VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE ANOMÁLIÍ ZA BĚHU VIRTUÁLNÍ ELEKTRÁRNY

DETECTION OF ANOMALIES DURING THE OPERATION OF A VIRTUAL POWER PLANT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAN VYMAZAL**
AUTHOR

VEDOUCÍ PRÁCE **FRANTIŠEK ZBOŘIL, doc, Ing., Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Vymazal Jan**
Program: Informační technologie
Název: **Detekce anomálií za běhu virtuální elektrárny**
Virtual Power Plant Anomaly Detection
Kategorie: Modelování a simulace

Zadání:

1. Seznamte se s fungováním elektrárny s kogeneračními jednotkami a s měřenými veličinami během jejich provozu.
2. Na základě dodaných dat navrhněte metody, jak předvídat možnou poruchu v tomto systému, například poruchu zážehové svíčky pro ohřevné kotle.
3. Implementujte systém s využití těchto metod a ověřte jeho schopnost detekce poruch.
4. Diskutujte, v jakých případech a s jakým předstihem je možné úspěšně předvídat poruchy a jaká je využitelnost Vašeho systému v praxi.

Literatura:

- Russell, S., Norwig, P.: Artificial Intelligence: A Modern Approach, Pearson, 2016
- Lara-Benítez, P., Carranza-García, M,. C. Riquelme, J.: An Experimental Review on Deep Learning Architectures for Time Series Forecasting, International Journal of Neural Systems, Vol. 31, No. 3.2021

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

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

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Tato bakalářská práce se zabývá implementací multiagentního systému pro detekci a predikci anomálií za běhu virtuální elektrárny. Implementována je rovněž simulace, která vkládá tento multiagentní systém do prostředí, které reflektuje postupné přidávání dat v reálném světě. Rovněž se zabývá principy komunikace mezi agenty v multiagentním prostředí dle standardů FIPA. V rámci práce byl ve frameworku JADE v jazyce Java vytvořen onen multiagentní systém a v jazyce Python skript, který implementuje onu simulaci.

Abstract

This bachelor thesis deals with the implementation of a multi-agent system for the detection and prediction of anomalies during the operation of a virtual power plant. The thesis also deals with the implementation of a simulation that puts this multi-agent system into an environment that reflects the gradual addition of data in the real world. It also deals with the principles of communication between agents in a multi-agent environment according to FIPA standards. As part of the work, I created the multi-agent system in the JADE framework in the Java programming language and a script in the Python programming language that implements the simulation.

Klíčová slova

MAS, multiagentní systémy, virtuální elektrárna, detekce anomálií v časové řadě

Keywords

MAS, multi-agent systems, virtual power plant, anomaly detection in time series

Citace

VYMAZAL, Jan. *Detekce anomálií za běhu virtuální elektrárny.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce František Zbořil, doc, Ing., Ph.D.

Detekce anomálií za běhu virtuální elektrárny

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana docenta Zbořila. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jan Vymazal
3. května 2022

Poděkování

Chtěl bych poděkovat panu Františku Zbořilovi, doc. Ing., Ph.D. za vedení a odbornou pomoc s vypracováním bakalářské práce

Obsah

Kapitola 1

Úvod

1.1 Kogenerační jednotka

Má práce se zabývá detekcí a předvídáním anomálií za běhu virtuální elektrárnou. Pod označením virtuální elektrárna se skrývá kogenerační jednotka. Kogenerační jednotka je zařízení, které souběžně generuje elektrickou energii a teplo. Primárním účelem této jednotky je generování tepla. Na rozdíl od klasických elektráren, kde je teplo vytvořeno, ale není nijak využíváno a je pouze vypuštěno do ovzduší, zde se vytvořené teplo využije pro ohřev. Hlavní výhodou využití kogenerační jednotky oproti jiným zdrojům energie je výrazně vyšší využití energie obsažené v palivu. Palivem pro kogenerační jednotku může být zemní plyn či bioplyn. Pro generování elektřiny používá kogenerační jednotka spalovací motor se zážehovou svíčkou, upravený tak, aby byl schopný spalovat plyn. [9]

1.2 Formát dat

Má práce používá data dodané od společnosti TEPLO Břeclav s.r.o. TEPLO Břeclav zaznamenává stav kotelny každých dvanáct sekund. V jednom záznamu se nachází atribut Datum a čas, kde čas je měřen s přesností na sekundy. Pak hodnoty požadovaného a dodávaného výkonu, které jsou pro mou práci velmi podstatné. Následně jsou důležité ještě teploty každé zážehové svíčky (sloupce M až AB) na každém z šestnácti kotlů. Svíčky nesou označení A1 až A8 a B1 až B8. Hodnoty výkonu i hodnoty teploty jsou uloženy jako celá čísla. Kromě toho se ještě zaznamenávají hodnoty tlaku plynu, teploty oleje a další, které však nejsou pro mou práci příliš podstatné. Příklad hodnot za běhu elektrárny je vidět na obrázku 1.1. Data, která jsem měl k dispozici, jsem dostal v souboru s příponou csv, kde oddělovači mezi jednotlivými atributy byly středníky. Prvním řádkem v tomto souboru byla vždy legenda. Záznamy byly seřazeny od nejnovějšího po nejstarší.

1	Datum a čas	Act power [kW]	ActPwrReq [kW]	T461 cyl A1 [°C]
3913	07.09.2021 10:57:20	801	800	345
3914	07.09.2021 10:57:08	804	800	340
3915	07.09.2021 10:56:56	807	800	357
3916	07.09.2021 10:56:44	808	800	343

Obrázek 1.1: Typická forma dat. Ve sloupci A je datum, ve sloupci B a C výkon v kW, ve sloupci M teplota na svíčce A1 ve stupních Celsia.

1.3 Anomálie

Nejzásadnější indikací, že v systému je problémová svíčka, jsou propady dodávaného výkonu oproti požadovanému výkonu. Podle prvotních poskytnutých informací se jedná o propady o 5 a více procent (s touto hladinou jsem poté pracoval v kapitole pokusy). K indikaci, že se skutečně jedná o závadu a že dochází k tvrdému vypnutí systému, lze využít hodnotu požadovaného výkonu. V případě, že se elektrárna ukončuje korektně (dosáhla svého cíle), dojde nejprve ke snížení požadovaného výkonu na 1 kW, počká se až dodávaný výkon klesne a až pak se požadovaný i dodávaný výkon změní na 0 kW. To je vidět na obrázku 1.2. V případě, že došlo k anomálii a tvrdému vypnutí, se však požadovaný výkon změní na 0 kW okamžitě. To je patrné na obrázku 1.3.

1	Datum a čas	Act power [kW]	ActPwrReq [kW]
893	07.09.2021 21:01:32	0	0
894	07.09.2021 21:01:20	87	1
895	07.09.2021 21:01:08	126	1
896	07.09.2021 21:00:56	160	1
897	07.09.2021 21:00:44	196	1
898	07.09.2021 21:00:32	231	1
899	07.09.2021 21:00:20	263	1
900	07.09.2021 21:00:08	299	1
901	07.09.2021 20:59:56	332	1
902	07.09.2021 20:59:44	364	1
903	07.09.2021 20:59:32	400	1
904	07.09.2021 20:59:20	446	1
905	07.09.2021 20:59:08	477	1
906	07.09.2021 20:58:56	520	1
907	07.09.2021 20:58:44	565	1
908	07.09.2021 20:58:32	613	1
909	07.09.2021 20:58:20	661	1
910	07.09.2021 20:58:08	729	1
911	07.09.2021 20:57:56	775	1
912	07.09.2021 20:57:44	788	1
913	07.09.2021 20:57:32	803	1
914	07.09.2021 20:57:20	808	800

Obrázek 1.2: Příklad standardního ukončení běhu elektrárny. Požadovaný výkon nejprve klesne na jedna, a pak společně s dodávaným výkonem na nulu.

1	Datum a čas	Act power [kW]	ActPwrReq [kW]
4524	07.09.2021 8:55:00	0	0
4525	07.09.2021 8:54:48	13	800
4526	07.09.2021 8:54:36	14	800
4527	07.09.2021 8:54:24	14	800
4528	07.09.2021 8:54:12	15	800
4529	07.09.2021 8:54:00	17	800
4530	07.09.2021 8:53:48	17	800
4531	07.09.2021 8:53:36	19	800
4532	07.09.2021 8:53:24	23	800
4533	07.09.2021 8:53:12	32	800
4534	07.09.2021 8:53:00	62	800
4535	07.09.2021 8:52:48	105	800
4536	07.09.2021 8:52:36	146	800
4537	07.09.2021 8:52:24	191	800
4538	07.09.2021 8:52:12	240	800
4539	07.09.2021 8:52:00	278	800
4540	07.09.2021 8:51:48	317	800
4541	07.09.2021 8:51:36	365	800
4542	07.09.2021 8:51:24	421	800
4543	07.09.2021 8:51:12	491	800
4544	07.09.2021 8:51:00	595	800
4545	07.09.2021 8:50:48	702	800

Obrázek 1.3: Příklad anomálie. Elektrárna postupně přestávala dodávat výkon, i když se požadovaný výkon neměnil. Před ukončením požadovaný výkon spadl rovnou na nulu.

Kapitola 2

Teoretický úvod

Pro řešení problému jsem se po konzultaci s vedoucím rozhodl ustoupit od původně plánovaných statistických funkcí a přejít k multiagentním systémům. Pro teoretický úvod je vhodné uvést alespoň základní pojmy této problematiky.

Autonomním agentem lze rozumět aktivní prvek vytvořený člověkem, který samostatně pracuje v prostředí, jehož je součástí, a které vnímá, a kde vykonává akce za účelem splnění svých delegovaných cílů [14]. Zjednodušeně tedy lze říct, že agent svými senzory vnímá vjemy a podněty z prostředí, na jejich základu se samostatně rozhodne, jakou akci provést a zpětně ovlivní prostředí svými efektory[11].

2.1 Typy agentů

Nejjednodušším typem agentů jsou čistě reaktivní agenti. Tento typ agentů nemá vnitřní stav a rozhoduje se pouze na základě právě přijatých vjemů ze senzoru, přičemž nijak nebere v potaz historii přijatých vjemů [12]. V našem případě je reaktivní chování například detekce již vzniklé anomálie při nekorektním ukončení běhu elektrárny, kdy požadovaný výkon padne z vyšší hodnoty přímo na nulu bez postupného poklesu přes požadovaný výkon jedna, či kdy je dodávaný výkon výrazně nižší než hodnota požadovaného. Nicméně zde vystává problém, že senzor agenta sice zachytí, že je požadovaný výkon nula, případně že se liší od dodávaného, nicméně tento podnět ještě nemusí znamenat, že se jedná o anomálii, protože může být elektrárna vypnutá (tedy je nula validní) či může elektrárna startovat (tedy je dodávaný výkon výrazně nižší než požadovaný). Proto je potřeba představit reaktivní agenty s modelem chování.

Dalším typem agentů jsou reaktivní agenti s modelem chování. Tento agent v sobě uchovává vnitřní stav, který je určen předchozími záznamy senzorů. Pro výběr vhodné akce pak tedy nevyužije pouze výstup senzorů, ale i tento stav[12]. Tímto způsobem lze vyřešit obtíže, které jsem popsal v odstavci výše. Nestačí však mít pouze informace o změnách a stavu prostředí, je třeba také pro agenta vytyčit nějaký cíl, proto je na čase představit agenty rozhodující se na základě cíle. Tito agenti k rozhodování používají nejen vjemy ze senzorů, svůj vnitřní stav, ale právě i cíl, kterého má agent dosáhnout. Cíle však lze dosáhnout mnoha různými cestami, toto řeší agenti rozhodující se na základě užitku. V tomto případě je každému stavu daná nějaká hodnota užitku, v případě multiagentní skupiny je pak užitek stavu definován pro každého agenta zvlášť. Agent pak zvolí vždy tu akci, která maximalizuje očekávaný užitek [12] .

Doteď jsem mluvil pouze o jednotlivých agentech, má práce se ale zaobírá multiagentním systémem. Multiagentní systém je systém, kde se v prostředí nachází více agentů a nedochází zde k interakci pouze mezi agentem a prostředím, ale i mezi agentem a jiným agentem, případně více agenty [11]. Agenti mohou mít rozdílné cíle i užitky, a proto mezi nimi mohou nastat tři situace [11]

- agenti mají stejný cíl spolupráce
- agenti mají rozdílný cíl, ale jsou schopni se shodnout na společném nekonfliktním postupu
- agenti jsou v konfliktu. Toto nastává v případě, že užitek agenta v multiagentním prostředí je menší než v prostředí, kde je sám.

Interakce mezi agenty může probíhat buďto přímo nebo nepřímo. Interakce přímá znamená, že jeden agent přímo ovlivní senzory druhého agenta. V tom případě ale musí mít společný slovník, standardizaci zpráv, ontologii a komunikační protokoly. Agenti mohou interagovat i nepřímo pomocí prostředníka, kterým je jiný agent, který předává zprávy mezi dvěma účastníky, případně skrz prostředí pomocí změny stavu sdíleného prvku prostředí [11]. V našem případě mají agenti stejný cíl spolupráce a komunikují napřímo, případně skrze prostředníka (více v kapitole Architektura multiagentního systému).

2.2 Prostředí

Prostředí, ve kterém agenti operují, může mít mnoho vlastností:

- Plně pozorovatelné – Agentovy senzory jsou schopné pozorovat všechny prvky v prostředí, a tedy jeho celkový stav [11].
- Částečně pozorovatelné – Existuje neprázdná množina prvků v prostředí, které mohou prostředí ovlivnit, ale agent je není schopen pozorovat [11].
- Statické – Prostředí se mění pouze akcemi agenta [14].
- Dynamické – Prostředí se mění i v době, kdy agent nejedná, a to akcemi, které vykonává něco jiného než agent [14].
- Deterministické – Při stejném stavu prostředí a při stejné akci ve kterémkoliv čase se bude prostředí chovat vždy stejně [14].
- Nedeterministické – V prostředí vznikají anomálie.
- Diskrétní – Časová množina je tvořena výčtem prvků [11].
- Spojité – Časová množina je tvořena prvky spojitého intervalu [11].
- Epizodní – Situace, které agent řeší jsou v atomických „epizodách“ a navzájem se neovlivňují. Následující epizody nejsou ovlivněny předešlými epizodami. Prostředí se po skončení epizody vrací do původního stavu [14].
- Sekvenční – Situace a akce agenta se vzájemně dlouhodobě ovlivňují [14].

Virtuální elektrárna je prostředí, které je částečně pozorovatelné, dynamické, epizodní (kdy resetujícím faktorem je buďto vypnutí, nebo výměna zničené svíčky), nedeterministické a spojité. Nicméně data, která jsou mi dodávána a se kterými agenti pracují, jsou již prostředí plně pozorovatelné, dynamické, nedeterministické a diskrétní.

2.3 FIPA

FIPA – The Foundation for Intelligent Physical Agents je orgán pro vytváření standardů pro inteligentní agenty a multiagentní systémy. Hlavním cílem tohoto orgánu je „Propagace technologií a specifikací metodik spolupráce, které usnadňují end-to-end spolupráci mezi inteligentními agenty a multiagentními systémy v moderních komerčních a průmyslových prostředích“ [1].

FIPA byla založena v roce 1996 ve Švýcarsku jako nezisková organizace. V roce 2005 došlo k přeměně z neziskové organizace na výbor pro standardy IEEE. Od založení bylo její součástí více než 60 členů – například IBM, Sun Microsystems či Hewlett Packard. FIPA po svém vzniku vytvořila specifikace, které prošly několika cykly znovuvyhodnocení a úprav. V roce 2002 bylo vyhodnoceno, že základní specifikace jsou natolik stabilní, vyspělé a dobře pochopené, že mohou být použity v komerčním prostředí, a došlo k jejich standardizaci [2]. Standardy FIPA se používají v oblasti správy životního cyklu agentů, transportu meziagentních zpráv, struktury těchto zpráv, meziagentních interakčních protokolů, ontologií a zabezpečení [8]. Nejvíce rozšířeným a používaným standardem je FIPA-ACL (Agent Communication Language), tedy standardy komunikace mezi dvěma a více agenty a standardy pro správu agentů [8].

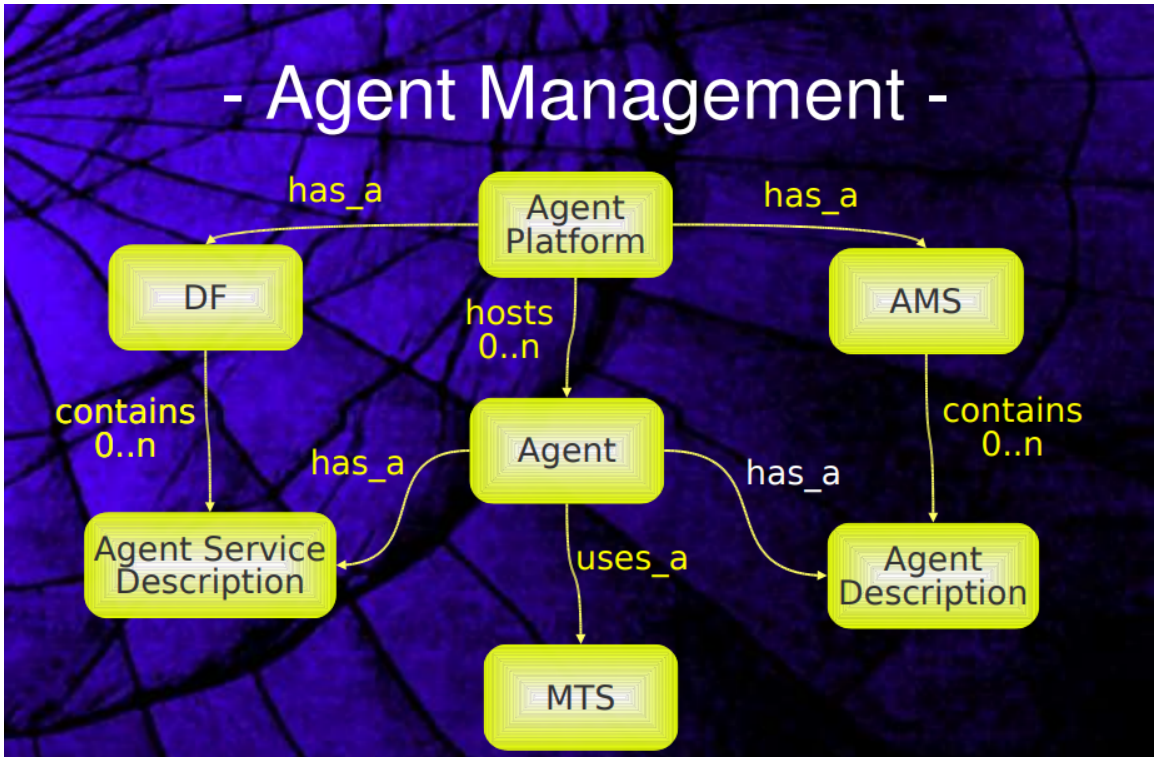
Dle FIPA standardů má agentní platforma systém pro správu agentů (Agent Management Systém (AMS)), sprostředkovatele adresáře (Directory Facilitator (DF)) a hostuje nula až n agentů. Agent má popis služby agenta a popis agenta a pro komunikaci využívá službu pro přenos zpráv (Message Transport Serivce (MTS)). Popis služby agenta obsahuje jméno agenta, jeho lokaci, jméno služby, protokoly, ontologie a dobu platnosti [8]. Popis agenta obsahuje jeho jméno, vlastníka a stav [8]. DF obsahuje všechny popisy služeb agentů a AMS obsahuje všechny popisy agentů. Diagram závislostí mezi prvky agentní platformy je vyobrazen na obrázku 2.1.

Základními operacemi nad DF i AMS je REGISTER pro vytvoření nového popisu agenta či popisu služeb agenta, DEREGISTER pro zrušení popisu, MODIFY pro změnu popisu a SEARCH pro vyhledání agenta s daným popisem. MTS je poskytována agentům jejich agentní platformou a umožňuje zasílání ACL zpráv mezi agenty na jejich či jiné agentní platformě. Samotný fyzický přenos zpráv mezi agenty probíhá podle Message Transport Protocolu. FIPA zpráva obsahuje obálku a obsah, kde obsah obsahuje parametry ACL zprávy a samotnou zprávu. Povinnými parametry obálky jsou [8]:

- *to*, který označuje adresáta
- *from*, který označuje odesilatele
- *acl-representation*, který informuje o formátu ACL zprávy (jestli je string, XML či Bit-efficient)
- *date*, které obsahuje datum vytvoření obálky

Volitelnými parametry jsou [8]

- *payload-lenght*, informující o bajtové délce obsahu zprávy
- *payload-encoding*, informující o kódování obsahu zprávy
- *security-object*, obsahující informace o šifrování a certifikátu
- *received*, který potvrzuje přijetí zprávy



Obrázek 2.1: Diagram závislostí agentní platformy. Převzato z [8]

Parametry ACL zprávy mají elementy [8]:

- *performative*, určující jakou akci zpráva provádí (například INFROM)
- *sender*, obsahující odesilatele zprávy
- *receiver*, obsahující adresáta
- *reply-to*, obsahující adresáta odpovědi na zprávu
- *language*, informující o jazyku použitém v samotném obsahu zprávy
- *encoding*, určující kódování obsahu zprávy
- *ontology*, určující ontologii obsahu zprávy
- *protocol*, definující interakční protokol, ve kterém byla zpráva vygenerována
- *conversation-id*, obsahující id konverzace, jejíž součástí je tato zprávy
- *reply-with*, obsahující výraz, který má agent, který na zprávu odpovídá, umístit do elementu in-reply-to
- *in-reply-to* – viz element reply-with
- *reply-by*, informující o čase, do kdy očekává odpověď
- *content*, obsahující samotný obsah zprávy

Kapitola 3

Využité technologie

3.1 Java

Java je vysokoúrovňový, třídní, objektově orientovaný programovací jazyk se statickým typováním. Byla poprvé vydána v roce 1995 společností Sun Microsystems. Vývoj přešel pod společnost Oracle po jejím získání Sun Microsystems v roce 2010. Jedna z předností tohoto jazyka je myšlenka „write once, run anywhere“, která znamená, že zkompilovaný Java kód lze bez nutnosti znovuzkompilování spustit na jakékoliv platformě [3]. Aby bylo této myšlenky dosaženo, jsou zkompilované Java soubory spouštěny na virtuálním stroji Java virutal machine (JVM). Většina Java technologií, jako jsou knihovny, OpenJDK JVM a referenční implementace překladače od společnosti Oracle, je licencována pod licencí GNU GPL 2.0. Dlouhodobě podporované verze Javy jsou Java 8, 11 a 17. Nejnovější verze Javy, Java 18, vyšla počátkem tohoto roku. Syntax Javy je inspirován jazyky C a C++. Narozdíl od těchto jazyků je však více striktně objektově orientovaný – všechen kód musí být zapsán vevnitř nějaké třídy a všechny proměnné uchovávající data jsou objekty (výjimku tvoří primitivní datové typy jako integer, float, boolean a char, u kterých toto neplatí z výkonnostních důvodů). Java používá automatický garbage kolektor.

3.2 Python

Python je vysokoúrovňový, imperativní, skriptovací programovací jazyk s možností objektově orientovaného přístupu a dynamickým typováním. Python byl poprvé vydán na začátku roku 1991 a je od svého vzniku vyvíjen Guiden van Rossumem a jeho organizací Python Software Foundation. Současná verze Pythonu – Python 3, byla vydána v roce 2008 a není zcela zpětně kompatibilní s předchozími verzemi. Poslední vydaná verze je Python 3.10. Python je navržen tak, aby samotné jádro jazyka bylo malé a rozšířené velkou standardní knihovnou a dalšími rozšiřitelnými moduly [13]. Zároveň je navržen s myšlenkou, že je vhodné, aby se ve zdrojovém kódu dalo snadno číst. Proto používá anglická klíčová slova, kde jiné programovací jazyky používají jiné symboly (například or či and). Na rozdíl od jiných programovacích jazyků (jako je Java či C++) je ohraničení bloků kódu děláno pomocí odsazení místo klasických složených závorek. Python používá automatický garbage kolektor.

3.3 Eclipse

Eclipse je open source vývojové prostředí původně vydáno na konci roku 2001 firmou IBM, nicméně vývoj později přešel pod nezávislou Eclipse Foundation. Eclipse je napsáno v Javě a C a je dostupné na Linuxu, FreeBSD, macOS i Windows. Eclipse je distribuováno pod licencí Eclipse Public License, která je navržena tak, aby byla přívětivější k byznysovým účelům než klasické GNU GPL. Jeho primární využití je tvorba Java aplikací, nicméně ho za pomoci pluginů lze použít i k vývoji aplikací v jiných programovacích jazycích jako je C, C++, C#, JavaScript, PHP, Python a mnoho dalších, a také například ke tvorbě dokumentů skrz rozšíření pro LaTeX. Architektura Eclipse je založena na pluginech. S výjimkou malého runtime jádra je všechno plugin. Základní verze Eclipse SDK (software development kid) však obsahuje i Eclipse Java development tools, které poskytují IDE se zabudovaným překladačem a debuggerem.

3.4 Visual Studio Code

Visual Studio Code je editor zdrojového kódu vyvíjený Microsoftem pro Windows, Linux a macOS. Visual Studio Code je napsán v TypeScriptu, JavaScriptu a HTML a byl poprvé vydán v dubnu roku 2015. Microsoft vydává své Visual Studio Code jako freeware a zároveň zpřístupnil zdrojové soubory na GitHubu pod MIT Licencí, která je plně kompatibilní s GNU GPL licencí, ale zároveň umožňuje pozdější využití v uzavřených aplikacích. Podle ankety na Stackoverflow se jedná o nejpoužívanější vývojový nástroj, který využívá 70 % dotázaných respondentů [4]. Visual Studo Code umožňuje práci s řadou programovacích jazyků, například C++, Python, Go, JavaScript, Node.js či Java a dalšími. V základní verzi podporuje u běžných programovacích jazyků zvýraznění syntaxe, párování závorek, skrývání bloků kódu pro větší přehlednost a automatické doplňovaní kódů. Visual Studio Code používá rozšíření, které lze doinstalovat z centralizovaného uložiště VS Code Marketplace. Rozšíření nabízejí podporu dalších programovacích jazyků, ale i přidávání funkcí jako je například překladač a debugger.

3.5 Apache Ant

Apache Ant (zkratka znamená another neat tool) je nástroj pro automatický překlad a spouštění spustitelných souborů ze zdrojových kódů a knihoven v jazyce Java. Ant používá XML formát k popisu závislostí a procesu překladu kódu. Pro vytvoření tohoto popisu jsem použil možnosti jej vygenerovat přímo z Eclipse IDE, které jsem používal. Apache Ant je distribuován pod licencí Apache License 2.0

3.6 GNU Make

GNU Make je nástroj pro automatický překlad a spouštění (v mém případě pouze spouštění) spustitelných souborů. Make je napsán v jazyce C a je šířen pod GNU GPL v3 licencí.

3.7 Org.json

Data, která jsem měl k dispozici byli uložené v .csv souboru, kdy jeden soubor tvořil jeden den. Pro simulaci běhu elektrárny jsem se rozhodl, že bude vhodné generovat jeden záznam

jako jeden soubor, který bude ve formátu JSON. Pro načtení JSON souborů v Javě jsem použil knihovnu org.json. Tato knihovna přidává třídu JSONObject, jejíž konstruktor počítá s přiřazením argumentu typu String, který obsahuje zpracovávaný JSON záznam. Pro zjištění hodnoty atributu podle jeho klíče lze pak použít metodu getString(klíč). Jedná se o open-source projekt, který je distribuován zdarma bez licenčních omezení, vyjma pravidla, že se nesmí použít pro zlé účely [10].

3.8 JADE

JADE (Java Agent Development Framework) je open-source framework pro implementaci agentů v prostředí jazyku Java. Je kompletně implementován v jazyku Java. Poskytuje takové nástroje, aby v něm vytvoření agenti splňovali standardy dané organizací FIPA (Foundation for Intelligent Physical Agents). Poskytuje abstrakci agentů a také umožňuje peer to peer komunikaci mezi agenty pomoci asynchronních zpráv. Součástí JADE je i grafické rozhraní zobrazující jednotlivé agenty, které je skrz něj možné debuggovat, či jim zadávat alternativní úkoly [6]. JADE je distribuován společností Telecom Italia pod licencí LGPL (Lesser General Public License Version 2).

3.9 Alternativní technologie pro tvorbu multiagentních systémů

Kromě JADE existuje mnoho dalších jazyků a frameworků pro vytváření multiagentních systémů v jazyce Java, Haskell či Python [7].

3.9.1 ASTRA

ASTRA je programovací jazyk pro tvorbu agentů, který kombinuje jazyk AgentSpeak (který je založen na softwarovém modelu belief-desire-intention) a teleo-reaktivního programování (teleo-reaktivní programy provádí všechny své akce tak, aby směřovaly k dosažení stanoveného cíle). ASTRA má blízkou integraci s jazykem Java, do nějž jsou agenti kompilovaní a na jehož základu je jazyk založený. ASTRA je distribuována pod licencí GPL3.0. [5]

3.9.2 Chromar

Chromar je programovací jazyk zabudovaný do jazyku Haskell. Má rule-based notaci, kdy pravidla popisují chování jednoho agenta či synchronizované akce více agentů.

3.9.3 GOAL

GOAL je deklarativní programovací jazyk pro tvorbu agentů. GOAL agenti volí prováděnou akci na základě svých domněnek (beliefs) a cílů (goals). Pravidla agentů jsou zapsaná v syntaxi jazyka prolog. GOAL je propojený s jazykem Java.

3.9.4 Gwendolen

Gwendolen je programovací jazyk založený na jazyku Java a jeho agenti se řídí belief-desire-intention modelem. Byl vytvořen za účelem vytváření verifikovatelných agentů, což ovšem znamená, že samotní agenti mají z tohoto důvodu pouze limitovanou funkčnost. Gwendolen je distribuován pod licencí LGPL 3.0.

3.9.5 JaCaMo

JaCaMo je programovací jazyk, který přináší integraci nástrojů a programovacích jazyků pro programování agentů (nástroj Jason, viz níže), jejich prostředí (framework Cartago) a organizaci (nástroj Moise). Je integrován do jazyku Java. JaCaMo je distribuován pod licencí LGPL 3.0.

3.9.6 Jadex

Jadex je framework umožňující programování inteligentních jazyků v XML a Javě. Je založen na modelu belief-desire-intention. Na rozdíl od jiných inteligentních agentů, je komunikace dělána použitím invokací služeb.

3.9.7 Jason

Jason je rozšíření jazyku AgentSpeak. Agenti v Jasonu reagují na události v systému provedením akcí v prostředí podle jejich plánu, uloženém v knihovně plánů každého agenta. Jedním z rozšíření oproti původní variantě je přidání pravidel v syntaxi jazyka Prolog, které mohou být přidány a použity v belief bázi agenta.

3.9.8 LightJason

LightJason je inspirován jazykem AgentSpeak a Jason, nicméně je implementován from scratch. Rozšiřuje možnosti AgentSpeaku o lambda výrazy, multi-plan a multi-rule definice, explicitní opravné akce, přiřazení více proměnných a paralelní provádění úkolů.

3.9.9 PADE

PADE (Python Agent Development Environment) je framework pro vytváření, spouštění a správu multiagentních systémů v prostředí jazyka Python. Pro komunikaci mezi agenty využívá knihovnu Twisted. PADE je distribuován pod licencí MIT.

3.9.10 SPADE

SPADE (Smart Python Agent Development Environment) je framework pro vytváření inteligentních agentů v prostředí jazyka Python. Komunikace mezi agenty je dělána ve formátu XMPP. Podporuje také rozšíření, které využívá model Belief-Desire-Intention. SPADE je distribuován pod licencí MIT.

Kapitola 4

Návrh a architektura řešení

4.1 Návrh řešení

Cílem mé práce bylo odhalení a předejití anomálii za běhu elektrárny, které se projevují tak, že elektrárna není schopna dodávat požadovaný výkon, dochází k jejímu odstavení a nutnosti manuální opravy a s těmito obtížemi spojené finanční ztrátě. Zatímco odhalení anomálie je velice snadné – zkrátka pokud dochází k velkému výkyvu v dodávaném výkyvu, pak k ní dochází, u jejího předejití to tak snadné není. Nicméně z dat, které jsem měl k dispozici bylo patrné, že svíčka, která anomálii způsobí, zažije ještě několik minut před výpadkem náhlý pokles teploty.

Po konzultaci s vedoucím jsme se rozhodli daný problém řešit pomocí multiagentního systému a běh programu řešit pomocí simulace, kdy budou programu postupně přidávaná data, jako by to bylo i za běžného provozu. Prvotní návrh počítal s tím, že vytvořím jednoho agenta, který bude sledovat dodávaný a požadovaný výkon, a pak jednoho agenta na každé ze šestnácti svíček. Agenti budou mezi sebou komunikovat a v případě zjištěného rizika vzniku anomálie, případně přímo při vzniku anomálie, korektně jednat.

4.2 Architektura řešení

Vzhledem k časově vázané simulaci jsem se rozhodl, že budu postupně generovat nové záznamy pomocí Python skriptu. Tento Python skript načítá vstupní data ze souboru typu .csv a pro každý záznam vytvoří zvlášť soubor typu JSON. Tyto soubory ukládá do složky input. JSON soubory mají jméno out{ID}.json, kde {ID} je autoinkrementující se proměnná, která se ale vynuluje v případě, že dosáhne 10000. Mezi vytvářením jednotlivých souborů se záznamy je vždy časová prodleva, která délkou odpovídá reálné prodlevě získané z reálných dat. Tato prodleva se dá zmenšit pomocí spouštěcích argumentů programu, viz kapitola Ovládání programu. Načítání vznikajících záznamů obstarává multiagentní systém (viz kapitoly Architektura multiagentního systému a Implementace), který po načtení a zpracování záznamu soubor vymaže. V momentě, kdy se zpracují všechna data a celý proces se má ukončit, vygeneruje Python skript poslední JSON soubor, kde datum nastaví na „end“ a následně se Python skript ukončí. Pokud multiagentní systém načte záznam s touto hodnotou dat, pak se i on ukončí.

4.3 Architektura multiagentního systému

V mém multiagentním systému se vyskytují tři druhy agentů – agent typu VelkySef, agent typu TeplotniSledovac a agent typu VykonAgent

4.3.1 Agent typu VelkySef

Agent typu Velký šéf se v mém multiagentním systému vyskytuje pouze jeden. Jedná se reaktivního agenta s modelem chování. Agent slouží jako nadřazený agent všem ostatním agentům a jedním z jeho úkolů je ostatní agenty spustit. Hlavním úkolem Velkého šéfa je načítání dat s jednotlivými záznamy s daty informujících o stavu a hodnotách atributů elektrárny. Patřičná data z těchto záznamů pak zasílá ostatním agentům (agentům typu Teplotní sledovač zašle hodnotu teploty svíčky, kterou daný agent sleduje, agentům typu Výkon Agent pak hodnoty požadovaného a dodávaného výkonu). Jeho dalším úkolem je sloužit jako prostředník v komunikaci mezi agentem typu Výkon Agent a agenty typu Teplotní sledovač. Posledním nezmíněným úkolem je pak přijímaní a zpracování zpráv od ostatních agentů, které obsahují informace o vzniklých chybách. Na základě těchto zpráv podrobně informuje uživatele o vznikajících či vzniklých anomáliích. Implementační detaily tohoto agenta lze nalézt v kapitole Agent třídy VelkySef.

4.3.2 Agent typu TeplotniSledovac

Agentů typu Teplotní sledovač se v mém multiagentním systému vyskytuje celkem šestnáct. Jedná se o reaktivní agenty s modelem chování. Jejich hlavním úkolem je analýza nové hodnoty teploty na svíčce, která jim byla právě zaslána agentem Velký šéf. Každý agent typu Teplotní sledovač sleduje jednu svíčku elektrárny. Agent si udržuje seznam hodnot teplot na svíčce při současném běhu elektrárny. Pokud běh skončí, tento seznam vymaže. Při analýze porovnává novou hodnotu teploty s průměrem tohoto seznamu a v případě velké odchylky informuje agenta typu Velký šéf. Implementační detaily jsou v kapitole Agent třídy TeplotniSleovac.

4.3.3 Agent typu VykonAgent

Agent typu Výkon agent se v mém multiagentním systému vyskytuje pouze jeden. Jedná se o reaktivního agenta s modelem chování. Agent si udržuje informaci o celkovém stavu elektrárny. Hlavním úkolem agenta je analýza nových hodnot dodávaného a požadovaného výkonu elektrárny (které dostává od Velkého šéfa). Na základě těchto hodnot pak přepíná mezi celkovými stavy elektrárny. Elektrárna se může nacházet v pěti různých stavech:

- Nečinnost – v tomto stavu se elektrárna nachází, pokud dodávaný i požadovaný výkon jsou oba nula kilowattů.
- Startování – v tomto stavu se elektrárna nachází, pokud byla elektrárna v nečinnost, ale požadovaný výkon již není nula kilowattů a dodávaný výkon se ještě hladině požadovaného výkonu dostatečně nepřiblížil, tedy dodávaný výkon $<$ požadovaný výkon.
- Plný běh – v tomto stavu se elektrárna nachází, pokud jsou dodávaný i požadovaný výkon na stejné, nenulové, hodnotě (respektive jsou od sebe jen minimálně vzdáleny), tedy dodávaný výkon $=$ požadovaný výkon.

- Ukončování – v tomto stavu se elektrárna nachází, pokud elektrárna byla v plném běhu, ale požadovaný výkon klesl na hodnotu jeden kilowatt.
- Anomálie – v tomto stavu se elektrárna nachází, byl-li detekován velký pokles dodávaného výkonu či elektrárna byla ukončena nesprávným způsobem (viz kapitola anomálie).

Agent mezi stavy přepíná na základě konečného automatu zobrazeného na obrázku 5.3 a popsaného v části Implementace metody analyzuj() z třídy VykonAgent, která tento automat implementuje.

Dalšími úkoly tohoto agenta je zasílání zpráv agentovi typu Velký šéf o vzniklých anomáliích a rovněž zpráv informujících o zahájení (tedy přechod z nečinnosti, anomálie či ukončení na stav startování nebo plný běh) běhu elektrárny a o ukončení (tedy přechod ze stavu startování či plného běhu na stav anomálie nebo ukončování) běhu elektrárny. Implementační detaily agenta typu Výkon Agent lze nalézt v kapitole Agent třídy VykonAgent.

4.4 Struktura a formát meziagentní komunikace

V této podkapitole se budu věnovat popisu komunikace mezi samotnými agenty v multiagentním systému. Pro odesílání zpráv ve frameworku JADE se používají ACL zprávy (viz kapitola FIPA). Jako identifikátor agentů, kterým se má zpráva zaslat slouží třída AID. Zasílání zpráv funguje tak, že se zpráva nejprve uloží adresátovi do jeho fronty zpráv a adresát pak funkcemi pro přijmutí zprávy odebírá zprávy z této fronty. Zprávy si mezi sebou zasílají agenti sledující teplotu a Velký šéf, a pak agent sledující výkon a Velký šéf. Agent sledující výkon přímo s agenty sledující teplotu nekomunikuje, nicméně některé zprávy (konkrétně informace o startu a konci běhu elektrárny) mají na agenty sledující teplotu přímý dopad a jsou jim interpretovány skrz Velkého šéfa. Agenti sledující teplotu a agent sledující výkon nabízejí své služby ve žlutých stránkách (více v kapitole Žluté stránky), díky čemuž je agent Velký šéf schopen je najít a zahájit s nimi komunikaci. Agenti sledující teplotu a agent sledující výkon získají identifikátor Velkého šéfa (a tím pádem možnost s ním komunikovat) až z první zprávy, kterou jim zašle – což je ale ideální stav, Velký šéf je ten, kdo načítá a rozesílá data a ostatní agenti zasílají zprávy až v reakci na tato data. Vzhledem k tomu, že Velkému šéfovi je zasíláno mnoho typů zpráv o chybách od mnoha různých odesilatelů, je potřeba je nějak rozlišit. Proto jsem se rozhodl, že chybová zpráva bude mít formát „error{označení typu agenta}[jméno svíčky]zpráva“. Označení typu agenta může být buď VA, pro agenta sledujícího výkon, nebo TS pro agenta sledující teplotu svíčky. Jméno svíčky se vyskytuje pouze u zpráv od agentů sledující teplotu svíčky a informuje o tom, na které svíčce chyba vyvstala. Zatímco Velký šéf dostává nepravidelné zprávy, které reagují na nové situace v elektrárně, on sám agentům zasílá zprávy pravidelně, proto u jeho zpráv není důležitý formát zprávy, ale to, v jakém pořadí jaké zprávy posílá.

4.4.1 Žluté stránky ve frameworku JADE

Služba žlutých stránek ve frameworku JADE umožňuje agentovi publikovat jednu či více svých nabízených služeb, díky čemuž jej mohou ostatní agenti vyhledat a využít jeho služeb. Služba je inspirována klasickými telefonními žlutými stránkami, které rovněž fungují na podobném principu – volající hledá firmu/osobu poskytující služby, které potřebuje. Službu žlutých stránek ve frameworku JADE poskytuje Directory Facilitator (viz kapitola FIPA). [6]

Pokud chce agent začít nabízet svou službu ve žlutých stránkách, musí Directory Facilitatoru poskytnout popis své služby, který obsahuje agentův identifikátor (ve frameworku JADE je tento identifikátor třída AID), seznam svých služeb a případně seznam jazyků a ontologií, kterými je tento agent schopen komunikovat. Pro každou službu je publikován její typ a jméno a případně výčet jazyků a ontologií, které jsou potřebné k využití této služby. Pro publikování své služby musí agent využít metodu Directory Facilitatoru register(). [6]

Pokud chce agent vyhledat jiného agenta ve žlutých stránkách, musí Directory Facilitatoru dodat šablonu popisu agenta, ve které jsou vyplněné atributy, kterými hledaný agent musí disponovat (jako je například typ služby kterou poskytuje a tak podobně). Pro vyhledání musí agent použít metodu Directory Facilitatoru search(). Tato metoda vrací seznam popisů agentů, které odpovídají šabloně (tj. nalezený popis agenta disponuje veškerými atributy, které byly upřesněny v šabloně a má v nich stejnou hodnotu). [6]

4.4.2 Komunikace mezi agenty

Komunikace směrem Agent sledující výkon -> Velký šéf

- „start“ – Zpráva, informující Velkého šéfa o tom, že elektrárna začíná svůj běh. Po obdržení této zprávy Velký šéf začne zasílat agentům sledující teplotu svíček data.
- stop“ – Zpráva, informující Velkého šéfa o tom, že elektrárna ukončuje svůj běh. Po obdržení této zprávy Velký šéf v příštím cyklu zasílání dat zašle agentům sledujících teplotu hodnotu teploty 0. Pro ně je to signál, že běh skončil a mají vymazat uložený seznam hodnot daného běhu. Od dalšího cyklu pak Velký šéf nezasílá agentům sledujících teplotu zprávy, až do chvíle, kdy dostane zprávu, že běh znovu začíná.
- „errorVADoslo k velkemu vykyvu dodavaneho vykonu“ – Chybová zpráva, informující o tom, že došlo v elektrárně k anomálii, která se projevuje velkým poklesem dodávaného výkonu oproti požadovanému výkonu.
- „errorVADoslo k vypadku a nekorektnimu ukonceni“ – Chybová zpráva, informující o tom, že došlo v elektrárně k anomálii, která se projevuje nastavením požadovaného výkonu na 0, aniž by před tím byl nastaven na 1 – tedy že elektrárna byla ukončena, aniž by proběhl řádný proces ukončování (více v kapitole anomálie)

Komunikace směrem Velký šéf -> Agent sledující výkon

- První a následně každá lichá zpráva – Hodnota požadovaného výkonu elektrárny.
- Druhá a následně každá sudá zpráva – Hodnota dodávaného výkonu elektrárny.
- Speciální signály – V případě, že simulace končí, pak Velký šéf zašle agentovi sledujícího výkon hodnotu liché zprávy „-1“. Agent sledující výkon na to zareaguje tak, že se ukončí.

Komunikace směrem Agent sledující teplotu -> Velký šéf

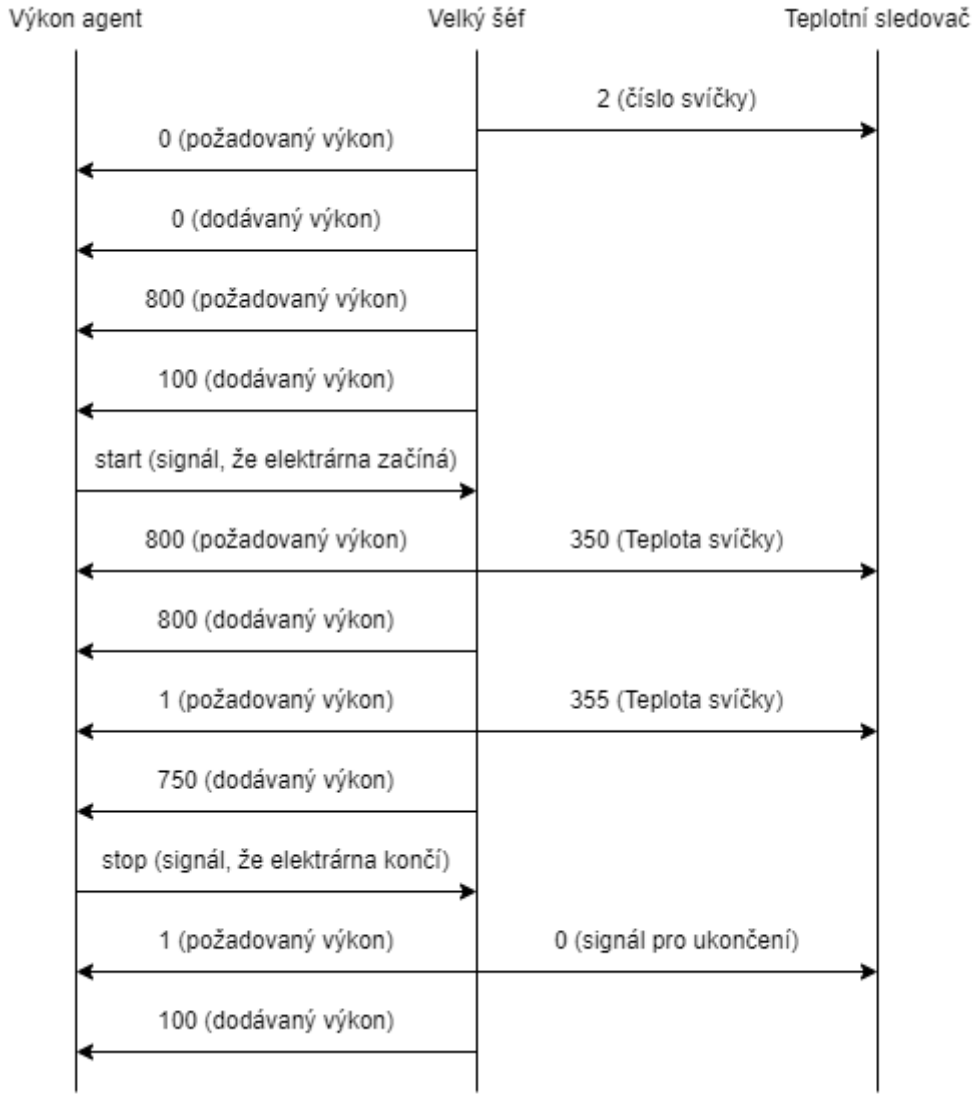
- „errorTS{označení svíčky} – Necekane vysoka teplota na svicce“ – Chybová zpráva, informující o tom, že elektrárna se patrně blíží do stavu anomálie, protože nová teplota svíčky je až příliš vysoká oproti průměru běhu.

- „errorTS{označení svíčky} – Necekane nizka teplota na svicce“ – Chybová zpráva, informující o tom, že elektrárna se patrně blíží do stavu anomálie, protože nová teplota svíčky je až příliš nízká oproti průměru běhu.

Komunikace směrem Velký šéf -> Agent sledující teplotu

- První zpráva – Číslo svíčky, kterou bude agent sledovat.
- Druhá a každá následující zpráva – Hodnota teploty na svíčce, kterou agent sleduje
- Speciální signály – V případě, že simulace končí, pak Velký šéf zašle všem agentům sledujícím teplotu na svíčkách zprávu s hodnotou -1. Agenti na to zareagují tak, že se ukončí. V případě, že končí jeden běh elektrárny, pak Velký šéf zašle všem agentům sledujícím teplotu na svíčkách zprávu s hodnotou 0. Agenti na to zareagují tak, že vymažou své seznamy hodnot teplot současného běhu a až jim Velký šéf zašle další hodnotu, tak tu již uloží do prázdného seznamu.

Příklad komunikace mezi agenty, je vyzobrazen na sekvenčním diagramu 4.1. Na tomto obrázku se multiagentní systém nejprve spustí a Velký šéf oznámí agentům monitorujících teploty svíček, číslo svíčky, kterou monitorují (v tomto konkrétním případě je to číslo 2, tedy svíčka A2). Elektrárna je nejprve ve stavu nečinnosti (požadovaný i dodávaný výkon jsou 0), následně startuje (požadovaný výkon vzrostl na 800 kW, dodávaný výkon mírně vzrostl, agent třídy VykonAgent posílá zprávu „start“ Velkému šéfovi), pak je v plném běhu (požadovaný i dodávaný výkon jsou 800 kW) a následně se ukončuje (požadovaný výkon klesl na 1, dodávaný výkon postupně klesá a agent třídy VykonAgent zasílá Velkému šéfovi zprávu „stop“. Velký šéf následně zašle signál 0 agentům monitorujícím teploty na svíčkách, čímž jim oznámí, že běh elektrárny skončil).



Obrázek 4.1: Ukázková komunikace mezi agenty při změně stavu elektrárny z nečinnosti na startování, následně na plný běh a následně na ukončování.

Kapitola 5

Implementace

5.1 Skript pro generování záznamů

Pro svůj skript pro generování JSON záznamů používám moduly csv (pro načítání a zpracování textového souboru s oddělovači), json (pro vytvoření souboru typu JSON), time (pro funkci čekání), os (pro práci se soubory) a sys (pro práci s parametry programu). V první řadě se načtou a zpracuji argumenty – argumenty jsou očekávány ve formátu Argument1:hodnota,argument2:hodnota. Podporované argumenty jsou Vstup, Zrychleni a Necekani (viz kapitola ovládání programu). Při spuštění bez argumentů jsou nastavitelné atributy nastaveny tak, že zrychlení je 1 (tedy že program běží stejně rychle jako reálný čas) (proměnná *speedy*), čekání je zapnuté (proměnná *waitingPeriodEnabled*) a vstupní soubor je in.csv (proměnná *csvFilePath*). Nezměnitelná parametry je složka, kam se budou generovat JSON soubory (jméno „imput“ uložené v proměnné *jsonDirPath*) a také jméno těchto souborů (jméno „out“ uložené v proměnné *jsonFilePath*). Skript nejprve pomocí funkce *os.path.exists()* zjistí, zdali složka input existuje a pokud ne, pak ji pomocí funkce *os.mkdir()* vytvoří. Následně se otevře vstupní soubor a pomocí funkce *csv.reader()* se načtou záznamy. Funkce jako oddělovač používá středník. Načtená data se uloží do seznamu (pomocí funkce *append()*), seznam se otočí pomocí funkce *reverse()* (protože data jsou zde uložena od nejnovějšího po nejstarší, tedy opačně, než se mají generovat) a pomocí funkce *pop()* se odstraní první záznam, který obsahuje legendu. Následně se data pomocí metody *json.dumps()* zapíší do souboru. Klíče pro prvky v JSONu jsou:

- *datum* pro datum a čas záznamu
- *power* pro dodávaný výkon
- *required* pro požadovaný výkon
- *A1-A8* a *B1-B8* pro teploty svíček

Doba intervalu mezi vznikem záznamů byla ve všech mých reálných datech 12 sekund, proto jsem tuto hodnotu nastavil jako základní čekací interval (proměnná *waitingPeriod*). Nicméně je možné že by došlo k nějaké změně, a proto se za běhu programu tento interval ještě jednou vypočítá ze samotných dat. Doba intervalu se pak následně podělí ještě hodnotou určující zrychlení (proměnná *speedy*). V případě, že není čekání vypnuté, pak se pro čekání v rozsahu tohoto intervalu používá funkce *time.sleep()*. Po projití všech záznamů se ještě vytvoří jeden JSON soubor, který obsahuje v atributu „datum“ hodnotu „end“.

5.2 Třída zaznam

zaznam je třída sloužící spíše jako struktura pro uchování dat jednoho záznamu.

5.2.1 Atributy

- int svicky[] – pole celých hodnot uchovávající teploty na jednotlivých svíčkách. Index 0 obsahuje teplotu svíčky A1, index 15 obsahuje teplotu svíčky B8. Typ int je dostatečný, protože reálná data uchovávají teplotu jen jako celé číslo.
- String datum – uchovává textový řetězec datumu a času, ze kterého je záznam
- int vykon – uchovává celočíselnou hodnotu momentálně dodávaného výkonu elektrárnou.
- int pozadovanyvykon – uchovává celočíselnou hodnotu požadovaného výkonu elektrárny. Podobně jako u teplot svíček jsou i hodnoty výkonů v reálných datech reprezentovány pouze celočíselně, takže typ int je dostačující.

5.2.2 Metody

- zaznam(JSONObject obj) – konstruktor, jehož parametrem je JSONObject, jehož formát jsem popsal v kapitole Skript pro generování záznamů (očekává klíče datum, power, required a A1 až A8 a B1 až B8). Pro získávání dat z objektu třídy JSONObject se používají metody *getString(klíč)* pro datum a *getInt(klíč)* pro hodnoty výkonu, požadovaného výkonu a teplot na svíčkách. V konstruktoru rovněž dojde k alokaci pole *int svicky* na pole o 16 prvcích.

5.3 Třída TeplotniSledovac

Třída TeplotniSledovac rozšiřuje třídu Agent a agent této třídy má za úkol sledovat a analyzovat teplotu na jedné svíčce.

5.3.1 Atributy

- long serialVersionUID – Třída jada.Agent je serializovatelná. Tento atribut je použit při serializaci/deserializaci objektu třídy, která je serializovatelná.
- int idSvicky – Číslo svíčky, na které agent pracuje, kde 0 znamená, že pracuje na svíčce A1 a 15, že na svíčce B8. Vzhledem k využití služby žlutých stránek je nutnost, aby se tato informace uchovávala tady, neboť Velký šéf (agent třídy *VelkySef*) nemá přehled který agent je na které svíčce.
- double prijatelnaOdchylka – Desetinná hodnota, obsahující procentuální hodnotu (tedy hodnotu v intervalu <0;1>). Pokud se nová hodnota teploty liší o větší než tento počet procent od průměru hodnot teploty současného běhu, pak agent detekuje nebezpečný výkyv teploty a informuje Velkého šéfa. Základní hodnota je 16 %, více o tomto v kapitole Pokusy.
- List<Integer> TeplotyBehu – Seznam hodnot teplot současného běhu elektrárny

- enum chyby – Výčet možných nestandardních stavů teploty svíčky, které agent detekuje a patřičně následně informuje Velkého šéfa. Jedná se o hodnoty *vysoka*, pro případ, že je nová teplota nezvykle vysoká a *nizka*, pro případ že je nová teplota příliš nízká.
- AID Boss – Tento atribut obsahuje identifikátor agenta, který Teplotnímu sledovači zasílá nová data a kterému Teplotní sledovač odesílá informace o nebezpečných výkyvech – tedy agent třídy VelkySef.
- class ZjisteniSvicky – třída popisující chování agentů (identifikaci na které svíčce agent bude pracovat), hlouběji popsaná v podkapitole Třída ZjisteniSvicky
- class TeploSledovani – třída popisující cyklické chování agentů (sledování a analýza teploty na jejich svíčce), hlouběji popsaná v podkapitole Třída TeploSledovani

5.3.2 Metody

- String identifikaceSvicky(int id) – Metoda pro převod čísla svíčky na její jméno. Vstupem je číslo svíčky. Funkce vrací textový řetězec se jménem svíčky.
- void vyresError(chyby chyba) – Metoda pro provedení adekvátní reakce v návaznosti na detekování nebezpečného výkyvu teploty. Vstupem je hodnota typu *enum chyba*, která označuje, o jaký druh chyby se jedná. Vevnitř funkce se pak na základě této chyby pomocí switche zvolí, jaká zpráva se má odeslat Velkému šéfovi. Konkrétní formát zpráv řeším v kapitole struktura a formát meziagentní komunikace.
- Double vypoctiPrumer(List <Integer> Teploty) – Metoda pro vypočítání průměrné teploty na této svíčce za současného běhu elektrárny. V případě, že současný běh elektrárny ještě nezačal, pak vrací funkce nulu. Parametrem funkce je celočíselný seznam teplot současného běhu elektrárny. Funkce vrací desetinné číslo s hodnotou průměru.
- void analyzuj (int teplota) – Metoda pro analýzu nové teploty, kterou agent právě obdržel. Vstupem je právě ona celočíselná hodnota nové teploty. Metoda nejprve přidá novou hodnotu do seznamu hodnot teplot současného běhu *TeplotyBehu*. Následně vypočítá průměrnou hodnotu tohoto seznamu pomocí metody *vypoctiPrumer()*. Následně zjistí, jestli je průměr ostře menší než součet nové teploty a násobku nové teploty a přijatelné odchylky – v takovém případě se jedná o nebezpečně nízkou teplotu, a obdobně pak zjistí, jestli je průměr ostře větší než rozdíl nové teploty a násobku nové teploty a přijatelné odchylky – v takovém případě se jedná o nebezpečně vysokou teplotu. Pokud odhalí nebezpečný výkyv zavolá metodu *vyresError()*. Algoritmus je vyzobrazen na obrázku 5.1.
- void takeDown() – Metoda, která se zavolá v případě, že se agent ukončuje. Agent se pokusí odhlásit svou službu ze žlutých stránek pomocí metody *DFService.deregister()*. Pokud neuspěje, vypíše chybové hlášení „Nepodařilo se agenta monitorujícího teplotu odebrat ze žlutých stránek“ a jméno agenta pomocí metody *getLocalName()*.
- void ukonceni() – Obalovací funkce, která zavolá metodu *doDelete()*, která je ze třídy agent a která daného agenta ukončí.

```
TeplotyBehu.add(teplota);
double prumer=vypoctiPrumer(TeplotyBehu);

if(prumer>Double.valueOf(teplota)+Double.valueOf(teplota)*prijatelnaOdchylka)
{
    vyresError(chyby.nizka);
}
else if(prumer<Double.valueOf(teplota)-Double.valueOf(teplota)*prijatelnaOdchylka)
{
    vyresError(chyby.vysoka);
}
```

Obrázek 5.1: Algoritmus analýzy nově načtené teploty

- void posliZpravu(String zprava) – Metoda pro odeslání zprávy agentovi třídy VelkySef. K identifikaci tohoto agenta využije atribut *Boss*. Pokud tento atribut má hodnotu rovnou null, pak k odeslání zprávy nedojde. Metoda vytvoří novou ACL zprávu s protokolem *ACLMessage.INFORM*. Pomocí metody *addReceiver()* nastaví odesilatele na hodnotu *Boss*, pomocí metody *setContent()* nastaví obsah zprávy na hodnotu *zprava* a zprávu odešle pomocí funkce *send()*.

- void setup() – Metoda, která se spustí při spuštění agenta. Nejprve dojde k analýze parametrů. Agent typu TeplotniSledovac očekává maximálně jeden argument, a to typu double, který bude obsahovat hodnotu přijatelné odchylky (viz kapitola Ovládání programu). Následně dojde k přidání agenta do žlutých stránek – do proměnné typu *DFAgentDescription* (záznam v Directory Facilitatoru viz kapitola FIPA) se pomocí metody *setName()* jako jméno nastaví hodnota identifikátoru tohoto agenta získaného metodou *getAID()* a pomocí metody *addServices()* popis služby agenta. Popis služby agenta se zapisuje do objektu třídy *ServiceDescription*. Jméno služby je pro celý projekt „JADE-elektrarny“ (nastaví se metodou *setName()*), typ služby jsem pro tohoto agenta zvolil „Sledovani-teploty“ (nastaví se metodou *setType()*). Registrace do žlutých stránek se dokončí metodou *DFService.register(agent, popis agenta)*. Algoritmus je vyzobrazen na obrázku 5.2 V případě že se nepodaří agenta přidat do žlutých stránek, vypíše se chybové hlášení „Nepodařilo se agenta monitorujícího teplotu přidat do žlutých stránek“ a jméno agenta pomocí metody *getLocalName()*. Úvodní nastavení agenta je zakončeno přidáním chování třídy ZjisteniSvicky pomocí třídy *addBehaviour()*.

5.3.3 Třída ZjisteniSvicky

Třída ZjisteniSvicky je třída popisu chování, která rozšiřuje třídu OneShotBehaviour (tedy se toto chování provede pouze jednou).

Atributy

- long serialVersionUID - Třída jade.OneShotBehaviour je serializovatelná. Tento atribut je použit při serializaci/deserializaci objektu třídy, která je serializovatelná.

```
try {
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(getAID());
    ServiceDescription sd = new ServiceDescription();
    sd.setName("JADE-elektrarna");
    sd.setType("Sledovani-teploty");
    dfd.addServices(sd);
    DFService.register(this, dfd);
}
catch (FIPAException fe) {
    System.out.println("Nepodarilo se agenta monitorujiciho teplotu pridat do zlutych stranek" + this.getLocalName());
}
```

Obrázek 5.2: Algoritmus registrace agenta ve žlutých stránkách

Metody

- void action() – Metoda popisující samotné chování. Cílem tohoto chování je dozvědět se od Velkého šéfa, jakou svíčku bude agent pozorovat. Proto pomocí funkce *blockingReceive()* čeká, než mu zašle zprávu, jejíž obsah by měl obsahovat číslo svíčky. Z této zprávy rovněž pomocí metody *getSender()* získá identifikátor Velkého šéfa a ten se uloží do atributu *Boss*. Po získání čísla svíčky a identifikátoru velkého šéfa se spustí nové cyklické chování *TeploSledovani*.

5.3.4 Třída TeploSledovani

Třída TeploSledovani je třída popisující chování agenta, která rozšiřuje třídu CyclicBehaviour (tedy se toto chování bude provádět pořád dokola).

Atributy

- long serialVersionUID - Třída jade.CyclicBehaviour je serializovatelná. Tento atribut je použit při serializaci/deserializaci objektu třídy, která je serializovatelná.

Metody

- void action() – Toto chování čeká na novou zprávou s novou teplotou (pomocí funkce *blockingReceive()*). Po získání zprávy nejprve určí, zdali se nejedná o signál k ukončení agenta (pokud je teplota menší než 0). V takovém případě zavolá metodu *ukonceni()*. Následně zkontroluje, zdali se nejedná o signál, který informuje, že současný běh končí – v takovém případě znovuinicializuje atribut *TeplotyBehu*, čímž ho vymaže. Pokud se nejedná o žádný ze signálů, spustí se funkce *analyzuj()*.

5.4 Třída VykonAgent

Třída VykonAgent rozšiřuje třídu Agent a implementuje agenta, který má za úkol sledování dodávaného a požadovaného výkonu a detekci vzniklých anomálií.

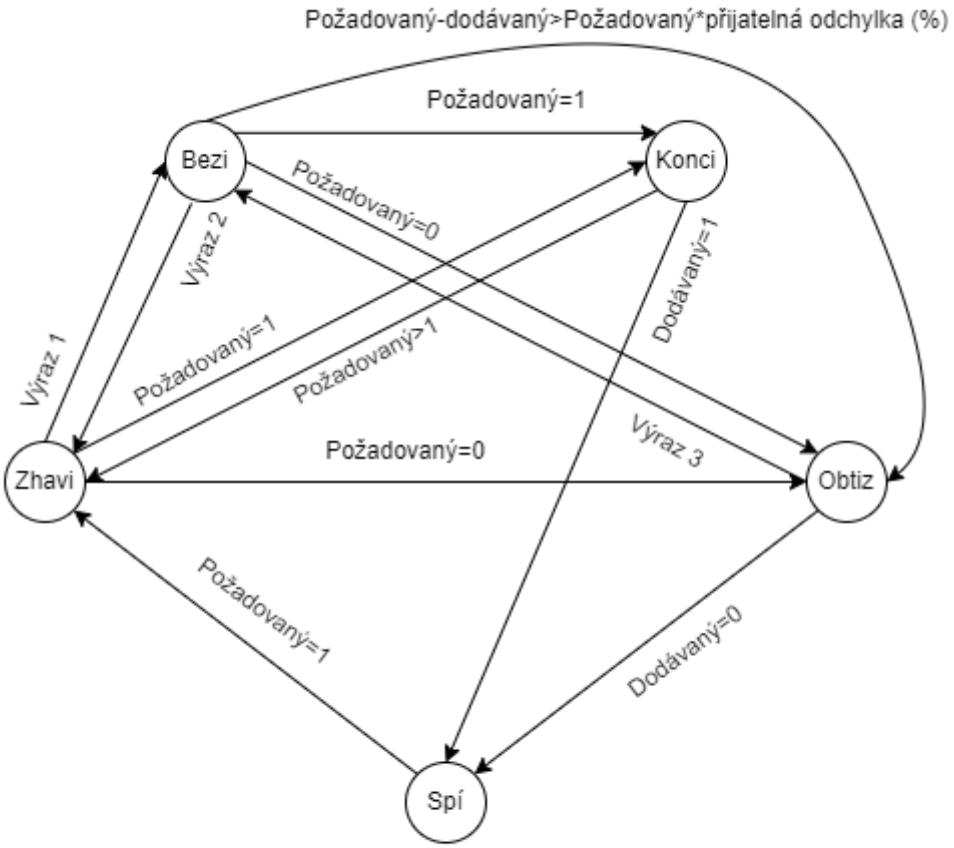
5.4.1 Atributy

- long serialVersionUID - viz třída TeplotniSledovac.

- enum chyby – výčet možných detekovaných anomálií. Obsahuje hodnoty *ukonceni*, pro anomálii nestandardního ukončení (požadovaný výkon padl rovnou na nulu, a ne před tím na jedničku) a *odchylka*, pro anomálii velkého dropu dodávaného výkonu.
- enum stav – výčet stavů ve kterém se elektrárna momentálně nachází. Obsahuje hodnoty *spi*, pro chvíli, kdy je elektrárna neaktivní, *zhavi*, pro chvíli, kdy elektrárna startuje, *bezi*, pro chvíli, kdy je elektrárna v plném provozu, *konci*, pro chvíli, kdy se elektrárna ukončuje a *obtiz*, pro chvíli, kdy byla detekována anomálie.
- int rozdilNazhaveni – Pokud se elektrárna zapíná a rozdíl mezi požadovaným a dodávaným výkonem je menší než tato hodnota, pak elektrárna přešla do stavu plného provozu. Nastaveno na hodnotu 10.
- double prijatelnaOdchylka – Procentuální hodnota (tedy hodnota z intervalu <0;1>) určující o kolik procent se již nemůže lišit hodnota dodávaného výkonu od požadovaného výkonu, než je detekována anomálie. Základní hodnota je 6 %.
- AID Boss – identifikátor agenta, který VykonAgentovi posílá nová data a kterému VykonAgent zasílá zprávy o anomálii a o stavu elektrárny – tedy agent třídy VelkySef
- stav Stav – uchovává současný stav elektrárny
- třída VykonSledovani – třída popisující cyklické chování agent, více v kapitole Třída VykonSledovani

5.4.2 Metody

- void vyresError(chyby Chyba) – Metoda pro zpracování informace o anomálii a zaslání patřičné zprávy Velkému šéfovi. Vstupem je hodnota z výčtu hodnot *chyby*. Na základě této hodnoty se pak pomocí switche zvolí patřičná zpráva a pomocí metody *posliZpavu()* se odešle.
- void analyzuj(int posledniPozadovany, int posledníDodavany) – Metoda pro analýzu nových dat. Vstupem je celočíselná hodnota nového požadovaného výkonu (*posledniPozadovany*) a celočíselná hodnota nového dodávaného výkonu (*posledniDodavany*). Metoda funguje jako konečný automat, jehož stavy jsou hodnoty z výčtu stavů elektrárny. Konečný automat je vyzobrazen na obrázku 5.3
  - Stav spi – Pokud je elektrárna ve stavu nečinnosti, zůstává v něm s výjimkou situace, kdy nový požadovaný výkon je nenulový. V takovém případě se změní stav elektrárny na stav startování elektrárny (tedy hodnota *zhavi*) a odešle se zpráva Velkému šéfovi s informací, že se elektrárna spouští. Ten tuto informaci předá Teplotním sledovačům, kteří od té chvíle analyzují teploty svíček.
  - Stav zhavi – Pokud je elektrárna ve startovací fázi, zůstává v ní, pokud nenastane nějaká z následujících situací:
    - Nový požadovaný výkon je 1 – to by znamenalo, že během startování došlo rovnou i k ukončení elektrárny. V takovém případě se přechází do stavu označující ukončování elektrárny (tedy hodnota *konci*) a Velkému šéfovi se zasílá zpráva s informací, že elektrárna končí. Ten tuto informaci předá Teplotním sledovačům, kteří v tu chvíli přestávají analyzovat teploty svíček.



Obrázek 5.3: Konečný automat popisující změny stavů elektrárny v návaznosti na nových hodnotách požadovaného a dodávaného výkonu

  - Nový požadovaný výkon je 0 – to by znamenalo, že během startování elektrárny došlo k anomálii (nestandardnímu ukončení) a elektrárna přejde do stavu anomálie (tedy hodnota *obtiz*). Zároveň se odešle zpráva velkému šéfovi, že se elektrárna ukončuje, který tuto informaci předá Teplotním sledovačům.
  - Rozdíl mezi novým požadovaným a dodávaným výkonem (případně jeho absolutní hodnota) je menší než atribut *rozdilNazhaveni*. V takovém případě se elektrárna již dostala do stavu plného provozu (hodnota *bezi*).
- Stav bezi – Pokud je elektrárna v plném provozu, zůstává v něm, pokud nenastane nějaká z následujících situací:
  - Nový požadovaný výkon je 0 – to by znamenalo, že během běhu elektrárny došlo k nestandardnímu ukončení. Elektrárna tedy přechází do stavu pro anomálii (hodnota *obtiz*) a zasílají se zprávy Velkému šéfovi, informující o anomálii a o ukončení běhu elektrárny.

        * Nový požadovaný výkon je 1 – V takovém případě se běh elektrárny ukončuje standardním způsobem. Elektrárna přechází do stavu ukončování (hodnota *konci*) a Velký šéf je o ukončování informován, což následně interpretuje teplotním sledovačům.
        * Rozdíl mezi požadovaným a dodávaným výkonem je větší nebo roven požadovanému výkonu vynásobeného atributem *prijatelnaOdchylka* – to by znamenalo, že došlo k anomálii způsobené velkým poklesem dodávaného výkonu. Elektrárna v takovém případě přechází do stavu pro anomálii. Velký šéf je informován o nastalé anomálii a je mu zaslána zpráva, že elektrárna končí.
        * Zvyší se požadovaný výkon. V takovém případě elektárna přechází do stavu žhavení (hodnota zhavi).
    - Stav konci – Pokud se elektárna ukončuje, zůstává v tomto stavu, pokud nenastane nějaká z těchto situací
        * Nový dodávaný výkon je 0 – V takovém případě bylo ukončení elektrárny úspěšné a elektrárna přešla do režimu nečinnosti (hodnota *spi*).
        * Nový požadovaný výkon je větší než 1 – V takovém případě se během ukončování elektrárny elektrárna znovu zapnula. Elektrárna tedy přechází do stavu zapínaní (hodnota *zhavi*) a Velký šéf je informován, že je elektrárna znovu spuštěna. Tato informace je následně předána Teplotním sledovačům.
    - Stav obtiz – Pokud je elektrárna ve stavu anomálie, zůstává v něm, kromě následujících případů
        * Nový dodávaný výkon je 0 – V takovém případě byla elektrárna kvůli anomálii ukončena a přechází do stavu nečinnosti (hodnota *spi*).
        * Rozdíl nového požadovaného a dodávaného výkonu je menší než násobek požadovaného výkonu a poloviny přijatelné odchylky. V takovém případě se zjevně podařilo anomálii opravit bez nutnosti ukončení elektrárny. Elektrárna tedy přechází do režimu plného běhu (stav *bezi*) a o této skutečnosti je informován Velký šéf a skrz něj následně i Teplotní sledovači.

- void ukonceni() - Obalovací funkce, která zavolá metodu *doDelete()*, která je ze třídy agent a která daného agenta ukončí.

- void posliZpravu(String zprava) – Metoda pro odeslání zprávy agentovi Velký šéf. Parametrem je textový řetězec obsahující onu zprávu. Pokud atribut *Boss* nemá hodnotu, pak se zpráva neodešle. Stejné, jako metoda *posliZpravu()* agenta třídy TeplotniSledovac.

- void takeDown() – Metoda, která se provede, když agent ukončuje svou činnost. Uvnitř metody agent odhlásí svou službu od žlutých stránek. V případě, že pokus o odebrání agenta ze žlutých stránek selže, vypíše se chybové hlášení „Nepodařilo se odebrat agenta monitoruji vykon ze žlutých stránek“.

- void setup() – Metoda, která se spustí při spuštění agenta. Nejprve se zpracují argumenty agenta – agent třídy VykonAgent očekává maximálně jeden argument, a to hodnotu přijatelné odchylky. Následně se agent přidá do žlutých stránek, což probíhá stejně jako u agenta třídy TeplotniSledovac, jen typ služby se u tohoto agenta jmenuje „Sledovani-vykonu“. Na závěr se spustí chování *VykonSledovani*. V případě, že se

ale nepodařilo agenta přidat do žlutých stránek, vypíše se chybové hlášení „Agenta monitorující výkon se nepodařilo přidat do žlutých stránek“ a agent se ukončí.

5.4.3 Třída VykonSledovani

Třída VykonSledovani je třída chování, která rozšiřuje třídu CyclicBehaviour (tedy toto chování provádí stále dokola)

Atributy

- long serialVersionUID – viz třída TeplotniSledovac.

Metody

- void action() – Agent vždy čeká na zprávu od Velkého šéfa (funkce *blockingReceive()*), která obsahuje hodnotu požadovaného výkonu z nejnovějšího záznamu. Pokud je tato hodnota menší než 0, pak se jedná o signál, že se má provést ukončení agenta. Následně (pokud se tedy neprovede ukončení) agent čeká na zprávu od Velkého šéfa, která obsahuje hodnotu nového dodávaného výkonu. Následně uloží do atributu *Boss* identifikátor Velkého šéfa a pomocí metody *analyzuj()* provede analýzu a případnou změnu stavu elektrárny na základě nového záznamu.

5.5 Třída VelkySef

Třída VelkySef rozšiřuje třídu agent a implementuje agenta, který má za úkol načítání dat, jejich předávání patřičným agentům, a naopak od agentů přijímat zprávy o anomáliích a stavu elektrárny. Ostatní agenty zároveň spouští a na závěr ukončuje celou agentní platformu, na které multiagentní systém běží.

5.5.1 Atributy

- long serialVersionUID – viz třída TeplotniSledovac.
- int id – Číslo JSON souboru, který se má právě načíst. JSON Soubory mají jména out{id}.json a v případě, že se id dostane nad číslo 10 000, pak se vynuluje.
- AID VA – Identifikátor agenta, který má za úkol sledování a analyzování dodávaného a požadovaného výkonu (tedy agenta třídy VykonAgent).
- AID[] TS – Pole identifikátorů agentů, kteří analyzují teplotu na svíčkách (tedy agentů třídy TeplotniSledovac).
- boolean bezi – Atribut uchovávající pravdivostní hodnotu, zdali elektárna zrovna běží, případně se startuje (hodnota true), nebo je v nečinnosti, případně se ukončuje nebo v ní zrovna dochází k anomálii (hodnota false).
- String datum – Textový řetězec obsahující datum právě zpracovávaného záznamu
- int delkaSpani – Hodnota určující kolikrát zrychleně simulace probíhá

```
File myObj = new File("input/out"+id+".json");
Scanner myReader = new Scanner(myObj);
while (myReader.hasNextLine()) {
   data += myReader.nextLine();
}

myReader.close();
String jsonString=data;
JSONObject obj=null;
try {obj=new JSONObject(jsonString);}
catch(Exception e) {return null;}
Zaznam = new zaznam(obj);
```

Obrázek 5.4: Algoritmus načtení nového JSON souboru

- double vykon – Procentuální hodnota (tedy hodnota z intervalu <0;1>), která určuje o kolik procent se může lišit dodávaný výkon od požadovaného, než dojde k detekci anomálie. Bez změny skrz argumenty programu je tato hodnota 6 %.
- double teplota – Procentuální hodnota (tedy hodnota z intervalu <0;1>), která určuje o kolik procent se může lišit nová teplota na svíčce od průměru teplot daného běhu, než dojde k detekci nebezpečné změny teploty na svíčce. Bez změny skrz argumenty programu je tato hodnota 16 %.
- Třída sefovani – třída popisující cyklické chování agenta. Více v podkapitole sefovani.

5.5.2 Metody

- zaznam nacteni() – Metoda pro načtení obsahu JSON záznamu. Funkce vrací objekt třídy zaznam s načteným novým záznamem, případně null, pokud se nepodařilo nový záznam načíst (například protože soubor, ze kterého se má načítat, neexistuje). Pokud soubor se jménem outid.json (kde *id* je atribut agenta) ve složce input existuje, otevře se pomocí třídy *File*. Následně se tento objekt použije jako parametr při vytvoření objektu třídy *Scanner*. Pomocí tohoto objektu a jeho metod *hasNextLine()* a *nextLine()* se postupně načte obsah souboru. Obsah souboru se použije jako parametr pro konstruktor objektu třídy *JSONObject*. Tento objekt se pak použije jako parametr pro konstruktor objektu třídy záznam. Algoritmus načtení ze souboru je vyzobrazen na obrazku 5.4. Následně se z nově vytvořeného záznamu získá nová hodnota atributu *datum*, odstraní se právě načtený soubor a inkrementuje se hodnota atributu *id*. Pokud hodnota atributu *id* přesáhla 10 000, pak je zresetována na 0. Na závěr se ještě zkontroluje, jestli nově načtené datum nemá hodnotu „end“, která by označovala, že skript pro generování json souborů již ukončil svou činnost a i multiagentní systém by se měl ukončit. V takovém případě by se zavolala metoda *ukonceni()*. Pokud tato situace nenastala, pak metoda vrátí nově načtený objekt třídy zaznam.
- void informujVykonAgenta(String zprava) – Metoda pro zaslání zprávy agentovi třídy VykonAgent. Parametrem je textový řetězec se zasílanou zprávou. Pokud je atribut *VA* bez hodnoty, pak se zpráva neodešle.

- void informujVykonAgenta(zaznam Zaznam) – Metoda pro zaslání nových dat agentovi třídy VykonAgent. Parametrem je objekt třídy zaznam, obsahující nová data. Pokud je atribut *VA* bez hodnoty, pak se zpráva neodešle. Ze záznamu se zasílají pouze atributy obsahující hodnoty požadovaného a dodávaného výkonu, a to jako dvě po sobě jdoucí zprávy (nejprve požadovaný a pak dodávaný výkon). Pokud by záznam obsahoval atributy, které vznikli chybou v datech (o tom více v kapitole Pokusy) a nabývají nevalidních, záporných, hodnot, pak se odešle místo toho odešle hodnota 0.

- void informujTeplotniSledovace (String zprava) – Metoda pro zaslání zprávy všem agentům třídy TeplotniSledovac. Parametrem je textový řetězec se zasílanou zprávou. Pokud je atribut *TS* bez hodnoty, pak se zpráva neodešle.

- void informujTeplotniSledovace(zaznam Zaznam) – Metoda pro zaslání nových dat všem agentům třídy TeplotniSledovac. Parametrem je objekt třídy záznam obsahující zasílaná data. Pokud je atribut *TS* bez hodnoty, pak se data neodešlou. Každému agentovi se odesílá jen jedna hodnota teploty, a to podle jeho indexu v poli *TS* (tj. Agentovi na indexu 0 se zašle teplota ze svíčky A1, agentovi na indexu 15 teplota ze svíčky B8). Pokud by záznam obsahoval atributy, které vznikly chybou v datech (o tom více v kapitole Pokusy) a nabývají nevalidních, tedy záporných či nulových, hodnot, pak se odešle místo toho odešle hodnota 1.

- boolean najdiVykonAgenta() – Metoda pro nalezení agenta poskytující služby sledování výkonu (tedy agenta třídy VykonAgent). Vrací pravdivostní hodnotu, podle toho, jestli agent byl (pak vrací true) nebo nebyl (pak vrací false) nalezen. Vytvoří se objekt třídy *ServiceDescription*, kterému se pomocí metody *setType()* nastaví hledaný typ služby na „Sledovani-vykonu“. Tento objekt se pomocí metody *addServices()* třídy *DFAgentDescription* vloží do objektu této třídy. Následně se pomocí metody *search()* třídy *DFService* vyhledají ve žlutých stránkách agenti, kteří poskytují tuto službu. Tato metoda vrací pole identifikátorů těchto agentů, nicméně agentovi stačí pouze jeden, tedy do atributu *VA* uloží identifikátor agenta na indexu 0 tohoto pole, a to pomocí metody *getName()*. Pokud však má vracené pole délku menší než 1, pak nebyl agent nalezen, hodnota atributu *VA* se nemění a metoda vrací hodnotu false.

- boolean najdiTeplotniAgenty() – Metoda pro nalezení agentů poskytující služby sledování teploty (tedy agentů třídy TeplotniSledovac). Vrací pravdivostní hodnotu, podle toho, jestli bylo nalezeno dostatek agentů (tedy šestnáct), pak vrací true, nebo nebylo, pak vrací false. Vyhledávání agentů ve žlutých stránkách probíhá podobně jako u metody *najdiVykonAgenta()*, jen tentokrát se jako typ služby použije „Sledovani-teploty“. Na rozdíl od jediného agenta pro sledování výkonu však multiagentní systém potřebuje šestnáct agentů pro sledování teploty. Proto, pokud nalezené pole agentů je menší než šestnáct, metoda vrací false. Pokud není menší než šestnáct, pak se všechny identifikátory agentů uloží do atributu pole *TS* pomocí metody *getName()* a metoda vrací hodnotu true.

- void vyresError(String zprava) – Metoda pro zpracování chybového hlášení. Parametr String zprava obsahuje chybovou zprávu zaslanou nějakým z agentů. Zpráva by měla začínat „error{dvouznaková identifikace typu agenta, který ji zaslal}“ a následovat samotným obsahem zprávy (viz kapitola Struktura a formát meziagentní komunikace). Metoda odstraní ze zprávy nápis error pomocí metody *replace()*, získá dvouznakovou

identifikaci typu agenta pomocí metody *substring()* (jen v případě, že má zpráva alespoň dva znaky) a na jejím základě rozhodne o jaký typ chybového hlášení se jedná a vypíše patřičné hlášení.

- void prijmiZpravu(ACLMessage msg) – Metoda pro zpracování přijaté zprávy. Parametrem je objekt třídy ACLMessage, obsahující přijatou zprávu. Pomocí metody *getContent()* se získá z přijaté zprávy její textový obsah. Pokud je obsah zprávy „start“, jedná se o signál od agenta pro sledování výkonu, že se elektrárna spustila. V takovém případě se nastaví atribut *bezi* na true. Pokud je obsah zprávy „stop“, jedná se o signál od agenta pro sledování výkonu, že se elektrárna ukončuje. V takovém případě se nastaví atribut *bezi* na false. Pokud zpráva obsahuje slovo „error“ jedná se o zprávu o vzniklé anomálii či nebezpečné situaci a je zpracována pomocí metody *vyresError()*.

- void ZpracujArgumenty(Object[] argumenty) – Metoda pro zpracování argumentů programu. Vstupem je pole objektů, kde každý objekt obsahuje jeden argument. Všechny argumenty jsou dobrovolné (o jejich formátu více v kapitole Ovládání programu). Každý objekt je nejprve převeden pomocí metody *toString()* na textový řetězec a následně se pomocí metod *equals()* a *startsWith()* vybírá, o který z podporovaných argumentů se jedná. Program podporuje argumenty

  - Necekani – v takovém případě se nastaví atribut *delkaSpani* (který zrychluje celou simulaci) na hodnotu 12 000. Tuto hodnotu jsem zvolil, protože agent mezi načítáním nových záznamů čeká 12 000 milisekund (více podkapitola Třída sefovani)
  - Zrychleni – z textového řetězce se získá odebráním podřetězce „Zrychleni:“ pomocí metody *replace()* požadovaná celočíselná hodnotu a atribut *delkaSpani* se nastaví na tuto hodnotu. Pokud je nová hodnota záporná, pak se rychlost simulace nastaví zpět na 1 a vypíše se chybové hlášení „Nevalidní hodnota zrychlení“.
  - Vykon – Metoda získá z textového řetězce požadovanou desetinou hodnotou odebráním podřetězce „Vykon:“ pomocí metody *replace()* a následným převedením tohoto textového řetězce na desetinné číslo pomocí funkce *Double.valueOf()*. Následně nastaví atribut *vykon* na tuto hodnotu. Pokud je nová hodnota tohoto atributu mimo interval <0;1> nastaví jej zpět na hodnotu 0.06 a vypíše chybové hlášení „Nevalidní hodnota výkonu“.
  - Teplota – Funguje velice podobně jako parametr Vykon, jen upravuje atribut *teplota*. Pokud je nová hodnota mimo interval <0;1>, nastaví jej zpět na hodnotu 0.16 a vypíše chybové hlášení „Nevalidní hodnota teploty“.

- void setup() – Metoda která se spustí při spuštění agenta. Agent nejprve zpracuje argumenty pomocí metody *ZpracujArgumenty()*. Následně musí spustit ostatní agenty, kteří budou monitorovat jednotlivé elementy elektrárny. To udělá tak, že nejprve funkcí *getContainerController()* získá kontejner agentů, ve kterém pracuje a v něm pak pomocí *createNewAgent()* spustí jednotlivé agenty. Metoda *createNewAgent()* očekává parametry jméno agenta, jméno třídy agenta a jeho spouštěcí argumenty ve formě pole třídy Object. Jméno agenta jsem v případě agenta monitorujícího výkon zvolil „powerWatcher“, v případě agentů monitorujících teplotu pak „temperatureWatcher0“ až „temperatureWatcher15“, Agentovi monitorujícího výkon ve spouš-

```
Object vykonArg[]=new Object[1];
Object teplotaArg[]=new Object[1];
vykonArg[0]=vykon;
teplotaArg[0]=teplota;
ContainerController cc= getContainerController();
try {
    AgentController ac = cc.createNewAgent("powerWatcher", "elektrarny.VykonAgent", vykonArg);
    ac.start();
    for(int i = 0; i<16;i++)
    {
        ac=cc.createNewAgent("temperatureWatcher"+String.valueOf(i), "elektrarny.TeplotniSledovac", teplotaArg);
        ac.start();
    }
```

Obrázek 5.5: Alogritmus spuštění všech agentů.

těcích argumentech metoda zasílá hodnotu argumentu *vykon* a agentům monitorujících teplotu pak hodnotu argumentu teplota. Metoda *createNewAgent()* vrací objekt třídy *AgentController*. Agent, kterého tento kontrolér reprezentuje, se spustí metodou *start()*. Algoritmus spouštění je zobrazený na obrázku 5.5. Po spuštění těchto agentů je třeba ještě najít jejich identifikátory pomocí metod *najdiVykonAgenta()* a *najdiTeplotniAgenty()*. Pokud by z nějakého důvodu nebyl nějaký agent nalezen (některá z metod by vrátila false), vypsalo by se chybové hlášení „Nenalezen agent pro sledování výkonu“, respektive „Nenalezen agent pro sledování teploty“ a multiagentní systém by se ukončil. Následně se ještě zašle Teplotním sledovačům číslo svíčky, kterou budou sledovat (číslo svíčky odpovídá jejich indexu v poli) a spustí se třída chování *sefovani*.

- void ukonceni() – Metoda pro ukončení agenta i celého multiagentního systému. Nejprve informuje ostatní agenty, že se mají ukončit. To udělá pomocí metody *informujTeplotniSledovace()*, respektive *informujVykonAgenta()*, kde jim zašle hodnotu -1. Následně se vytvoří a spustí nové vlákno programu pomocí třídy Thread a její metody *start()*. V původním vlákně se ukončí agent a v novém vlákně se pomocí funkce *getContainerController()* získá kontrolér kontejneru JADE agentů, následně z něj pomocí metody *getPlatformController()* získá platforma, na které multiagentní systém běží a ta se následně pomocí metody *kill()* ukončí. Tím se ukončí všichni agenti, platforma i její grafické rozhraní.

5.5.3 Třída sefovani

Třída sefovani rozšiřuje třídu CyclicBehaviour. Jedná se tedy o třídu popisující opakující se chování.

Atributy

- long serialVersionUID – viz třída TeplotniSledovac.

Metody

- void action() – Agent nejprve zkontroluje a případně přijme jemu zaslané ACL zprávy od ostatních agentů pomocí funkce *receive()* – zde bylo potřeba použít neblokovací přijímaní zpráv, protože ne vždy se mu musí některý z agentů ozvat.Rovněž bylo nutné počítat s variantou, že mu bylo zasláno více zpráv od více různých agentů,

proto načítá a zprácovává zprávy cyklicky než vyčerpá jejich frontu. Pokud mu nějaké zprávy přišly, zpracuje je pomocí metody *prijmiZpravu()*. Následně načte nový záznam pomocí metody *nacteni()*, zašle potřebná data agentovi monitorující výkon pomocí metody *informujVykonAgenta()*, a pokud elektrárna běží (což zjistí z atributu *bezi*), rovněž zašle nová data i agentům, kteří monitorují teploty na svíčkách, pomocí metody *informujTeplotniSledovace()*. Následně se uspí na délku 12 sekund pomocí funkce *TimeUnit.MILLISECONDS.sleep(12000)* – tuto hodnotu jsem zvolil protože s takovou rychlostí přibývají záznamy v datech, které jsem měl k dispozici. Délka spaní je ale ještě podělena hodnotou atributu *delkaSpani*, který určuje kolikrát je simulace zrychlena.

Kapitola 6

Pokusy

K analýze jsem měl k dispozici data z elektrárny od 29. srpna do 26. září 2021. Můj první pokus bylo spustit multiagentní systém na všechny dny a zaznamenat počet detekovaných anomálií s hranicí rozdílu požadovaného a dodávaného výkonu 3 %, 4 %, 5 % a 6 %. Zároveň pak zaznamenat, zdali v daném dni došlo k výpadku, v takovém případě pak ještě s jakým časovým předstihem byl detekován vznikající výpadek, a v poslední řadě, zdali byla detekována nějaká neobvyklá situace nějakým agentem monitorujícím teplotu na svíčce (příliš vysoká nebo nízká teplota na svíčce), jehož práh detekce byl nastaven na 15 %. Výsledky tohoto pokusu jsou zaneseny do tabulky 6.1.

Výsledkem tohoto testu bylo zjištění, že procentuální hranici mezi požadovaným a dodávaným rozdílem je vhodné nastavit na 6 % (10 falešných detekcí), případně 5 % (26 falešných detekcí), nebo vyšší. Cokoliv nižšího detekovalo až příliš mnoho falešně pozitivních anomálii (v případě prahu 3 % to bylo 201 falešných detekcí, v případě 4 % 75 falešných detekcí). Co se výpadků elektrárny týče, sedmého a dvacátého září došlo k výpadku elektrárny (sedmého rovnou dvakrát), který se projevil nejprve poklesem dodávaného výkonu oproti požadovanému a následně výpadkem. Časový předstih, s jakým můj multiagentní systém byl schopen předpovědět blížící se anomálii, jsem počítal jako rozdíl času, kdy byla poprvé detekována nestandardní teplota na svíčce (a tyto detekce se opakovaly až do času samotné anomálie) a času, kdy nastala samotná anomálie, kterou detekoval agent monitorující výkon, jehož práh detekce anomálie byl nastaven na 6 %. Osmého a desátého září došlo k situaci, kdy elektrárna za plného běhu bez předchozího výkyvu v dodávaném výkonu zničehonic klesla na hodnoty 0 kilowatt dodávaného a 0 kilowatt požadovaného výkonu. Detekce výkyvu teploty na svíčkách byla detekována přibližně v polovině případů, přestože výpadky tak často nenastaly, proto bylo nutné v detekci nastávajících anomálií provést úpravy.

Mnoho z chybných detekcí nastávajících anomálií bylo způsobeno jedním záznamem s velmi obskurními hodnotami, což byla zjevně chyba v naměřených datech. Při žhavení a v plném běhu elektrárny se běžně pohybuje v rozsahu 350 až 400 stupňů Celsia. Nicméně v některých dnech byl během běhu elektrárny naměřen jeden záznam, kde se teplota nečekaně vyhoupla na více než 25 000 stupňů Celsia. Tato chyba v datech bohužel způsobila, že ať bych nastavil procentuální hranici detekce vznikající anomálie (tedy o kolik procent se může nová teplota lišit od průměru) jakkoliv, tato abnormálně vysoká teplota by vždy průměr teplot posunula natolik nahoru, že by bylo detekováno mnoho příliš nízkých teplot. Tento problém jsem se rozhodl vyřešit mírnou úpravou architektury agenta třídy Teplotní sledovač. Doposud jsem vždy přidával novou hodnotu teploty do seznamu hodnot teplot, nově jsem se rozhodl, že hodnotu, která indikuje, že vzniká anomálie (tedy se liší od průměru hodnot více než kolik umožňuje procentuální hranice), do tohoto seznamu nepřidám.

Datum	6%	5%	4%	3%	Svíčky%	Výpadek	Předstih
29.8.	0	0	0	0	Ne	Ne	
30.8.	0	0	0	0	Ano	Ne	
31.8.	0	0	0	0	Ano	Ne	
1.9.	0	0	0	0	Ne	Ne	
2.9.	0	0	0	0	Ano	Ne	
3.9.	0	0	0	0	Ne	Ne	
4.9.	0	0	0	0	Ne	Ne	
5.9.	0	0	0	0	Ne	Ne	
6.9.	0	0	3	7	Ano	Ne	
7.9.	2	2	3	4	Ano	Ano (2)	2:00 a 2:12
8.9.	0	0	0	0	Ano	Ne	
9.9.	0	0	2	12	Ano	Ne	
10.9.	0	0	0	3	Ano	Ne	
11.9.	0	0	0	0	Ano	Ne	
12.9.	0	0	0	1	Ano	Ne	
13.9.	0	1	2	3	Ne	Ne	
14.9.	1	3	4	6	Ne	Ne	
15.9.	0	2	6	10	Ne	Ne	
16.9.	3	6	11	28	Ano	Ne	
17.9.	0	0	2	9	Ne	Ne	
18.9.	0	0	0	0	Ne	Ne	
19.9.	3	6	15	25	Ano	Ne	
20.9.	1	2	5	8	Ano	Ano	2:01
21.9.	0	0	1	3	Ne	Ne	
22.9.	0	0	1	9	Ne	Ne	
23.9.	2	2	8	27	Ano	Ne	
24.9.	1	5	15	49	Ano	Ne	
25.9.	0	0	0	0	Ne	Ne	
26.9.	0	0	0	0	Ne	Ne	
Počet chybných detekcí	10	26	75	201			

Tabulka 6.1: Výsledek detekcí anomálií mým multiagentním systémem v různých dnech, které jsem měl k dispozici. Sloupce 6,5,4 a 3 % označují jaká byl práh procentuálního rozdílu mezi požadovaným a dodávaným výkonem, který pokud byl překročen, byla detekována anomálie. Sloupec Svíčky označuje, zdali agent monitorující teploty na svíčkách detekoval nějakou vznikající anomálii. Sloupec Výpadek určuje, zdali daný den došlo k výpadku elektárny.

Cílem mého dalšího pokusu bylo nalezení optimální procentuální hodnoty, určující o kolik se může lišit nová hodnota teploty svíčky od průměru hodnot teplot svíčky, než by měl agent detekovat vznikající anomálii. Zde se ovšem falešné detekce dají rozdělit do dvou kategorií. Jednou z nich je jednorázová falešná detekce způsobená nevhodně zvoleným prahem detekce vznikající anomálie. Druhý je pak dlouhodobější odchylka teploty od průměru na jedné ze svíček. Zde se může jednat o skutečně vznikající anomálii, kterou se ale podařilo opravit bez nutnosti vypnutí elektrárny, nebo o statistickou odchylku vytvořenou chybným prahem. Má metodika rozdělování mezi těmito dvěma falešnými detekcemi byla taková, že

jsou-li falešné detekce na stejné svíčce v krátkém sledu minimálně 2 za sebou (reálně však bylo těchto detekcí v krátkém sledu vždy výrazně více), pak se jedná o druhou variantu, je-li jich méně, pak o první. Tento pokus mělo smysl provádět pouze na těch datech, kde agenti monitorující teploty na svíčkách v předchozím pokusu detekovali nějakou vznikající anomálii, neboť jsem chtěl práh detekce anomálie pouze zvýšit (či ponechat), abych zmenšil počet falešných detekcí. Do tabulky jsem nezapočítával chybný záznam obsahující silně abnormální hodnoty, o kterých se zmiňuji v odstavci výše.

Datum	Jednorázová	Dlouhodobá	Anomálie
30.8.	0	1	0
31.8.	0	1	0
6.9.	1	0	0
7.9.	2	1	2:00 a 2:12
8.9.	3	0	0
9.9.	1	0	0
11.9.	2	0	0
12.9.	1	0	0
16.9.	1	0	0
19.9.	0	1	0
20.9.	0	0	2:01
23.9.	1	2	0
24.9.	1	3	0

Tabulka 6.2: Výsledek detekcí vznikajících anomálií agentem monitorujícím teplotu svíčky s hranicí detekce 15 %. Sloupec Jednorázová označuje počet detekcí jednorázových falešných nálezů. Sloupec Dlouhodobá označuje počet detekcí dlouhodobých nálezů, které nevedly k výpadku. Sloupec Anomálie označuje časový rozdíl mezi detekcí vznikající anomálie a samotným vznikem anomálie.

Z testu vyšel nejlépe práh procentuální hranice detekce vznikající anomálie 16 % (tabulka 6.3). Tento práh nemá oproti původnímu prahu 15 % (tabulka 6.2) výrazné ztráty na času, ve kterém je schopný vznikající anomálie předvídat, a zároveň eliminuje značné množství falešných detekcí vznikajících anomálií. Práh 17 % (tabulka 6.4), přestože dále snižuje množství falešných detekcí vznikajících anomálií, bohužel již přináší až přílišnou časovou ztrátu s jakou je schopen agent předvídat anomálii. Stejná bude i situace u vyšších prahů. Proto se domnívám, že práh 16 % je ideální i za cenu několika falešných detekcí. Při použití systému v praxi by pak tedy bylo vhodné reagovat až v případě, kdy se vznikající anomálie v krátkém sledu (pravděpodobně ve dvou po sobě jdoucích záznamech) detekuje aspoň dvakrát. I v případě, že se při detekci s prahem 16 % počká na potvrzující záznam, je prodleva mezi předvídáním a vznikem anomálie lepší nebo stejná, jako kdyby se detekční práh zvýšil.

Datum	Jednorázová	Dlouhodobá	Anomálie
30.8.	0	0	0
31.8.	0	0	0
6.9.	1	0	0
7.9.	1	1	2:00 a 2:12
8.9.	2	0	0
9.9.	1	0	0
11.9.	1	0	0
12.9.	1	0	0
16.9.	0	0	0
19.9.	0	1	0
20.9.	0	0	2:01
23.9.	0	2	0
24.9.	0	3	0

Tabulka 6.3: Výsledek detekcí vznikajících anomálií agentem monitorujícím teplotu svíčky s hranicí detekce 16 %. Sloupec Jednorázová označuje počet detekcí jednorázových falešných nálezů. Sloupec Dlouhodobá označuje počet detekcí dlouhodobých nálezů, které nevedly k výpadku. Sloupec Anomálie označuje časový rozdíl mezi detekcí vznikající anomálie a samotným vznikem anomálie.

Datum	Jednorázová	Dlouhodobá	Anomálie
30.8.	0	0	0
31.8.	0	0	0
6.9.	1	0	0
7.9.	0	1	1:24 a 1:59
8.9.	1	0	0
9.9.	1	0	0
11.9.	1	0	0
12.9.	1	0	0
16.9.	0	0	0
19.9.	0	1	0
20.9.	0	0	1:48
23.9.	0	2	0
24.9.	0	3	0

Tabulka 6.4: Výsledek detekcí vznikajících anomálií agentem monitorujícím teplotu svíčky s hranicí detekce 17 %. Sloupec Jednorázová označuje počet detekcí jednorázových falešných nálezů. Sloupec Dlouhodobá označuje počet detekcí dlouhodobých nálezů, které nevedly k výpadku. Sloupec Anomálie označuje časový rozdíl mezi detekcí vznikuající anomálie a samotným vznikem anomálie.

Kapitola 7

Ovládání programu

7.1 Obsah odevzdaného archivu

Mé řešení se skládá z programu implementující multiagentní systém napsaného v jazyce Java za pomocí knihovny JADE pro tvorbu agentů sestávajících se ze souborů TeplotniSledovac.java, VelkySef.java, VykonAgent.java a zazanam.java, které jsou uložené ve složce src a skriptu pro generování nových záznamů napsaného v jazyce Python se jménem json-generator.py. Použité java knihovny (tedy jade.jar a json-20220320.jar) jsou ve složce lib. Vstupem pro skript jazyka Python je textový soubor s oddělovači (.csv), kde očekávaným oddělovačem je středník (jedná se o česká data, takže čárka v tomto souboru nefunguje jako oddělovač, ale jako desetinné znaménko). Ukázková vstupní data lze nalézt v souboru in.csv. Součástí archivu jsou ještě soubory build.xml pro překlad pomocí nástroje Apache ant a makefile pro spuštění pomocí nástroje GNU make na platformě Linux. Pro překlad a spuštění na platformě Windows je k dispozici soubor run.bat. Více o možnostech spuštění a překladu je v kapitole Ovládání programu.

7.2 Spuštění a překlad programu

Program lze spustit na platformách Windows a Linux (já ho testoval konkrétně na Windows 10 verze 10.0.19044 a Ubuntu 20.04). Pro správnou funkčnost je potřeba mít nainstalovanou Javu verze 17 a výše a Python 3. Pro překlad se používá kompilátor javac, který je dostupný v Java Development Kitu.

Ke spouštění a překladu na platformě Windows lze využít skript run.bat. Tento skript nejprve přeloží zdrojové soubory pro multiagentní systém a následně souběžně spustí Python skript pro generování dat a právě tento multiagentní systém. Pro správnou funkčnost je tedy potřeba mít nainstalovaný Python 3 a Java Development Kit verze 17 a výše a v systémové proměnné Path mít uložené adresáře vedoucí k těmto aplikacím. Spouštěcí argumenty programu je potřeba zadat jako run.bat "Parametr1:Hodnota,Parametr2:Hodnota,..."(například run.bat "Necekani,Vykon:0.10", více v podkapitole spouštěcí argumenty programu).

Na platformě Linux je seznam potřebných prerekvizit ke spuštění a překladu ještě rozšířen o nástroje GNU Make a Apache Ant. Překlad lze provést pomocí nástroje Apache Ant příkazem ant build, alternativně pomocí nástroje make příkazem make. Program po přeložení lze spustit pomocí nástroje make příkazem make run. Pro odstranění všech přeložených tříd a dočasných souborů, které vznikly za běhu programu (tj. JSON záznamy ve složce input), lze použít příkaz make clean. Pomocí příkazu make help je možné vypsat nápovědu

k programu. Pokud chcete spouštět program se spouštěcími argumenty, pak je potřeba program spouštět jako make run USER_OPTIONS= "Parametr1:Hodnota,Parametr2:Hodnota,... "(například make run "Necekani,Vykon:0.10 ").

7.3 Spouštěcí argumenty

Formát zadávání spouštěcích argumentů je bohužel trochu nestandardní a liší se při spouštění programu na Windows (v takovém případě formát run.bat "parametr1:hodnota,parametr2,... ") a Linux (v takovém případě make run USER_OPTIONS= "Parametr1:Hodnota,Parametr2,... "). Je to tak uděláno z toho důvodu, že agenti, vytvoření ve frameworku JADE, se spouští jako jméno:třída(argumenty), kde jednotlivé argumenty jsou oddělené čárkou a není očekáván žádný bílý znak, jako je například mezera. Podporované spouštěcí argumenty programu jsou:

- Vstup – Argument s hodnotou. Hodnota obsahuje jméno, případně cestu a jméno souboru se vstupními daty, tedy souboru, který má příponu .csv, jehož rozdělovači jsou čárky a jehož formát jsem popsal v kapitole formát dat. Pokud není argument zadán, pak je základní hodnota in.csv.
- Zrychleni – Argument s hodnotou. Hodnota určuje, kolikrát zrychlený je běh programu oproti reálnému času. Jeden vstupní soubor obsahuje data pro 24 hodin (respektive o trochu míň, protože první a poslední záznam nejsou přesně o půlnoci), tedy přibližně takto dlouhá je nezrychlená simulace. Pokud není argument zadán, pak je základní hodnota 1 (tedy simulace není zrychlená).
- Necekani – Argument bez hodnoty. Pokud je tento argument zadán, pak se program spustí s nejkratší možnou délkou simulace. Python skript bude generovat JSON záznamy „okamžitě“ bez jakéhokoliv čekání a multiagentní systém bude JSON záznamy načítat s nejkratší možnou prodlevou, což je 1 milisekunda.
- Vykon – Argument s hodnotou. Hodnota určuje procentuální hranici, určující, jak moc může dodávaný výkon poklesnout v porovnání s požadovaným výkonem, než je detekována anomálie. Tuto hodnotu je potřeba zapsat jako číslo z intervalu <0;1> a jako oddělovač mezi celočíselnou a desetinnou částí použít desetinou tečku (tedy například 0.10 pro hranici 10 %). Pokud není argument zadán, pak je základní hodnota 6 %.
- Teplota – Argument s hodnotou. Hodnota určuje procentuální hranici, určující, jak moc se může teplota na svíčce lišit v porovnání s průměrem během jednoho běhu elektrárny, než je detekována vznikající anomálie. Tuto hodnotu je potřeba zapsat jako číslo z intervalu <0;1> a jako oddělovač mezi celočíselnou a desetinnou částí použít desetinou tečku (tedy například 0.10 pro hranici 10 %). Pokud není argument zadán, pak je základní hodnota 16 %.
- Help – Argument bez hodnoty. Pokud je argument zadán, pak se vypíše nápověda a simulace se neprovádí.

Kapitola 8

Závěr

Domnívám se, že cíl práce, tedy vytvořit multiagentní systém, který bude detekovat a předvídat výpadky ve virtuální elektrárně, se podařilo naplnit. Byl vytvořen agentní systém složený z agentů ch teplotu svíček elektrárny, agenta monitorujícího výkon elektrárny a její celkový stav a agenta obstarávajícího vzájemnou komunikaci a přísun dat. Multiagentní systém je schopen anomálie detekovat i je předvídat s předstihem zhruba dvou minut. Pomocí pokusů jsem zjistil, že ideální práh procentuální hodnoty, určující jak moc může klesnout dodávaný výkon pod požadovaný výkon, je 6 %. Rovněž jsem zjistil, že ideální práh, určující jak moc se může lišit teplota od průměru hodnot teplot za jeden běh, než je detekována vznikající anomálie, je 16 %. Při testování s těmito prahy bylo detekováno několik falešně pozitivních nálezů, nicméně další zvýšení těchto prahů by vedlo ke zkrácení prodlevy mezi odhalením vznikající anomálie a samotným vznikem anomálie.

Do budoucna lze práci rozšířit tak, že by predikce anomálií vycházela z předvídání vývoje časových řad hodnot atributů elektrárny na základě jejich korelace s časovými řadami z minulosti. Nicméně k tomu je potřeba výrazně více dat, než jsem měl k dispozici.

Literatura

[1] *FIPA's mission statement*. Dostupné z: http://www.fipa.org/about/mission.html.

[2] *History of FIPA*. Dostupné z: http://www.fipa.org/subgroups/ROFS-SG-docs/History-of-FIPA.htm.

[3] *Write once, run anywhere?* 2002. Dostupné z: https://www.computerweekly.com/feature/Write-once-run-anywhere.

[4] *2021 Developer surveys*. 2021. Dostupné z: https://insights.stackoverflow.com/survey/2021.

[5] ASTRA. *ASTRA v1.0.0 Released*. Dostupné z: http://astralanguage.com/wordpress/astra-v1-0-0-released/.

[6] Bellifemine, F. L., Caire, G. a Greenwood, D. *Developing Multi-Agent Systems with JADE*. Wiley, 2007. ISBN 978-0-470-05840-4.

[7] Cardoso, R. C. a Ferrando, A. A Review of Agent-Based Programming for Multi-Agent Systems. *Computers*. 2021, sv. 10, č. 2. DOI: 10.3390/computers10020016. ISSN 2073-431X. Dostupné z: https://www.mdpi.com/2073-431X/10/2/16.

[8] Greenwood, D. *FIPA – The Foundation for Intelligent, Physical Agents*. Whitestein Technologies AG. Dostupné z: https://jade.tilab.com/papers/JADETutorialIEEE/JADETutorial_FIPA.pdf.

[9] Krbek, J. a Polesný, B. *Kogenerační jednotky - Zřizování a provoz*. 1. vyd. GAS s.r.o., Praha, 2007. ISBN 978-80-7328-151-9.

[10] Leary, S. *Org.json*. Dostupné z: https://github.com/stleary/JSON-java.

[11] Neruda, R. a Pilát, M. *MULTIAGENTNÍ SYSTÉMY NAIL 106*. MFF UK Praha, 2014. Dostupné z: http://ktiml.mff.cuni.cz/~neruda/mas-14.pdf.

[12] Russell, S. a Norvig, P. *Artificial Intelligence: A Modern Approach*. 4. vyd. Prentice Hall, 2020. ISBN 9780136042594.

[13] Venners, B. The Making of Python – A Conversation with Guido van Rossum, Part I. *Artima*. 2003. Dostupné z: https://www.artima.com/articles/the-making-of-python.

[14] Zbořil, F. *Agentní a multiagentní systémy*. FIT VUT, 2006.