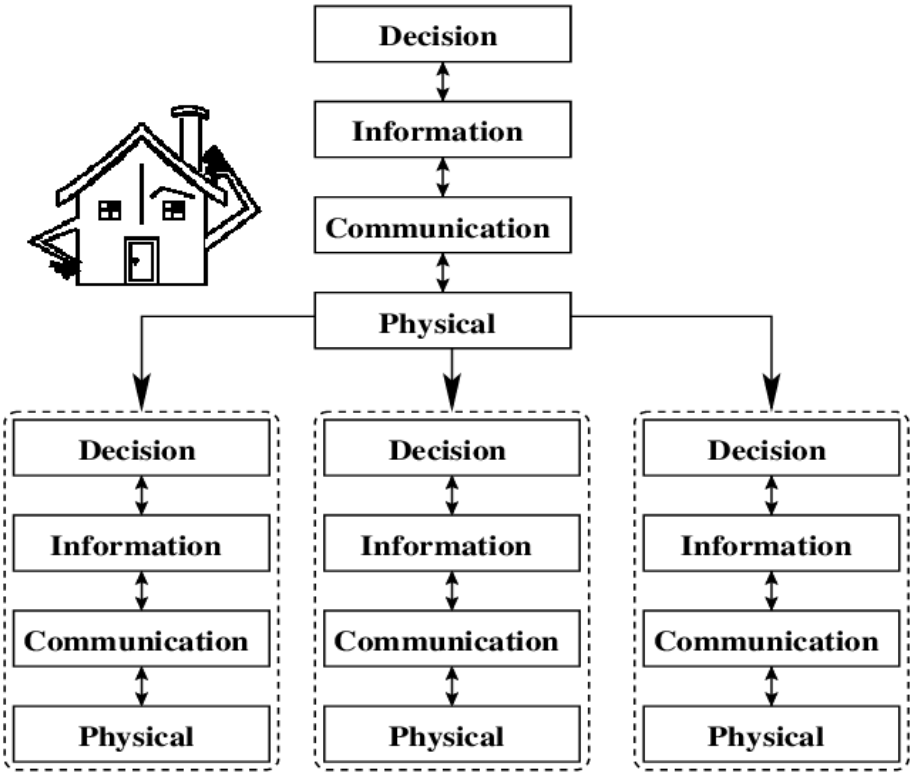
Cílem projektu chytré domácnosti MavHome je vytvoření chytrého prostředí, které bude jednat jako racionální agent, jehož úkolem je funkce, která maximalizuje komfort uživatelů a zároveň minimalizuje cenu chodu domácnosti [7].

Hlavním modulem MavHome architektury je agent. Pro jednoduché škálování je navržen tak, aby jej bylo možné skládat do hierarchické struktury, a dekomponovat dané problémy do jednotlivých částí.

4.1.1 Architektura agenta

Agent je navržen dle vertikální vrstvené architektury a skládá se ze čtyř hierarchicky strukturovaných vrstev (viz. obrázek 4.1).

- **Rozhodovací vrstva (Decision)** – Rozhodovací vrstva volí akci, která se má vykonat na základě dostupných informací, které získá za pomocí informační vrstvy.
- **Informační vrstva (Information)** - Úkolem komunikační vrstvy je získávání, ukládání a generování informací, které mohou sloužit k rozhodování.
- **Komunikační vrstva (Communication)** - Agenti si mezi sebou mohou posílat informace, požadavky a dotazy, pomocí komunikační vrstvy.
- **Fyzická vrstva (Physical)** - Fyzická vrstva obsahuje již fyzická zařízení, jako jsou senzory, aktory a síťové prvky. Protože lze agenty hierarchicky skládat, fyzická vrstva může také obsahovat jiné agenty, kteří dekomponují řešený problém ještě více.



Obrázek 4.1: Interakce agent s prostředím. Převzato z [7].

Agent vnímá prostředí nejnižší vrstvou (fyzickou) a informace je předávána směrem nahoru k rozhodovací vrstvě. Po přijetí nové informace fyzickou vrstvou, je předána komunikační vrstvě, ve které je možnost informaci zaslat dalším agentům. Následně se informace dostane do informační vrstvy, která informaci uloží do databáze, aktualizuje predikční algoritmy a informuje rozhodovací vrstvu, že se objevila nová data.

Směr vykonávání akce je shora dolů, od rozhodovací vrstvy po fyzickou vrstvu. Poté co rozhodovací vrstva vybere akci pro provedení, ji předá informační vrstvě. Po aktualizaci databáze komunikační vrstva směruje akci na vhodný efektor ve fyzické vrstvě, tím může být i další agent, ten se po přijetí akce musí rozhodnout, jaká je nejlepší reakce na danou informaci.

4.1.2 Algoritmy pro predikci akce uživatele

Projekt MavHome představuje několik algoritmů vhodných pro predikci akce uživatele z historických dat. Zde jsou popsány některé z nich.

Predikce na základě párování sekvencí (SHIP)

Algoritmus SHIP páruje aktuální sekvenci akcí s historickými sekvencemi v databázi, databáze se aktualizuje s každou interakcí uživatele se zařízeními. Při párování sekvencí se hledá historická sekvence, která odpovídá sekvenci v aktuální čase. Algoritmus se skládá ze dvou kroků. V prvním kroku je aktuálně hledaná sekvence aktualizována o právě provedenou akci, následně se vypočítá délka nejdelších sekvencí $l_t(s, a)$, která končí provedenou akcí a v aktuálním stavu s v čase t, a napáruje historickou sekvenci před časem t. Dále je také

definována frekvence $f(s,a)$ provedení akce a v aktuálním stavu. Druhý krok zhodnocuje párování s historickými sekvencemi dle rovnice

$$R(s,a) = \alpha \frac{l_t(s,a)}{\sum l_t(s,a_i)} + (1-\alpha)\frac{f(s,a)}{\sum f(s,a_i)} \tag{4.1}$$

Následující akce a k provedené je ta s největší odhadovanou hodnotou $R(s,a)$. Výpočet hodnoty lze upravit povolením nepřesností v historických sekvencích, a nebo definováním časové konstanty pro nejdéle přetrvávající sekvence (po uplynutí času se odstraní).

Predikce za použití Markovova modelu na základě úloh

Algoritmus úlohového Markovova modelu identifikuje úlohy (úloha je sekvence akcí, které tvoří jednu činnost, příkladem může být vaření, cvičení, sprchování atd.) ze sekvencí akcí pro řízení tvorby Markovova modelu pro predikci akcí. Jednoduchý Markovův model je vygenerován ze získaných sekvencí akcí a použit při predikci dalších akcí, na základě aktuálního stavu agenta. Informace o stavu obsahuje stavy jednotlivých zařízení, čas dne a datum.

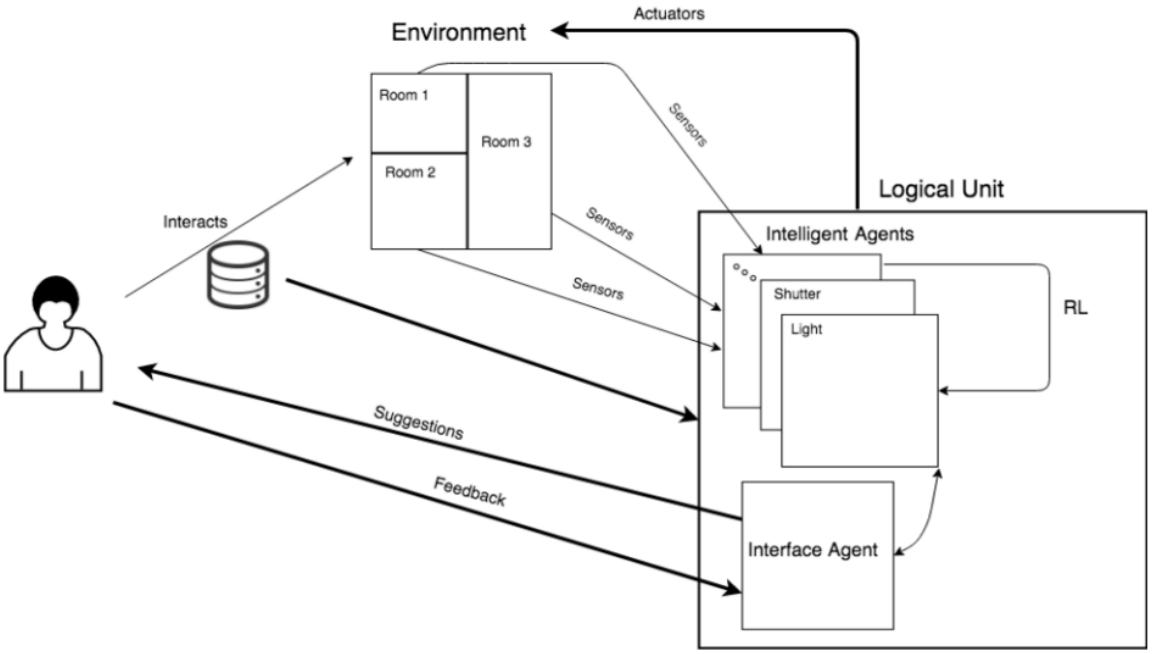
Sekvence akcí jsou první rozděleny podle jednotlivých úloh. Změna v úlohách je identifikována pomocí pauz mezi akcemi a změnou lokace, ve které jsou akce prováděny. Následovně je použit algoritmus k-means pro shlukování rozdělených sekvencí úloh do množin podobných úloh. Výstupem shlukování je skupina množin pro úlohy, kde každá množina může být označena jako specifický typ úloh. Informace o úlohách jsou základem pro přechodové pravděpodobnosti v Markovově modelu, na základě přítomnosti daných úloh ve stejné úlohové množině.

4.2 Chytrá domácnost založená na použití posilovaného učení

Tento projekt se zaměřuje na použití posilovaného učení pro chytrou domácnost [8]. Posilovaného učení využívá při naučení interakcí uživatele s prvky systému. Systém se snaží naučit správný čas dne pro provedení určité akce, kde den je interně rozdělen do patnácti minutových intervalů. Projekt klade velký důraz na přítomnost uživatele v systému a na konvergenci automatizovaných činností k preferencím uživatele.

4.2.1 Architektura

Uživatel interaguje s prostředím denně, tyto denní interakce jsou zaznamenány a uloženy do datasetu. Pro vyřešení problému, který se zaměřuje na důležitost uživatele v automatizovaném prostředí, je potřeba vytvořit pevné spojení mezi automatizačním systémem a uživatelem. Proto je systém navrhnut jako multiagentní systém, který je definován jako logická jednotka (logical unit na obrázku 4.2), která je zodpovědná za dělání rozhodnutí. Logickou jednotku tvoří dva typy agentů, racionální agent a agent rozhraní. Agent rozhraní je zodpovědný za zprostředkování komunikace mezi uživatelem a systémem. Systém je schopný navrhovat uživateli akce zvolené racionálním agentem, a uživatel může na tento návrh agenta reagovat. Díky tomuto systém konverguje k akcím, které uživatel preferuje, agent rozhraní je pro toto klíčový, protože plní roli prostředníka mezi systémem a uživatelem.



Obrázek 4.2: Architektura systému. Převzato z [8].

Racionální agenti vnímají prostředí skrze senzory a mohou jej ovlivňovat pomocí aktorů. Prostředí je složeno z několika rozdílných místností, ve kterých se nacházejí senzory i aktory. Logická jednotka obsahuje několik racionálních agentů, jednoho pro každé zařízení v prostředí, světla, rolety, teplota a další. Tito agenti vnímají jejich dané senzory v určitých místnostech a po naučení vzorů uživatele, ovládají aktory v prostředí. Například agent rolet je zodpovědný za automatizaci rolet v různých místnostech. Funkce odměny je závislá na vzorech uživatele, nacházejících se v historických datasetech, a na akci, kterou uživatel v danou chvíli provede, pokud se akce uživatele a akce vybraná agentem, podle historických dat, shodují, tak je vydána pozitivní odměna. Funkce odměny každé akce se může drasticky změnit po přijetí zpětné vazby od uživatele, což indikuje možnost přizpůsobení se preferencím uživatele.

4.2.2 Algoritmus posilovaného učení

V tomto systému se agenti stávají racionálními, díky přítomnosti posilovaného učení v logické jednotce, který jim umožňuje učit se chodu prostředí s uživatelem, za účelem optimální automatizace vůči preferencím uživatele. Cílem posilovaného učení v tomto projektu je vybrat pro každou akci čas dne (patnácti minutový interval), kdy je nejlepší ji provést.

Posilované učení probíhá následovně, první se definují a inicializují všechny globální proměnné, definuje se počet intervalů ve dni, inicializují se váhy, ztrátová funkce, počet epizod na kterých se bude agent učit, odměny pro intervaly každého agenta, a definuje se e, což je pravděpodobnost vybrání náhodné akce. Poté se definují rozdílné odměny pro agenta zodpovědného za jednotlivé zařízení. Pak už jen agent bude vybírat akce pro provedení a bude se učit, zda jeho akce je v souladu s uživatelskými preferencemi, či nikoliv (viz. algoritmus 1).

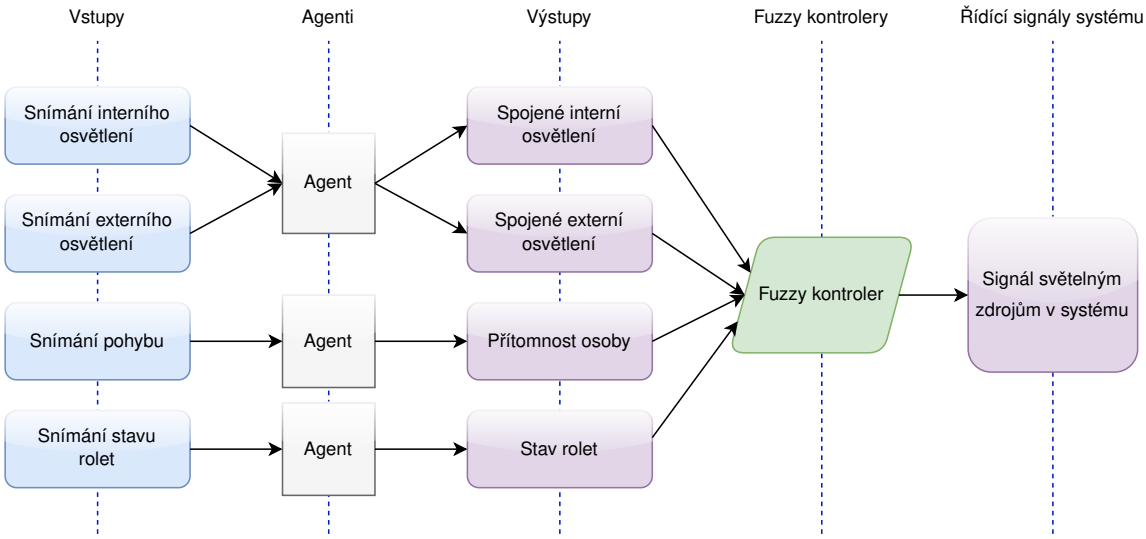
Algorithm 1 Hluboké posilované učení

```
procedure TENSERFLOW SESSION
    tf.Sessions() as sess:
    dataSet ← Historická data
    Inicializace čítače i = 0
    while Počet dní do
        Inicializace čítače j = 0
        while Počet epizod do
            if náhodné číslo < e then interval ← náhodný interval
            else
                interval ← zvolený interval
                odměna ← Funkce odměny pro daný interval
                Aktualizace sítě a odměny
            end if
        end while
    end while
end procedure
```

Procedura je stejná pro všechny racionální agenty, jedinou změnou je rozdílná funkce odměny pro agenty plnící různou činnost.

4.3 Řízení inteligentních budov za použití multiagentního systému: Fuzzy přístup

Multiagentní systém řízení inteligentních budov za pomocí fuzzy přístupu [9] se skládá ze senzorů, které jsou seskupeny podle lokace a podle cílové veličiny, kterou ovlivňují, tedy podle aktoru, kterému jsou tyto informace z těchto senzorů dodávány na vstup. Informace ze senzorů mají přímo na vstupech agenti, kteří informace zpracovávají do vhodného formátu. Zpracovaná data agenti předávají vhodnému fuzzy kontroleru, který na základě těchto vstupů a zadaných pravidel vybere akci, která povede ke splnění požadovaných podmínek v prostředí (příklad tohoto chování je naznačen na obrázku 4.3, který zobrazuje příklad diagramu chodu systému pro skupinu prvků ovlivňující osvětlení). Agenti jsou zde realizováni pomocí rekonfigurovatelného hardwaru. Jejich hlavní úlohou je samostatné paralelní zpracování informací, které jim jsou určeny v inteligentní budově, tedy urychlení zpracování těchto dat a včasné ovládání.



Obrázek 4.3: Řídící diagram fuzzy přístupu pro prvek osvětlení

Kapitola 5

Návrh multiagentního systému učícího se vzorům a preferencím uživatelů

Kapitola je zaměřena na návrh multiagentního systému a na všechny potřebné moduly k jeho chodu. Nejprve je popsán návrh struktury multiagentního systému, kde jsou představeny jednotlivé části systému. Následně je představen přístup, který bude použit pro implementaci naučení uživatelských vzorů a jak bude tyto naučené vzory využívat. Dále je popsán způsob, jakým bude multiagentní systém komunikovat s prvky v prostředí. Pokračuje sekce o návrhu metody, která bude využita k odhadu lokace uživatele v prostředí. Pokračuje představení návrhu reálného zařízení, přesněji digitální termostatické hlavice, která bude v prostředí figurovat jako aktor i senzor, a také bude agentem v prostředí. Jako poslední je představen návrh grafického uživatelského rozhraní a prvky, které v něm uživatel potřebuje pro ovládání systému.

5.1 Struktura systému

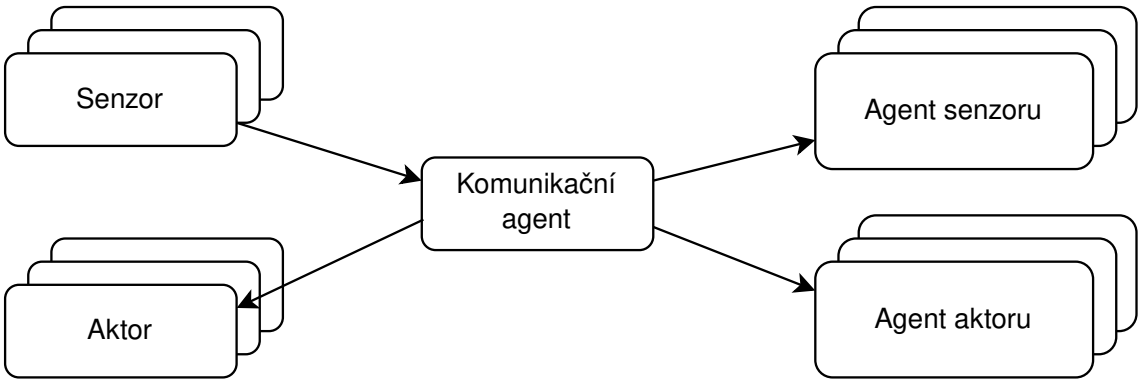
Navrhovaný systém podporuje subsystém chytré domácnosti pro ovládání teploty, tedy topení. Systém se skládá ze dvou základních částí. První část, jejímž úkolem je komunikace s prostředím, tedy s prvky, které nejsou přímo zasazeny v jádru multiagentního systému, tyto prvky jsou senzory a aktory, jež jsou fyzicky zasazeny v prostředí. Druhou částí je již skupina agentů, jejichž úlohou je zajistit řízení těchto aktorů v prostředí, podle uživatelských vstupů, nebo podle naučených preferencí.

5.1.1 Interakce s prvky v prostředí

V systému musí existovat část, která bude plnit požadavky uživatele, nebo požadavky které bude vydávat agent obstarávající automatizaci uživatelských preferencí. Tato část se skládá z prostředí, které je možné vnímat pomocí v něm zasazených senzorů, a je možné jej ovlivňovat pomocí přítomných aktorů.

V této části systému se nacházejí tři rozdílní agenti, agent senzoru, agent aktoru a agent uživatele. Úkolem agenta senzoru je periodicky, nebo při změně stavu získávat data z daného senzoru a zasílat je agentům, kteří o tyto data mají zájem. Agent aktoru má za cíl řídit nebo ovládat daný aktor tak, aby vyhověl požadavkům uživatele, nebo v případě

automatizované činnosti naučeným uživatelským preferencím. Jak požadavky, tak naučené preference agentu aktora dodává agent uživatele (viz. obrázek 5.1). Aktory i senzory mohou být také realizovány jako agenti, pokud disponují potřebnými prvky a funkcionalitou, ale také to mohou být v podstatě jen pasivní prvky vykonávající činnost, bez ohledu na stav prostředí.



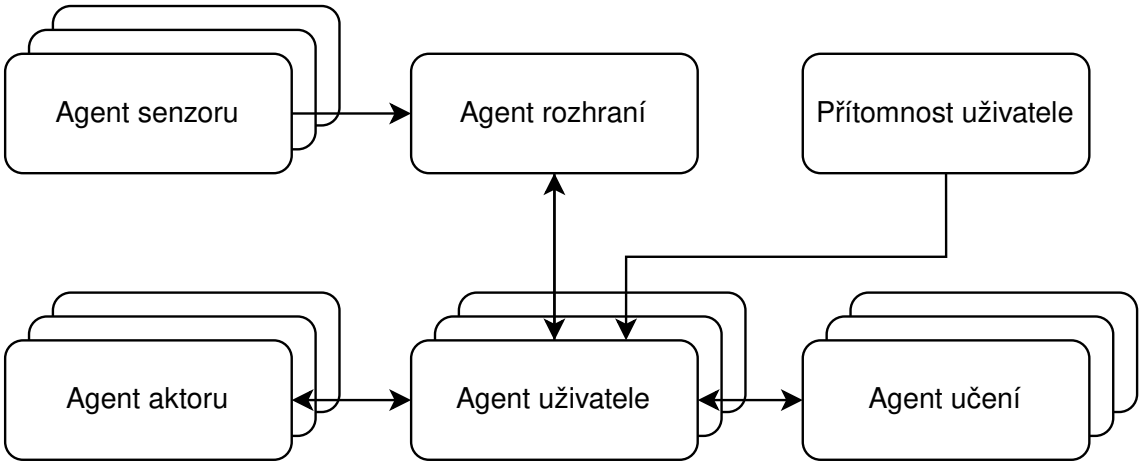
Obrázek 5.1: Návrh podsystému komunikace požadavků k zařízením

5.1.2 Interakce systému s uživatelem a ovládání systému

Systém obsahuje část, která zastřešuje zpracování uživatelských vstupů, komunikaci s uživatelem, a učení se z uživatelských vstupů a následné použití naučených vzorů k automatickému ovládání.

V této části se nachází pět typů agentů, z nichž dva byli zmíněni v předcházející části 5.1.1, jedná se o agenta aktoru a agenta senzoru, další tři jsou agent uživatele, agent rozhraní a agent učení. Agent rozhraní je spojen s uživatelským rozhraním. Když uživatel zadá nějaký vstup, tak o tom agent rozhraní informuje daného agenta uživatele, kterému je vstup určen. Úkolem agenta rozhraní je také informovat uživatele o stavu prostředí, proto získává informace od agentů senzoru, které zobrazuje v uživatelském rozhraní. Agent uživatele má za úkol reagovat na uživatelské vstupy z agenta rozhraní vhodnou akcí. Takovou akcí může být zaslání zprávy vhodnému agentu aktora, aby provedl akci, která by přivedla prostředí do stavu, který vyhovuje podmínkám z uživatelského vstupu, nebo zažádáním o vhodnou akci agenta učení. Agent učení je agent obsahující algoritmus strojového učení, který slouží k naučení uživatelských preferencí, které patří k určité části prostředí, pro určitého uživatele, tedy agent učení se učí preferencím jednoho uživatele pro jeden určitý aktor (viz. obrázek 5.2).

Systém je schopen pracovat s více uživateli v prostředí a také rozlišovat mezi jejich preferencemi, proto je potřeba, aby byl schopen rozlišovat, jaké podmínky by mělo prostředí splňovat, aby bylo dosaženo preferovaného stavu pro daného uživatele. Proto se v systému nachází lokátor uživatelů, který vnímá vzdálenost uživatelů a na základě tohoto dostane agent uživatele zprávu, že jemu daný uživatel se nachází v určité blízkosti specifické lokality a tedy je vhodné přizpůsobit stav prostředí preferencím daného uživatele.



Obrázek 5.2: Návrh podsystému interakce s uživatelem

5.1.3 Agenti

Agenti jsou realizování jako víceméně reaktivní agenti, kteří však mají určitou bázi znalostí. Tedy jejich rozhodování probíhá hlavně na základě zpráv od ostatních agentů nebo na základě podnětů z prostředí, ale také může být do určité míry ovlivněno znalostmi.

5.2 Učení se uživatelským preferencím

Pro učení uživatelských preferencí jsem zvolil algoritmus posilovaného učení Q-Learning, který je popsán v sekci 3.2. Stavový prostor bude rozdělen na sedm dní, které budou dále rozděleny na patnácti minutové intervaly, tedy systém se bude učit preference pro jednotlivé patnácti minutové intervaly ve dnech v týdnu. Prostor akcí bude rozdělen na 21 akcí, z nichž každá značí jednu teplotní hodnotu s inkrementem půl stupně Celsia, teplotní rozmezí je od 15 stupňů až po 25 stupňů Celsia.

Q-Learning bude aktualizován dle rovnice 5.1 v situaci, kdy bude aktivní uživatelský vstup (systém je řízen manuálně uživatelem). V takovém případě bude aktualizována taková akce, která odpovídá zadané hodnotě uživatele, v aktuálním časovém intervalu kladnou odměnou.

$$Q(s,a) = Q(s,a) + 0.1 * (r + 0 * max(Q(s',a')) - Q(s,a)) \tag{5.1}$$

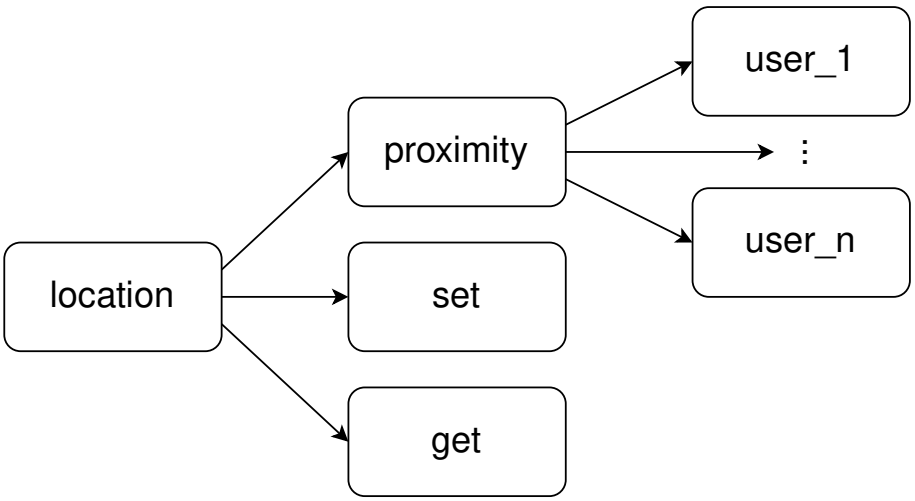
Další situace, kdy proběhne aktualizace, nastane, když je systém řízen automaticky naučenými preferencemi a uživatel manuálně zadá změnu nastavené hodnoty, která je chápána jako neshoda s naučenými preferencemi, pokud byla hodnota zvolená Q-Learningem rozdílná od zadané uživatelem, tak je tato hodnota (akce) aktualizována se zápornou odměnou, pokud se hodnota liší pouze o půl stupně, je aktualizována s malou zápornou odměnou (hodnota odměny je 1, nebo 0.1 v případě malé odměny).

5.3 Komunikace s prostředím

Pro komunikaci systému s prvky prostředí, senzory a aktory, bude použit komunikační protokol MQTT. Komunikace bude obsahovat několik témat pro rozlišení, jakou informaci daná zpráva obsahuje (viz. obrázek 5.3). První téma je /location/proximity/user_n, které

slouží k předávání informace o přítomnosti uživatele *user_n* v dané lokaci (lokací je zamýšlena místnost) *location*, zpráva tohoto tématu obsahuje informaci buď, že uživatel se nově v lokaci nachází a nebo, že se nově v lokaci již nenachází, informuje tedy o změně polohy uživatele.

Další dvě témata jsou /location/set a /location/get, první téma je použito při nastavování hodnoty, které má aktor v dané lokaci *location* dosáhnout, a druhé téma je použito při sdílení hodnoty senzoru v lokaci *location* o aktuálním stavu v prostředí.



Obrázek 5.3: Struktura témat pro mqtt komunikaci

5.3.1 MQTT

MQTT je komunikační protokol, který pro výměnu zpráv mezi zařízeními používá princip publikování a odebírání [14]. MQTT je navržen tak, aby byl otevřený, jednoduše implementovatelný a umožňoval připojení mnoha lehkých klientů k serveru. Díky těchto vlastnosti je MQTT vhodné pro použití v omezených sítích s malou šířkou pásma nebo pro zařízení s malou kapacitou paměti a omezenými výpočetními zdroji. Návrh MQTT minimalizuje požadavky na šířku pásma a zároveň může zajišťovat určitou úroveň spolehlivosti doručení.

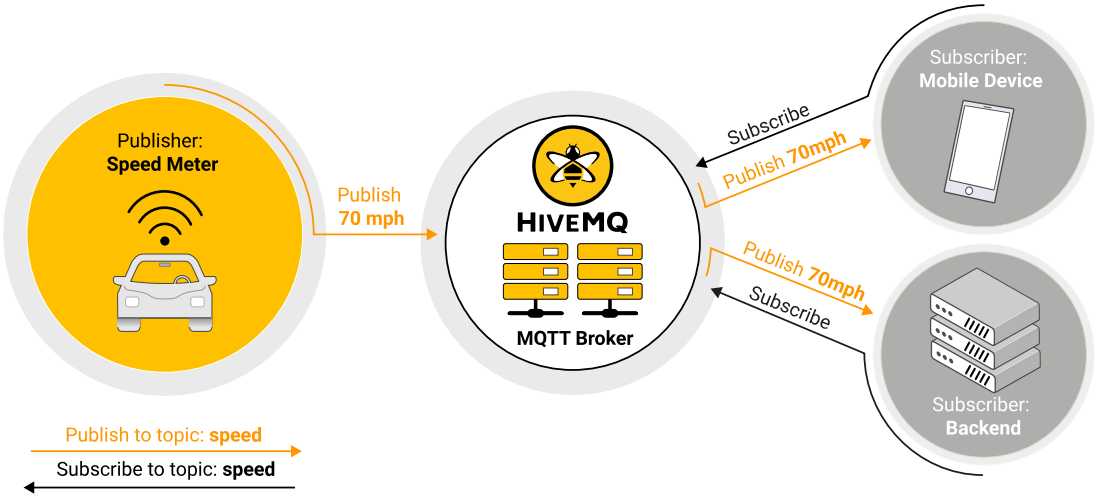
MQTT definuje dva typy rolí v síti, brokera který spravuje provoz zpráv a klienty. Broker je server, který přijímá zprávy od připojených klientů a následně je předává náležitým klientům. Klientem jsou veškerá zařízení, která jsou schopna komunikovat s brokerem. Klient může být senzor či aktor v prostředí. Komunikace probíhá následovně:

1. Klient se připojí k danému brokeru, následně má možnost si registrovat odběr jakýchkoliv témat. Připojení může být TCP/IP připojení, ale také šifrované TLS připojení. Klient může při připojování brokeru definovat zprávu, kterou by měl broker publikovat na zadané téma při odpojení klienta.

2. Klient publikuje zprávu pod zadaným tématem, zasláním zprávy pod daným tématem brokeru.

3. Broker jemu zaslané zprávy směruje na klienty, kteří mají registrovaný odběr témat, pod kterými jsou dané zprávy zaslány.

Protokol nedefinuje formát obsahu zpráv a tedy je možné využít libovolného existujícího formátu nebo i vlastního. Ukázka komunikace pomocí MQTT protokolu je dostupná na obrázku 5.4.



Obrázek 5.4: Příklad MQTT komunikace. Převzato z [`https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/`].

MQTT komunikace je strukturována do témat, což je textový řetězec, pomocí kterého broker směruje zprávy. Téma se skládá z několika úrovní, kde úrovně jsou odděleny znakem `/`. MQTT témata mohou při odběru být definovaná pomocí takzvaných divokých karet (jsou schopny nahradit jednu či více úrovni v tématu), což jsou znaky, pomocí kterých lze odebírat nespecifické téma, těmito znaky jsou `+` a `#`. Znak `+` lze v tématu použít jako divokou kartu jedné úrovně. Znak `#` lze použít jako divokou kartu více úrovní.

MQTT podporuje tři úrovně kvality služeb. Kvalita služeb definuje způsob, jakým bude broker dodávat zprávy klientům. Úroveň kvality služeb určuje klient při registraci odběru jednotlivých témat a při publikování zprávy pod daným tématem.

- **Úroveň 0** - broker i klient zprávu pouze odešlou.
- **Úroveň 1** - po odeslání zprávy odesilatel čeká na potvrzení o přijetí zprávy příjemcem, pokud potvrzení nepřijde v daném časovém intervalu, odesilatel odesílá zprávu znovu. Broker přijatou zprávu přeposílá odběratelům s kvalitou služeb stejnou nebo nižší v případě, že cílový klient kvalitu služeb úrovně 1 nepodporuje. Kvalita služeb úrovně 1 zajišťuje, že zpráva bude cílovému zařízení doručeno alespoň jednou.
- **Úroveň 2** - odesilatel po odeslání zprávy čeká na potvrzení od příjemce, pokud potvrzení nepřichází po nějakou dobu, tak odesilatel odesílá zprávu znovu s příznakem duplicity. Po přijetí potvrzení od příjemce odesilatel odesílá zprávu, na jejímž základě si příjemce smaže veškeré informace ohledně dané komunikace, po zpracování této zprávy příjemce zasílá odesilateli další potvrzení, přijetím tohoto potvrzení odesilatelem komunikace končí. Tato úroveň kvality služeb zajišťuje, že zpráva bude doručena právě jednou.

V případě, že je publikována zpráva klientem na nějaké téma s vyšší kvalitou služeb, než má zadanou odběratel, tak broker bude zprávu publikovat tomuto klientovi s nižší kvalitou služeb, se kterou je přihlášen k odběru.

5.4 Snímání přítomnosti uživatele

Snímání přítomnosti uživatele slouží k určení přibližné polohy uživatele, podle které se bude určovat, zda by se měl systém snažit vyhovět preferencím daného uživatele ve snímané lokaci.

Pro zjišťování přítomnosti je použita bezdrátová komunikační technologie Bluetooth Low Energy (dále jen Bluetooth LE), kde každý uživatel přítomen v prostředí vždy má zařízení podporující Bluetooth LE. V každé lokaci bude umístěno zařízení, které bude periodicky skenovat přítomná Bluetooth LE zařízení, a na základě předdefinovaných názvů zařízení a síly signálu jednotlivých zařízení bude určovat, který uživatel se v dané lokaci nachází.

K určování přítomnosti bude použit parametr některých bezdrátových technologií (včetně Bluetooth LE), a to parametr RSSI (*Received Signal Strength Indication*), který je indikátorem síly signálu ze snímaného zařízení. Z parametru RSSI je možné vypočítat přibližnou vzdálenost pomocí rovnice 5.2 [11].

$$d = 10^{(M-RSSI)/(10*N)} \tag{5.2}$$

V rovnici 5.2 d značí výslednou vzdálenost v metrech, M je konstanta naměřené hodnoty RSSI zařízení ve vzdálenosti jednoho metru, $RSSI$ je naměřená hodnota RSSI, a N je faktor prostředí, který se pohybuje v rozmezí 2 až 4 a udává, v jakém prostředí se vzdálenost měří, zda se v prostředí nacházejí překážky, nebo zda je v prostředí rušení jinými signály a další. Faktor prostředí je hodnota, která se nejčastěji získává zkoušením různých hodnot a výběrem takové, která se nejvíce blíží realitě.

5.4.1 Bluetooth Low Energy

Bluetooth Low Energy je část standartu Bluetooth [15]. Ačkoliv Bluetooth LE a obyčejný Bluetooth se do jisté míry překrývají, Bluetooth LE pracuje na trochu rozdílných principech. Bluetooth LE je bezdrátový komunikační protokol, který je hojně používán zařízeními pro chytrou domácnost. Pokud v dnešní době navrhujeme moderní mobilní platformou, která má komunikovat z okolním světem, je Bluetooth Low Energy jeden z nejjednodušších způsobů pro tuto aplikaci. Velkou výhodou této technologie je nízká spotřeba energie a cena, nevýhodou je poměrně krátký dosah.

5.5 Termostatická hlavice jako agent

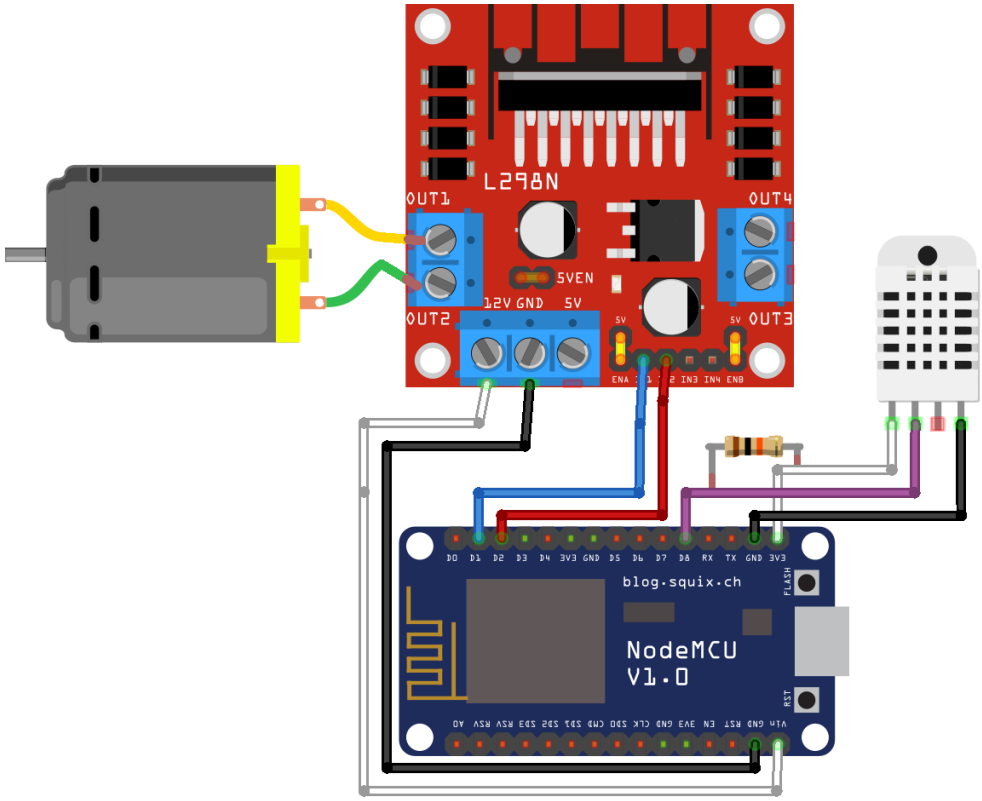
Návrh termostatické hlavice [4] se skládá ze čtyř základních komponent, ze stejnosměrného motoru, který je pomocí přípojky k radiátoru schopen ovládat ventil, z h-můstku L298N[1], jež slouží ke spolehlivému ovládání pohybu stejnosměrného motoru, z digitálního teploměru a vlhkoměru DHT22[2], a mikrokontroleru NodeMCU s čipem ESP8266, který obsahuje modul připojení k WiFi síti, který bude použit ke komunikaci se zbytkem systému.

[1] Příklad s L298N: `https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/`
[2] Příklad s DHT22: `https://www.instructables.com/How-to-use-DHT-22-sensor-Arduino-Tutorial/`

Mikrokontroler je využit k ovládání ventilu radiátoru, pomocí stejnosměrného motoru, a také ke snímání teploty z prostředí pomocí senzoru DHT22, tedy vytvořená hlavice bude figurovat v systému jako aktor i senzor.

5.5.1 Zapojení

Pro chod hlavice, musí být mikrokontroler připojen k elektrickému zdroji s napětím o hodnotě 5 voltů. Mikrokontroler poté působí jako zdroj pro zbylé části hlavice. Na senzor DHT22 je přivedeno napětí z NodeMCU pomocí pinu 3V3, který poskytuje napětí 3.3 voltů. Senzor je nutné připojit k zemi, a také je potřeba připojit datový pin k NodeMCU, kterým se budou získávat naměřená data. Pro správné chování datového vývodu je potřeba propojit datový vývod a zdrojový vývod pull-up rezistorem o hodnotě 4.7kΩ. Na modul L298N musí přicházet napětí 5 voltů, které bude přivedeno z pinu Vin z NodeMCU, který je přímo propojen se zdrojem mikrokontroleru, modul musí být také připojen k zemi. Pro ovládání motoru je třeba zapojit digitální piny mikrokontroleru na vstupní piny modulu, a propojit motor s výstupy modulu (viz. obrázek 5.5).



Obrázek 5.5: Schéma zapojení návrhu digitální termostatické hlavice

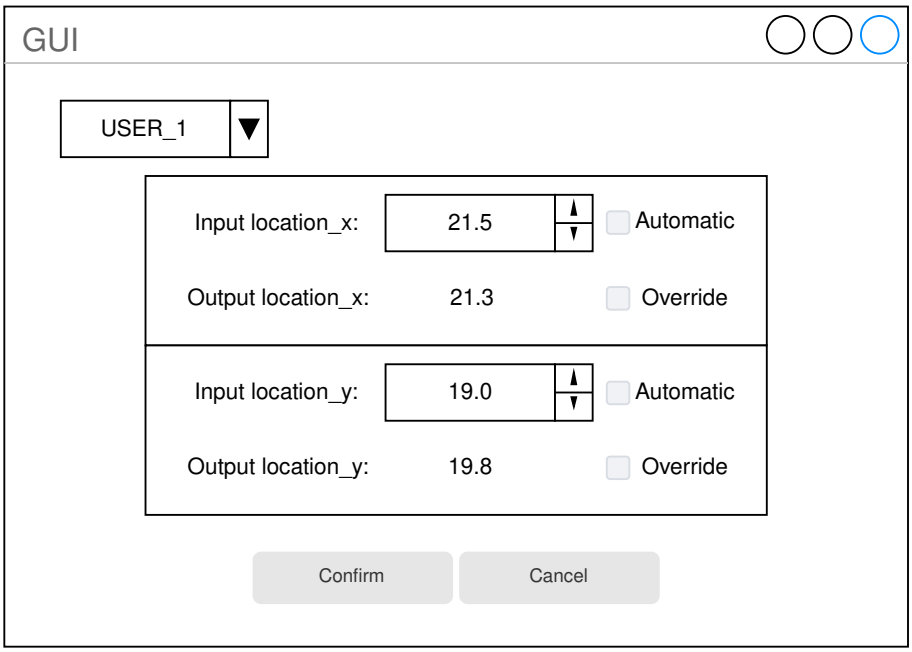
5.5.2 Funkcionalita

Termostatická hlavice bude identifikována lokací, ve které se nachází, aby byla komunikace se systém unikátně identifikovatelná. Hlavice bude systému periodicky zasílat naměřené hodnoty z prostředí, a pomocí těchto hodnot se bude také ovládat její řízení, aby bylo

dosaženo zadaných podmínek. Hlavice bude ze systému dostávat zprávy o aktuálních podmínkách, které mají být splněny, tedy bude dostávat aktuální požadovanou teplotu. Na základě této hodnoty, bude hlavice pohybovat motorkem, aby se ventil radiátoru otevřel či zavřel a bylo dosaženo požadované teploty.

5.6 Grafické uživatelské rozhraní

Grafické rozhraní (viz. mockup na obrázku 5.6) slouží uživatelům pro zadávání požadavků na preferované podmínky v prostředí a ke sledování aktuálního stavu v prostředí. Uživatel může pomocí něj zadávat hodnoty a tím tedy ovládat jednotlivé aktory, pro jednotlivé uživatele v systému. Grafické rozhraní také nabízí možnost přepnout na automatické ovládání, které nastavuje podmínky pro jednotlivé aktory na základě naučených preferencí. Lze také použít přepisovací (*override*) mód, který slouží k rychlejšímu přeučení preferencí na zadanou hodnotu.



Obrázek 5.6: Mockup grafické rozhraní

Kapitola 6

Implementace multiagentního systému učícího se vzorům a preferencím uživatelů

Tato kapitola se zabývá implementací jednotlivých modulů systému. Zprvu je představen multiagentní systém. Poté následuje popis implementace snímání přítomnosti uživatelů v prostředí. Poslední sekce je zaměřena na termostatickou hlavici v roli agenta.

6.1 Multiagentní systém

Multiagentní systém je implementován v jazyce Java, za pomocí frameworku JADE, který byl popsán v 2.4. Systém se skládá z několika typů agentů, kteří jsou vytvoření rozšířením JADE třídy `Agent`.

6.1.1 Komunikace s prostředím

Pro úlohu komunikaci s prostředím jsou důležité tři typy agentů, agent komunikace, agent senzoru a agent aktoru.

Agent senzoru

Implementace agenta senzoru je obsažena ve třídě `SensorAgent`.

Agent hned po svém vytvoření zaregistruje své služby agentovi *Directory Faciliator*, a jako typ služby zadá `sensor`. Poté je agentovi přidáno cyklické chování, které implementuje obsluhu zpráv s řečovým aktem `INFORM` a `SUBSCRIBE`. Zprávy s řečovým aktem `INFORM` přicházejí od agenta komunikace a jedná se o informace o stavu prostředí v dané lokalitě. Tuto informaci agent po přijetí rozešle všem agentů, kteří u něj mají zaregistrovaný odběr. Zprávy `SUBSRIBE` slouží k registraci odběru odesilatele k danému agentu senzoru.

Agent aktoru

Agent aktoru je implementován ve třídě `ActorAgent`.

Ihned po vytvoření si agent přidá cyklické chování, kterým obsluhuje příchozí zprávy. Agent přijímá zprávy s řečovým aktem `REQUEST` a `CANCEL`. Zpráva s řečovým aktem `REQUEST` značí požadavek na řízení daného aktoru, agent požadavku vyhoví, pokud již neobsluhuje

jiný požadavek od jiného zadavatele nebo aktuální zadavatel požadavku je stejný, jako aktuálně obsluhovaný agent. Informace o obsluze je uložená v proměnné `beingServed` ve formě zadavatelova agentního identifikátoru v případě, že již někoho obsluhuje, nebo hodnotou null v opačném případě. Přijetí požadavku probíhá odesláním požadavku agentu komunikace na publikaci požadovaného řízení na MQTT server, a následně požadavek potvrdí zadávajícímu agentovi zprávou s řečovým aktem `AGREE`. V případě příchozího požadavku, kdy je již někdo agentem obsluhován, tak agent odpoví zprávou `REFUSE` s informací, koho agent právě obsluhuje.

Zpráva typu `CANCEL` značí, že obsluhovaný agent chce zrušit řízení daného aktoru. Po přijetí takové zprávy si agent změní stav své obsluhy a odešle agentu komunikace požadavek na řízení s hodnotou `off`, která značí, že na daný aktor v dané chvíli nikdo nemá požadavky na řízení.

Agent komunikace

Implementace agenta komunikace je obsažena ve třídě `CommunicationAgent`.

Po vytvoření, se agent pokusí připojit k zadanému MQTT serveru a nastavit si callback funkci pro událost, kdy přijde publikace na některé z odebíraných témat. Callback obsahuje rozlišení témat zpráv na téma `<location>/get`, což je téma pod kterým publikují senzory v prostředí aktuální stav, a téma `<location>/proximity/<user>` oznamující změnu polohy uživatele ze senzoru přítomnosti. Pokud přijde zpráva o aktuálním stavu ze senzoru, tak je tato zpráva přeposlána agentu senzoru v dané lokaci. Je-li příchozí zpráva o přítomnosti uživatele, tak je její informace přeposlána agentu uživatele.

Po úspěšném připojení k MQTT serveru, se agent dotáže agenta *Directory Facilitator* na existující agenty, kteří publikovali své služby pod typem `sensor`. Po obsloužení tohoto požadavku má agent k dispozici názvy agentů senzoru, jejichž součástí je i název lokací. Následně si agent registruje odběry témat pod získanými lokacemi.

Poslední činností agenta je přidáni si cyklického chování, které obsluhuje zprávy řečového aktu `REQUEST`, které přicházejí od agentů aktoru a jejichž účelem je nastavení aktorů v prostředí. Tyto zprávy jsou publikovány na MQTT server pod tématem `<location>/set`.

6.1.2 Učení preferencí

Učení preferencí je realizováno pomocí agenta učení, který je implementován ve třídě `RLAgent`.

Každý agent učení si při inicializaci vytvoří instanci objektu `QLearningTemperature`, který implementuje algoritmus posilovaného učení Q-Learning, ve formě v jaké byl popsán v návrhu 5.2, jedinou změnou je, že v učení nejsou rozlišeny stavy podle dnů. Dny bylo zamýšleno rozlišovat, protože jednotlivé dny mohou uživatelé trávit jinak, a tedy by trávili stejné časy v různých lokacích, nebo by nemuseli být přítomni vůbec, ale jelikož toto ošetřuje snímání přítomnosti, není třeba dny rozlišovat. Touto změnou se také zrychlí konvergence k preferovaným podmínkám, protože Q-Learning bude dostávat aktualizace rychleji, díky zmenšení velikosti Q-tabulky na 96x21 hodnot (96 intervalů, 21 teplotních hodnot).

Následně agent obsluhuje požadavky řečového aktu `QUERY_REF`, kterými se agent uživatele dotazuje na hodnotu z algoritmu Q-Learning pro zadaný interval dne. Dále také přijímá zprávy `INFORM`, kterými jej agent uživatele informuje, jak aktualizovat kterou akci, v kterém intervalu a s jakou odměnou.

6.1.3 Ovládání systému

Pro ovládání systému jsou použity dva typy agentů, agent uživatele a agent rozhraní.

Agent uživatele

Agent uživatele je implementován ve třídě `UserAgent`.

Po svém vytvoření agent registruje své služby typu `user` pod agentem *Directory Facilitator* a následně získá všechny agenty nabízející službu `sensor`. Poté vytvoří n agentů učení, kde každý agent koresponduje k jednotlivé lokalitě.

Následně si přiřadí cyklické chování vykonávající obsluhu zpráv a ovládání podmínek dle vstupů. První úkon ve smyčce je kontrola, zda se nezměnil časový interval (intervaly jsou rozděleny po patnácti minutách), pokud se změnil a některé vstupy pro aktory jsou nastaveny v režimu automatickém, tak se agent dotáže agenta učení, aby ho informoval jaké podmínky je vhodné nastavit. Pokud jsou aktory v režimu přepisu (*override*), tak agent požádá agenta učení o nové podmínky nastavení, ale také jej informuje, že má aktualizovat zadanou hodnotu uživatelem s pozitivní odměnou. Pokud aktor není v režimu automatickém, ani přepisu, tak pouze informuje agenta o aktualizaci hodnoty s pozitivní odměnou.

Následuje obsluha příchozích zpráv. Pokud přijde zpráva s řečovým aktem `REQUEST`, tak se jedná o požadavek na změnu nastavení podmínek pro aktory, buď to z rozhraní, nebo od jiného agenta uživatele, to lze rozeznat z obsahu zprávy, která obsahuje atribut `_type`. Jedná-li se o požadavek z rozhraní, tak agent požadavek potvrdí, přijaté nastavení si uloží, kde toto nastavení je ve formátu slovníku, kde klíčem je název aktoru a hodnoty jsou zadaná hodnota pro aktor a případně dodatečné znalost, které informuje o nastaveném režimu, automatický nebo přepis. Agent poté na základě těchto informací rozešle požadavky jednotlivým agentům aktoru. Pokud se však jedná o požadavek od jiného agenta uživatele, jde o požadavek od odmítnutého agenta agentem aktoru, takový požadavek je potvrzen a žádající agent je přidán do skupiny agentů (pouze pokud je dotazovaný agent vedoucím dané skupiny, a má tedy právo měnit nastavení daného agenta aktoru), kteří pro maximalizování společného komfortu všech uživatelů ovládají aktor společnou hodnotou, která je průměr všech nastavených podmínek (teplot).

V případě zprávy s řečovým aktem `CANCEL` se jedná o žádost zrušení nastavení. Pokud je odesilatelem rozhraní, tak požadavek značí zrušení nastavení pro daného agenta. Pokud se agent nenachází pro daný aktor ve skupině, tak odešle žádost o zrušení nastavení danému agentu aktoru. Pokud se agent ve skupině nachází, tak v případě že je jejím vedoucím, požádá agenta aktoru o zrušení a odešle informování o zrušení skupiny všem agentům uživatele, kteří byli členy skupiny. Když agent není vedoucím skupiny, tak pouze zašle žádost o zrušení vedoucímu agentu uživatele. Jestliže není odesilatelem zprávy rozhraní, tak to je agent, který chce zrušit nastavení v rámci skupiny, jejímž vůdcem je dotazovaný agent. Vůdčí agent odebere odesilatele ze skupiny a požádá agenta aktoru o změnu nastavení.

Zprávy řečového aktu `REFUSE` jsou zprávy o odmítnutí požadavku na nastavení od agenta aktoru, protože již jiného agenta obsluhuje. Odmítnutí nese informaci, jaký agent je obsluhován a na tohoto agenta je odeslán požadavek na připojení do skupiny a společné nastavování daného aktoru.

Dále přijímá zprávy řečového aktu `INFORM`, které přicházejí od jiných agentů uživatele nebo od agenta komunikace. Agent komunikace informuje o změně polohy uživatele, který je přiřazen k danému agentu. Pokud se uživatel nově nachází v nějaké lokalitě, tak na všechny nastavené aktory v dané lokalitě odešle požadavek o nastavení. V opačném případě v dané

lokalitě zruší nastavení všech aktorů. Jiný agent uživatele informuje agenta, že jako vůdce skupiny ji ruší. Agent po zrušení skupiny zašle na daného agenta aktora vlastní požadavek na řízení.

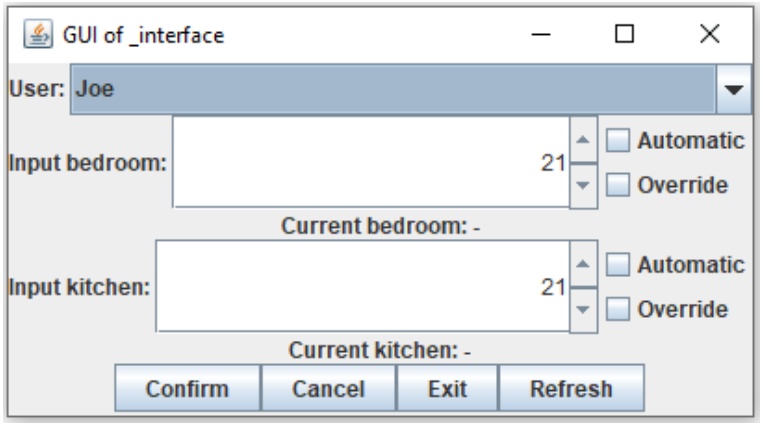
Agent dále obsluhuje zprávy s řečovým aktem `INFORM_REF`. Tyto zprávy odesílá agent učení na požadavek, jaká naučená hodnota má být v daném intervalu použita. Když je agent uživatele v režimu přepisovacím, tak informuje agenta učení, aby aktualizoval hodnotu s negativní, nebo malou negativní odměnou, pokud hodnota nesedí se zadanou hodnotou uživatelem. Pokud je aktivní režim automatický, tak pouze dodanou hodnotu nastaví.

Zprávy s řečovým aktem `AGREE` potvrzují nastavení hodnoty pro daný aktor, ať už samostatně, nebo skupinově. Pokud je odesilatel agent aktoru, tak potvrzuje přijetí požadavku pro řízení, a agent se nastaví jako vůdce pro daný aktor. V případě agenta uživatele jako odesilatele se jedná o potvrzení přijetí agenta do skupiny, agent si nastaví odesilatele, jako vůdce skupiny pro daný aktor. Je-li agent v režimu automatickém, tak ještě při potvrzení informuje rozhraní jaká naučená hodnota byla potvrzena pro nastavení aktoru.

Agent rozhraní

Agent rozhraní je obsažen ve třídě `InterfaceAgent`, která rozšiřuje JADE třídu `GuiAgent`.

Agent jako první získá od agenta *Directory Facilitator* záznamy o agentech, kteří poskytují službu `sensor` a `user`. Poté si vytvoří instanci objektu `InterfaceAgentGui`, jehož třída implementuje grafické uživatelské rozhraní, a předá mu získané názvy agentů uživatele a agentů senzoru. Tato třída v konstruktoru vytvoří jednotlivé prvky grafického rozhraní, aby bylo možné ovládat jednotlivé aktory a zobrazovat stavy v prostředí pro jednotlivé uživatele. Grafické rozhraní obsahuje čtyři tlačítka, která vytváří určitou událost, kterou případně pošle agentu rozhraní, *Confirm* potvrzuje nastavené hodnoty pro daného uživatele a vytváří událost o potvrzení, *Cancel* ruší všechny nastavené hodnoty daného uživatele, *Exit* zastaví agenta rozhraní, a *Refresh* nastaví prvky rozhraní na poslední potvrzené nastavené hodnoty pro daného uživatele (viz obrázek 6.1).



Obrázek 6.1: Grafické uživatelské rozhraní

Poté si agent přiřadí chování, ve kterém obsluhuje zprávy řečového aktu `INFORM`. Tyto zprávy jej buď informují o aktuálním stavu prostředí, který mu dodá agent senzoru, a nebo o aktuálně nastavených hodnotách pro aktory v případě, že je daný aktor nastavený v režimu automatickém. Agent tyto informace předá grafickému rozhraní, které je vhodně zobrazí. Agent také implementuje metodu, která obsluhuje události, které mu předává grafické roz-

hraní. Pokud se jedná o potvrzení (*Confirm*) nebo o zrušení (*Cancel*), tak tuto informaci agent přepošle danému agentu uživatele, pokud se jedná o zastavení agenta (*Exit*), tak agent uvolní zdroje spojené s grafickým rozhraním a smaže se.

6.2 Snímání přítomnosti uživatelem

Snímání přítomnosti uživatelů je implementováno v jazyce Python a je k němu použit jednodeskový počítač Raspberry Pi, přesněji Model B1+ V1.2[1], ale pro tento účel je možné použít jakýkoliv model, který disponuje podporou komunikace Bluetooth a nějakou možností připojení k internetu, ať už přes rozhraní ethernet, nebo bezdrátově pomocí WiFi sítě.

Lokace uživatelů je implementována pomocí vytvořené třídy `BluetoothLocator`. Tato třída přijímá dva argumenty, název lokace a hraniční vzdálenost, které značí hranici, od které se bere, že se zařízení nachází v dané lokaci. Při instanciaci třídy se načtou jednotlivý uživatelé ze souboru `users.txt`, ve kterém se nacházejí na jednotlivých řádcích název uživatele a název jeho Bluetooth zařízení. Tyto hodnoty jsou v souboru odděleny čárkou. Po načtení uživatelů proběhne připojení k MQTT serveru. Následuje pouze volání funkce `scanLoop`, ve které běží nekonečná smyčka, která na začátku skenuje okolí pro nalezení Bluetooth LE zařízení. Skenování probíhá za využití knihovny *bluepy*[2]. Po dokončení skenování se výsledná zařízení proiterují a zkontroluje se vzdálenost těch zařízení, jejichž jména se shodují se jmény zařízení, které byly při inicializaci načteny ze souboru `users.txt`. Pokud u těchto zařízení došlo ke změně podmínky, tedy že například zařízení bylo v lokaci, ale už není, tak se na server publikuje zpráva pod tématem `<location>/proximity/<user>` o aktuálním stavu přítomnosti daného uživatele. Tato smyčka toto chování opakuje periodicky každých 60 vteřin.

6.3 Agentní termostatická hlavice

Řízení termostatické hlavice je implementováno za pomocí modulo NodeMCU, pomocí vývojového prostředí Arduino[3] a programovacího jazyka Arduino, založeném na jazyce C++.

Při spuštění mikrokontroléru se první definují a deklarují globální proměnné, jako jsou například potřebné informace k připojená k WiFi síti, či cílová adresa MQTT serveru. Pin z něhož se budou získávat data ze senzoru DHT se inicializuje za pomocí Adafruit knihovny $DHT - sensor - library$[4], vytvořením objektu `DHT`, kterému je třeba při definici předat parametry o typu senzoru DHT, neboť knihovna podporuje ovládání všech senzorů typu DHT, DHT 11, DHT 21 (AM2301), DHT 22 (AM2302) a AM2321. Při návrhu byl použit senzor DHT22, je tedy třeba zadat jako parametr `DHTTYPE` hodnotu `DHT22`, dále je nutné specifikovat pin, ze kterého se budou za chodu programu získávat naměřené hodnoty, zvoleným pinem je `D7`, který se nachází ve stabilních stavech při inicializaci nebo resetu modulu NodeMCU. Dále je potřeba inicializovat klienta pro MQTT komunikaci, pomocí knihovny *PubSubClient*[5].

Následná implementace se skládá z dvou hlavních funkcí – `setup` a `loop`. Funkce `setup` probíhá pouze jednou během chodu programu a to hned po spuštění modulu a nebo po jeho

[1] `https://www.raspberrypi.com/products/raspberry-pi-1-model-b-plus/`
[2] `https://github.com/IanHarvey/bluepy`
[3] `https://www.arduino.cc/en/software`
[4] `https://github.com/adafruit/DHT-sensor-library`
[5] `https://github.com/knolleary/pubsubclient`

resetu. Tato funkce má za úkol inicializaci stavu modulu a jeho prvků. Po jejím dokončení se začne provádět cyklická funkce `loop` po zbytek běhu mikrokontroléru, která obsahuje hlavní logiku prováděného programu.

Ve funkci `setup` je potřeba zprvu inicializovat jednotlivé piny, ovládající připojené moduly jako čtecí nebo zapisovací. Pin `D1` a `D2` se nastaví jako zapisovací, protože se jimi bude ovládat směr pohybu motoru. Pin `D7` se nastaví na čtecí, z tohoto pinu se budou číst hodnoty z teplotního senzoru. Následně začne připojování k zadané síti WiFi s použitím knihovny $ESP8266WiFi$. Po úspěšném připojení mikrokontroléru k WiFi síti si MQTT klient nastaví adresu serveru a také callback funkci, která se má použít při přijetí zprávy na odebírané téma. Tato funkce obsahuje přečtení obdržené zprávy a její použití pro požadovanou hodnotu, pokud je hodnota řetězec `off`, poté na hlavici nemá nikdo žádný požadavek a zvolí se výchozí požadovaná hodnota z proměnné `off_tmp`. Jako poslední ve funkci `setup` se načte 128 bytů paměti EEPROM (Electrically Erasable Programmable Read-Only Memory), použitím funkcí knihovny $EEPROM$[6], ačkoliv modul NodeMCU EEPROM paměť neobsahuje, knihovna je schopná ji emulovat na paměti flash, knihovna je schopna emulovat až 4096 bytů paměti EEPROM. Po načtení se přečtou 99., 100. a 101. byte (byty se zde číslují od nuly), obsahuje-li 99. a 101. byte hodnotu 111, tak se na 100. bytu nachází aktuální pozice ventilu, která se načte do proměnné, která indikuje pozici ventilu. Pokud 99. a 101. byte neobsahuje hodnotu 111, tak se jedná o prvotní aktivaci hlavice a je třeba ventil dostat do nulové pozice, tudíž se aktivuje motorek, aby se ventil úplně uzavřel. Po této akci končí funkce `setup`.

Ve funkci `loop` se vždy na začátku kontroluje, zda je MQTT klient připojen k serveru, pokud není, tak se pokouší o připojení a následný odběr k tématu `<location>/set`. Při prvním cyklu funkce se ihned získá naměřená hodnota z teplotního senzoru, která se poté uloží do pole posledních tří naměřených teplot. Naměřená hodnota se následně také publikuje pod tématem `<location>/get`. Poté zkontroluje zda nemá nastat změna pozice ventilu funkcí `control_valve`, funkce si nejdříve zkontroluje, zda za poslední tři měření neklesla teplota o 1.0 stupňů Celsia (pouze pokud jsou k dispozici tři poslední měření, tedy nejdříve až po šesti minutách chodu), což by indikovalo otevření okna a ventil by se na 6 minut zavřel, po této době by se ovládání vrátilo do normálu. Pokud otevřené okno nebylo detekováno, tak se nastaví ventil podle vytápěcího algoritmu využívajícího PID regulátoru[7], implementováno je také ovládání pomocí jednoduchého hysterezního algoritmu, to však zatím není použito. Algoritmus PID regulátoru pracuje dle rovnice 6.1.

$$Kp * error + \sum_{n=0}^{x}(Ki * error_n * time) + Kd * (temp_i - temp_{i-1})/time \tag{6.1}$$

V rovnici 6.1 hodnoty K značí jednotlivé složky regulátoru, tedy Kp je proporcionální složka, Ki je integrační složka a Kd značí derivační složku, $error$ je rozdíl mezi žádanou a aktuální hodnotou, $temp_i$ je aktuální teplota v čase i a $time$ je doba v minutách mezi měřeními, výsledek je zaokrouhlen na celé číslo a ořezán do intervalu $\langle 0; 30 \rangle$, ten se poté použije pro nastavení pozice hlavice, kde 0 značí uzavřený ventil a tedy neprochází žádná teplá voda do radiátoru, a 30 značí úplně otevřený ventil, teplá voda prochází bez omezení. Po získání hodnoty pro nastavení ventilu se nastaví signály pro řízení motoru na potřebnou dobu, tak aby bylo dosaženo získané pozice. Když se nejedná o první cyklus, tak se tato sekvence úkonů opakuje každé tři minutu.

[6] `https://github.com/esp8266/Arduino/tree/master/libraries/EEPROM`

[7] Seznámení s PID regulací: `https://www.ni.com/cs-cz/shop/labview/pid-theory-explained.html`

Kapitola 7

Ověření funkčnosti

V této kapitole je rozebráno ověřování funkčnosti jednotlivých modulů systému a také systém jako celek. V první části je popsán způsob testování modulu pro snímání přítomnosti uživatele. Další část se zaobírá ověřováním funkčnosti vytvořené termostatické hlavice. A jako poslední je popsán způsob ověřování funkce celého systému v reálném prostředí.

7.1 Modul snímání přítomnosti uživatele

Modul pro detekci přítomnosti uživatelů, pomocí zařízení podporující Bluetooth LE, byl testován na několika zařízeních stejného výrobce.

Pro účely testování byl vytvořen skript, který specifické zařízení snímá a ukládá nasnímané hodnoty *RSSI*. Tento skript lze také použít pro zjištění hodnoty *RSSI* ve vzdálenosti jednoho metru, která je potřebná pro výpočet vzdálenosti.

Zařízení byla testována v různých vzdálenostech a za různých podmínek, příklad měření je v tabulce 7.1. Z příkladu lze vyčíst, že *RSSI* v jednom metru je průměrně -69, tato hodnota lze použít jako konstantní při výpočtu vzdálenosti rovnicí 5.2, pomocí této hodnoty a dalších naměřených hodnot lze také aproximovat hodnotu faktoru prostředí, v přirozeném prostředí ji nelze přesně dopočítat, protože v různých vzdálenostech se nachází rozdílné překážky, které signál zkreslují, je tedy potřeba tuto hodnotu vybrat přibližně.

1 m	**3 m**	**3 m (za zdí)**	**4 m**
-71	-76	-87	-78
-81	-71	-86	-80
-68	-71	-87	-83
-66	-76	-86	-85
-66	-76	-85	-83
-65	-74	-86	-83
-69	-84	-87	-83
-67	-75	-87	-83
-67	-76	-87	-83
-71	-77	-88	-84

Tabulka 7.1: Příklad naměřených hodnot *RSSI* v různých vzdálenostech

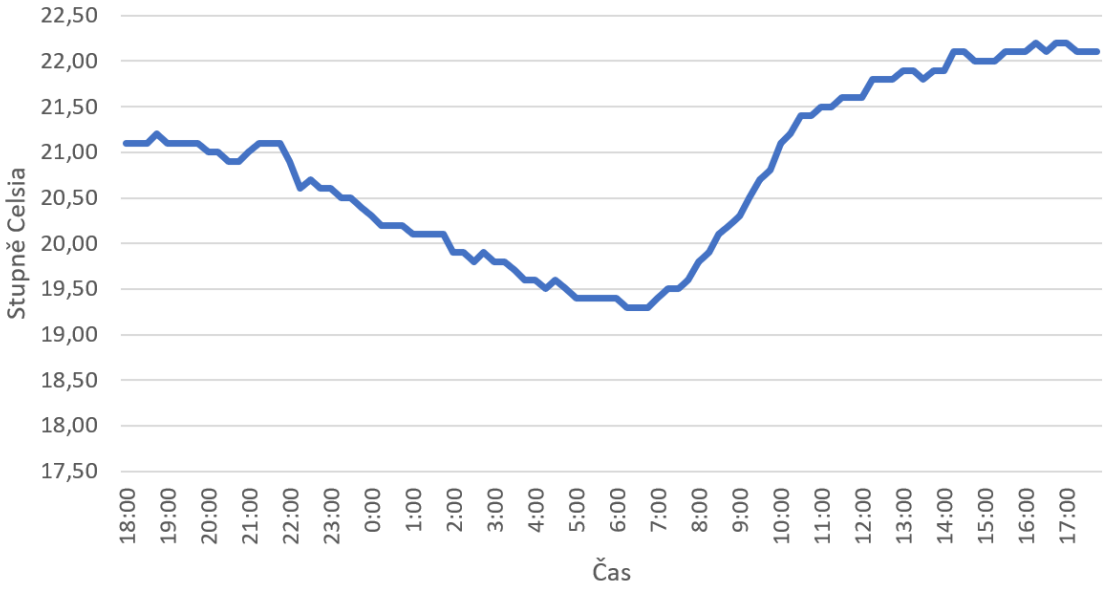
Lze vidět, že velkým problémem jsou stěny a podobné objekty, které zakrývají velkou část prostoru mezi zařízením a modulem. Při měření z tabulky s označením za zdí, bylo

zařízení pouze za rohem vůči modulu a i přesto docházelo k velkým rozdílům oproti měření ve stejné vzdálenosti, ale bez většího zastínění.

Až na nepřesnosti způsobené překážkami zastiňujícími zařízení vůči modulu, snímání přítomnosti dle testovacích měření funguje správně a relativně stabilně. Občas se hodnota *RSSI* výrazněji liší od průměru v daném umístění, toto řeší periodické skenování zařízení, takže jedna špatná hodnota nemá velký dopad na chod systému.

7.2 Termostatická hlavice

Agentní hlavice byla testována zkušebním chodem a také společně s nasazením celého systému. Bylo také provedeno manuální testování pro ověření správné funkce topného algoritmu a komunikace s dalšími prvky systému.



Obrázek 7.1: Příklad řízení teploty termostatickou hlavicí

Na obrázku 7.1 je zobrazen graf, který reprezentuje průběh teploty v čase, která byla naměřena při běhu celého systému, průběh teploty správně vyobrazuje nastavení systému, které je popsáno v 7.3.

7.3 Nasazení systému v reálném prostředí

Celý systém byl otestován nasazením v reálném prostředí. Testovací systém se skládal z multiagentího systému obsahujícího dva uživatele a jeden aktor/senzor, kterým byla termostatická hlavice, v prostředí se také tato hlavice nacházela společně s modulem pro detekci přítomnosti osob, systém byl tedy nasazen pro jednu místnost, kterou byla ložnice/pracovna.

Testování probíhalo po dobu tří týdnu, kdy první dva týdny byly vyhrazeny pro manuální ovládání, aby se systém byl schopen naučit uživatelským preferencím a další týden byl řízen automaticky s občasnými uživatelskými vstupy, když požadoval jiné nastavení. Při učení systému byla snaha udržovat návyky stejné, ale nebyl na to kladen velký důraz,

aby používání odpovídalo reálnému používání. Ve výsledku průměrně uživatel se systémem interagoval třikrát denně, ale při potřebě jiného nastavení i vícekrát.

V průběhu učení se do systému zadávaly hodnoty, dle prováděné činnosti a času. Pro spánek trvající v průměru od 22 hodin do 7 hodin, se nastavovala hodnota 19 stupňů Celsia, pro dopolední hodiny hodnota 22 stupňů Celsia, a pro odpoledne až do večera hodnota 21 stupňů Celsia. Ačkoliv se stávalo, že tyto etapy měly v různých dnech trochu jiné rozsahy a průběhy, tak se systém naučil vybírat akce, které odpovídali nejčastěji volené hodnotě při učení, jak vyplývá z tabulky 7.2. Během automatického režimu nebylo nutné měnit vybíraná nastavení, protože v základě korespondovalo s požadavky uživatele.

Interval	**Hodnota**	**Q-hodnota**	**Počet zvolení**
7:00	19	0.75	13
7:00	22	0.1	1
7:15	19	0.34	4
7:15	22	0.65	10
10:45	22	0.69	11
10:45	21	0.271	3
11:00	22	0.34	4
11:00	21	0.65	10
21:45	21	0.61	9
21:45	19	0.41	5
22:00	21	0.47	6
22:00	19	0.57	8
22:15	21	0.34	4
22:15	19	0.65	10

Tabulka 7.2: Hraniční intervaly automatického výběru hodnoty

Otestována byla také funkčnost režimu přepisu. Bylo zjištěno, že pro změnu preferencí postačují pouze tři až čtyři dny (záleží na Q-hodnotě naučené hodnoty) manuálních změn a nová hodnota bude preferována oproti staré.

Kapitola 8

Závěr

Primárním cílem této diplomové práce bylo navrhnout a implementovat multiagentní systém učící se zvykům uživatelů a na jejich základě maximalizovat jejich komfort v subsystému chytré domácnosti sloužící pro vytápění. Dále vytvořit aktora, který by figuroval v prostředí jako koncový agent přímo ovlivňující prostředí, a modul pro detekci přítomnosti jednotlivých uživatelů v prostředí. Nakonec systém otestovat nasazením v reálném prostředí.

V první části práce je popsána struktura agentního systému a jak vypadají a jaké vlastnosti mají jednotlivé části, následuje popis jednotlivých zavedených architektur agentů a jejich výhod a nevýhod, dále je přiblížen multiagentní systém, jaké způsoby komunikace mezi agenty mohou probíhat a způsoby jakým může být tato komunikace realizována, jako poslední je představena softwarová platforma JADE, její architektura a způsob jakým realizuje běh agentů. Další část je zaměřena na posilované učení a jeho základní koncepty, jsou zde zmíněny metody jako je Q-Learning, metody Monte Carlo, metody hlubokého posilovaného učení a někteří jeho zástupci. Následující část pojednává o existujících přístupech, řešících problematiku chytré domácnosti s pomocí multiagentních systémů, jsou zde představeny projekt MavHome, který využívá několika prediktivních algoritmů pro rozpoznání různých sekvencí činností uživatele, dále projekt využívající posilovaného učení (hluboké neuronové sítě) pro jednoduchou predikci, kdy se má zapnout/vypnout určitý aktor v chytré domácnosti, a jako poslední multiagentní systém pro inteligentní budovy, který je řízen flexibilními fuzzy pravidly pro optimální nastavení. Poslední část se zabývá návrhem a implementací jednotlivých částí systémů, a následně ověřením funkčnosti těchto částí.

V rámci práce byl vytvořen multiagentní systém, který umožňuje skupině uživatelů ovládat jednotlivá vytápění v prostředí, na základě jejich požadavků a nebo pomocí naučených vzorů, které jsou učeny algoritmem Q-Learning pro časové intervaly v rámci dne při každé interakci uživatele se systémem, uživatelské vstupy jsou možné pomocí vytvořeného grafického rozhraní. Dále byla také vytvořena termostatická hlavice, která v prostředí figuruje jako agent, tedy dle nastavení ze systému autonomně řídí vytápění, aby ho bylo dosaženo. Posledním vytvořeným modulem je detekce přítomnosti uživatele, která je realizována za pomocí snímání síly signálu zařízení s technologií Bluetooth Low Energy.

Systém je navrhnut tak, aby umožňoval jednoduché přidání dalších zařízení. V rámci budoucích rozšíření by bylo vhodné zdokonalit modul přítomnosti uživatelů například přidáním senzorů pohybu, aby se zamezilo detekci přítomnosti, když je uživatel v dosahu, ale ne přítomen v dané lokaci. Také by bylo užitečné systém rozšířit o predikci přítomnosti uživatele, aby bylo dosaženo uživatelského komfortu rychleji, přesněji a bylo by možné dosáhnout nižší spotřeby, když by se například neměnilo nastavení při krátké návštěvě dané

lokace. Dále vytvoření mobilní aplikace, kterou by bylo možné zadávat požadavky na řízení systému pomocí MQTT komunikace, nebo přímo s agenty v multiagentním systému, pro jednodušší ovládání, nebo systém přímo propojit s fyzickými rozhraními v chytré domácnosti.

Literatura

[1] *The Foundation for Intelligent Physical Agents*. 2022. Dostupné z: http://www.fipa.org.

[2] Alam, M. R., Reaz, M. B. I. a Ali, M. A. M. A Review of Smart Homes—Past, Present, and Future. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*. 2012, sv. 42, č. 6, s. 1190–1203. DOI: 10.1109/TSMCC.2012.2189204.

[3] Anthony, P., Soon, G., On, C., Alfred, R. a Lukose, D. Agent Architecture: An Overview. *TRANSACTIONS ON SCIENCE AND TECHNOLOGY*. 2014, s. 18–35.

[4] Bednařík, R. *Digital TRV with WiFi communication*. 2021. Dostupné z: https://github.com/xbedna74/dtrv_wifi.

[5] Bellifemine, F., Caire, G., Rimassa, G., Poggi, A., Bergenti, F. et al. *JAVA Agent DEvelopment Framework*. 2023. Dostupné z: https://jade.tilab.com.

[6] Bellifemine, F., Caire, G. a Greenwood, D. *Developing Multi-agent Systems with JADE*. Wiley, 2007. ISBN 978-0-470-05747-6.

[7] Cook, D., Youngblood, M., Heierman, E., Gopalratnam, K., Rao, S. et al. MavHome: an agent-based smart home. In: *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003. (PerCom 2003)*. 2003, s. 521–524. DOI: 10.1109/PERCOM.2003.1192783.

[8] Mirra, J., Silva, F. a Analide, C. Reinforcement Learning Based Approach for Smart Homes. In: *Intelligent Environments 2018*. IOS Press, 2018, s. 38–47.

[9] Naji, H. R., Meybodi, M. N. a Falatouri, T. Intelligent building management systems using multi agents: Fuzzy approach. *International Journal of Computer Applications*. Citeseer. 2011, sv. 14, č. 6, s. 9–14.

[10] Sandholm, T. W. Distributed rational decision making. *Multiagent systems: a modern approach to distributed artificial intelligence*. 1999, s. 202–204.

[11] Shah, R. *Convert RSSI Value of the BLE (Bluetooth Low Energy) Beacons to Meters* [online]. 2021 [cit. 2023-02-27]. Dostupné z: https://medium.com/beingcoders/convert-rssi-value-of-the-ble-bluetooth-low-energy-beacons-to-meters-63259f307283.

[12] Singh, S. *A Comprehensive Guide to Neural Networks in Deep Q-Learning* [online]. Turing Enterprises Inc, Oct 2022 [cit. 2023-02-11]. Dostupné z: https://www.turing.com/kb/how-are-neural-networks-used-in-deep-q-learning.

[13] Sutton, R. S., Barto, A. G. et al. *Introduction to reinforcement learning*. MIT press Cambridge, 1998.

[14] Team, T. H. *MQTT Essentials* [online]. 2020 [cit. 2023-03-02]. Dostupné z: https://www.hivemq.com/tags/mqtt-essentials/.

[15] Townsend, K. *Introduction to Bluetooth Low Energy* [online]. 2014 [cit. 2023-03-04]. Dostupné z: https://learn.adafruit.com/introduction-to-bluetooth-low-energy.

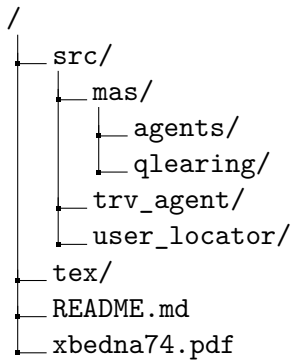
[16] Weng, L. *A (long) peek into reinforcement learning* [online]. Feb 2018 [cit. 2023-02-11]. Dostupné z: https://lilianweng.github.io/posts/2018-02-19-rl-overview/.

[17] Wooldridge, M. *An Introduction to Multiagent Systems*. 2. vyd. Chichester, UK: Wiley, 2009. ISBN 978-0-470-51946-2.

[18] Zbořil, F. Brno, CZ, 2004. Disertační práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/phd-thesis/72/.

Příloha A

Obsah přiloženého paměťového média



- složka src obsahuje zdrojové kódy jednotlivých částí systému
- složka tex obsahuje zdrojové kódy a potřebné soubory pro vyhotovení textu práce
- soubor `README.md` obsahuje potřebné informace k instalaci a zprovoznění