VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KOALIČNÍ HRY V DYNAMICKÉM MULTIAGENTNÍM PROSTŘEDÍ

COALITION GAMES IN A DYNAMIC MULTIAGENT ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETER HAMRAN
AUTHOR

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

BRNO 2019

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

21985

Student: **Hamran Peter**
Program: Informační technologie
Název: **Koaliční hry v dynamickém multiagentním prostředí**
Coalition Games in a Dynamic Multiagent Environment
Kategorie: Umělá inteligence
Zadání:

1. Seznamte se s pravidly multiagentní soutěže MASSIM a identifikujte problémy, které vyžadují koaliční přístup k řešení.
2. Nastudujte současné algoritmy pro vytváření multiagentních koalic s ohledem na časová omezení ve výše uvedených herních prostředích.
3. Zvolený přístup k formování koalic vhodně upravte vzhledem k pravidlům soutěže a implementujte jej.
4. Vytvořte rozhraní, které umožní použití vzniklého systému v multiagentním prostředí JADE.
5. Porovnejte schopnost agentů vytvářet koalice a dosahovat zvolených cílů. Studujte také výkonnost celého multiagentního týmu vzhledem k náchylnosti k přehodnocování koalic při změně možných cílů v systému a diskutujte dosažené výsledky.

Literatura:

- Wooldridge, M.: An Introduction to MultiAgent Systems, 2nd Edition, Willey, 2009
- Shoham, Y., Leyton-Brown, K.: Multiagent Systems, Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2009

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

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

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Podstatou tejto bakalárskej práce bolo implementovať algoritmus na zostavenie schopných koalícií v multiagentnom systéme. Pod pojmom schopných koalícií rozumieme koalície, ktoré nie sú v superaditívnom prostredí, čiže záleží na ich veľkosti a kompetentnosti členov. Cieľom práce je upraviť algoritmus budovania koaličných väzieb medzi agentami tak, aby bol aplikovateľný pri multiagentnej súťaži MASSIM. Kľúčovými aspektami sú množstvo správ zaslaných medzi agentmi na ustanovenie koalície a rýchlosť výpočtu koaličných hodnôt. Cieľovým scenárom je súperenie dvoch multiagentných systémov o zdroje.

Abstract

The essence of this bachelor thesis was to implement an algorithm for suitable coalition forming in multi-agent system. Suitable coalition denotes a coalition which is not formed in super-additive environment and so it depends on its size and members. Aim of this thesis was to adjust an algorithm for coalition bounds forming between agents in a way, that it is applicable in a multi-agent competition MASSIM. Key aspects are amount of messages sent between agents for coalition to form and also a time it takes to calculate all coalitional values required for such a process. Final scenario is where two competitive groups of agents compete for resources in a multi-agent system simulation.

Kľúčové slová

multiagentný systém, agent, fipa, jade, massim, koaličné hry, koalície, shehory-kraus

Keywords

multi-agent systems, agent, fipa, jade, massim, coalition games, coalitions, shehory-kraus

Citácia

HAMRAN, Peter. *Koaliční hry v dynamickém multiagentním prostředí.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Koaliční hry v dynamickém multiagentním prostředí

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána doc. Ing. Františka Zbořila, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

. . . . . .
Peter Hamran
15. mája 2019

Poďakovanie

V prvom rade by som chcel poďakovať vedúcemu mojej práce pánovi doc. Ing. Františkovi Zbořilovi, Ph.D. za jeho odbornú pomoc a vedenie pri písaní tejto bakalárskej práce. Taktiež by som chcel poďakovať svojej matke Jane Hamranovej za pomoc pri syntaktickosémantickej analýze tejto práce.

Obsah

1 Úvod **2**

2 Multiagentné systémy **3**
2.1 Agent 3
2.2 Prostredie 4
2.3 Typy agentov 5
2.4 Vnímanie 7
2.5 Organizácia agentov 7
2.6 Komunikácia medzi agentmi 9
2.7 Distribuované riešenie problémov 9

3 Vývoj agentov **11**
3.1 Špecifikácia FIPA 11

4 Nástroj JADE **13**
4.1 Služba prenosu správ 13
4.2 Systém riadenia agentov 15
4.3 Sprostredkovateľ adresárov 15

5 Multi-Agent Systems Simulation Platform (MASSim) **17**
5.1 Scenár simulácie 17

6 Koaličné hry **18**
6.1 Prideľovanie úloh 18
6.2 Algoritmus Shehory-Kraus 18

7 Implementácia **22**
7.1 Pomocné metódy 22
7.2 Implementácia agenta 23

8 Výsledky **29**
8.1 Prehľadávanie stavového priestoru 29
8.2 Dopad veľkosti koalícií na množstvo správ 30

9 Záver **32**

Literatúra **33**

Kapitola 1

Úvod

V súčastnej dobe sa simulačné a výpočetné úlohy stávajú čoraz ťažšími. Preto spracovanie týchto úloh klasickými centralizovanými metódami sa stáva náročnejším a pracnejším. Multiagentné systémy so svojimi vlastnosťami poskytujú praktické riešenie týchto problémov. Poskytovaním distribuovaného rozhodovania a konceptov umelej inteligencie odbremeňujú samotného programátora od statického určovania cieľov. Zatiaľ čo ho prenechávajú na agentoch systému, ktorí sa dajú opísať ako počítačové systémy schopné vnemu prostredia a autonómneho rozhodovania o svojích činoch pre účel dosiahnutia zvoleného cieľa.

Aplikáciu multiagentných systémov môžeme vidieť napríklad pri simuláciách trhu, monitorovaní alebo diagnostike systému. Jednou zo známych implementácií tohto typu je systém na monitorovanie fizyckej bezpečnosti elektrických rozvodní. Tento systém využíva agentov s adaptívnymi kritériami rozhodovania na zabránenie katastrofických následkov medzi prepojenými sieťami. Pri simuláciách trhu sú agenti pomocou Q-learning-u schopní odhaliť a využiť slabiny trhu, pričom im to prinesie zvýšený zisk.

Podstatou tejto práce je vytvoriť modul, ktorý pracuje s princípmi návrhu multiagentných systémov a docieli vytvorenie koalícií medzi agentmi. Každý agent je schopnou entitou prostredia, ktorá vníma svoje okolie a vykonáva činnosti, ktoré toto prostredie ovplyvňujú.

Multiagentný systém, s ktorým pracujeme v tejto práci je konkurenčná simulácia, kde proti sebe zápasia dva tímy, umiestnené v pomyselnom meste, kde súperia o zdroje. Keďže na mape mesta sa v každej simulácií objavujú studne, ktoré agenti musia obsadiť a udržovať. Taktiež v simulácií vznikajú ponuky na prácu, ktoré sú splniteľné len spoluprácov viacerých agentov.

Význam slova koalícia je označenie skupiny, v našom prípade skupiny agentov, ktorá obsahuje dvoch alebo viac agentov. Títo agenti vytvárajú dočasné partnerstvo na dosiahnutie svojich cieľov. Vytváranie koalícií je preto nevyhnutné pre agentov na dosiahnutie cieľov, ktoré by samostatne neboli schopní splniť. Prípadne na zvýšenie efektivity plnenia daných úloh.

Na začiatku tejto práce je popísaná všeobecná teória o multiagentných systémoch a ich agentoch. Typy prostredia, v ktorom sa nachádzajú a princíp vnímania tohto okolia. Ďalšou kapitolou nasleduje špecifikácia autonómnych a inteligentných agentov, z ktorej prechádzame k nástroju JADE, ktorý túto špecifikáciu implementuje a poskytuje nástroje pre vývoj multiagentných systémov v jednotlivých platformách, ktoré sú závislé od zariadenia, na ktorom bežia. V kapitole 5 je popis simulačného servera a súťaže. Nasleduje popis implementovaného algoritmu, ktorý sme použili na vytváranie koalícií a implementácia projektu. Poslednou kapitolou uzatvárame celok popisom nárokov, ktorý má tento algoritmus na systém.

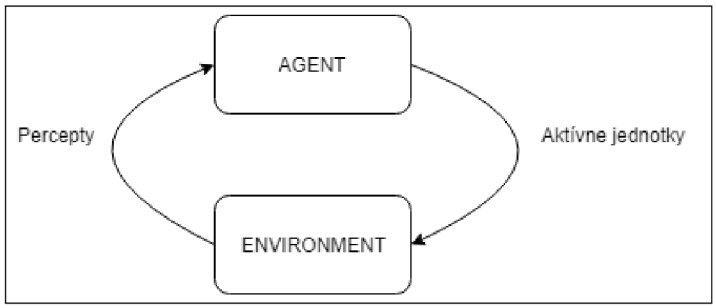
Kapitola 2

Multiagentné systémy

Multiagentný systém je kolekcia výpočtových elementov nazývajúcich sa taktiež agenti. Títo agenti spolupracujú alebo navzájom súperia v dosahovaní osobných alebo všeobecných cieľov.

2.1 Agent

Základom každého multiagentného systému je agent a prostredie, v ktorom sa nachádza. Pre účely tejto práce si môžeme pojem agent definovať ako čokoľvek, čo dokáže vnímať svoje okolie pomocou senzorov a taktiež ovplyvňovať toto svoje prostredie pomocou aktívnych jednotiek, ktoré sú mu k dispozícii.



Obr. 2.1: Interakcia agenta s prostredím, v ktorom sa nachádza.

V tejto práci sa budeme baviť práve o inteligentných agentoch keďže nie každý agent vyskytujúci sa v našich aplikáciach alebo okolo nás v podobe termostatov alebo Unixových démonov je inteligentný. Síce už vieme čo je agent, ale aké vlastnosti ho robia inteligentným? Nasledujúce vlastnosti boli zmienené v [13]

- **Autonómnosť** - Agenti konajú bez ľudského alebo iného priameho zásahu a majú kontrolu nad vlastnými činmi a svojím vnútorným stavom.
- **Schopnosť socializácie** - Agenti komunikujú s ostatnými agentmi (a možno aj s ľuďmi) pomocou nejakých, vopred určených, komunikačných jazykov.
- **Reaktivita** - Agenti vnímajú svoje prostredie a reagujú v rozumnom čase na výskyt zmien v tomto prostredí.

- **Proaktivita** - Agenti nielen že reagujú odpoveďou na zmenu prostredia, ale sú schopní prebrať iniciatívu a tým preukázať správanie zamerané na určitý cieľ.

Tieto vlastnosti nám pomáhajú budovať našich agentov, ale taktiež kladú nároky na ich implementáciu. Zabezpečiť cieľovo orientované správanie systému nie je nič atypické. Takýto prístup volíme vždy, keď píšeme napríklad funkciu v jazyku C alebo metódu v Pythone. V tomto prípade túto funkcionalitu popisujeme v podobe podmienok, ktoré musia byť splnené pred vykonávaním funkcie a efektu, ktorý nastane po jej vykonaní. Spomínaný efekt sa dá považovať za cieľ alebo zámer autora danej funkcie alebo metódy. Tento funkcionálny prístup funguje, pokiaľ môžeme predpokladať, že sa prostredie, v ktorom sa daná funkcia vykonáva nezmení počas jej vykonávania. V prípade, že prostredie agentovi neumožní vznik týchto predpokladov, agent musí byť schopný upravovať svoje ciele reaktívne vzhľadom na zmeny prostredia. Keďže slepé vykonávanie funkcií by neprinieslo žiaden benefit k jeho aktuálnemu stavu. Takýmito systémami sú práve multiagentné systémy. Čiže systémy, kde sa prostredie mení vplyvom viacerých agentov alebo iného vonkajšieho vplyvu.

2.2 Prostredie

Jedným z kľúčových problémov, s ktorým sa musí agent vysporiadať je, ktorú akciu zo svojho repertoára má vykonať za účelom uspokojenia svojich predurčených cieľov. Agentné architektúry, ku jednej z nich sa dostaneme v sekcii 3, sú softvérové architektúry, ktoré zabezpečujú rozhodovanie a sú spojené s prostredím. Náročnosť rozhodovacieho procesu môže byť ovplyvnená množstvom rozličných vlastností prostredia. Russel a Norvig [10] navrhujú nasledovnú klasifikáciu vlastností prostredia.

- **Plne pozorovateľné vs. čiastočne pozorovateľné** - Pozorovateľnosť prostredia je možné rozhodnúť na základe senzorov, ktorými agent disponuje. V prípade, že jeho senzory poskytujú všetky informácie o prostredí, v ktorom sa nachádza, je prostredie plne pozorovateľné a agent si nemusí uchovávať žiadne informácie o svojom okolí. Prostredie môže byť čiastočne pozorovateľné napríklad z dôvodu nedostatočných schopností senzorov (hlavne v reálnom prostredí). Alebo z dôvodu zatajovania informácií o prostredí.

- **Jednoagentné vs. multiagentné** - Rozdiel medzi jednoagentným prostredím a multiagentným prostredím môže byť na prvý pohľad zrejmý. V prípade agenta riešiaceho krížovky sa jasne jedná o jednoagentné prostredie a zase pri agentovi hrajúceho šach ide o multiagentné prostredie. Na začiatku sekcie 2.1 sme si povedali, aká entita môže byť agentom. Avšak zatiaľ nevieme, aké entity musia byť agentom v prostredí. Kľúčovou vlastnosťou pre túto špecifikáciu je, či sa entita snaží maximalizovať opatrenia, ktorých hodnoty záležia na správaní iných entít. Napríklad pri šachu, čo je kompetitívne multiagentné prostredie, sa snaží oponent maximalizovať svoje opatrenia pre zaručenie výhry. Čo zapríčiní maximalizáciu opatrení oboch účastníkov, pretože náš agent zareaguje na toto správanie maximalizáciou vlastných opatrení.

- **Deterministické vs. stochaistické** - V prípade, že ďalší stav prostredia je úplne podmienený terajším stavom a akciou vykonanou agentom hovoríme, že prostredie je deterministické. Prakticky v tomto prípade sa agent nemusí zaoberať neistotou v prípade plne sledovateľného deterministického prostredia. No v prípade, že prostredie nie je plne sledovateľné, automaticky sa stáva stochaistickým, pretože je nemožné predpokladať všetky možné nasledujúce stavy prostredia.

- **Epizodické vs. sekvenčné** - Epizodické prostredie sú pocity agenta rozdelené do jednotlivých epizód. V každej z týchto epizód agent dostane percepty a následne vykoná jedinú akciu. Čo je avšak dôležité, jednotlivé epizódy sú od seba nezávislé, čiže akcie vykonané v aktuálnej epizóde nemajú žiaden vplyv na epizódu nadchádzajúcu. Naopak to platí v sekvenčnom prostredí, kde akcie momentálne vykonané môžu mať vplyv na všetky nadchádajúce rozhodnutia agenta.

- **Statické vs. dynamické** - Pre agenta je jednoduché rozhodovať sa v statickom prostredí, pretože počas rozhodovania nemusí brať do úvahy možné zmeny prostredia alebo plynutie času. Dynamické prostredie sa však agenta neustále pýta na akciu, ktorú chce vykonávať. Ak agent nie je v tom momente ešte rozhodnutý pre činnosť, ktorú chce vykonať, prostredie to vníma ako rozhodnutie nerobiť nič. Jednoducho to znamená, pokiaľ sa prostredie môže meniť počas agentovho rozhodovania hovoríme, že je dynamické. V opačnom prípade je statické.

- **Diskrétne vs. spojité** - Toto rozdelenie špecifikuje hlavne stav prostredia vzhľadom k spôsobu správy času, perceptov a akciám agentov.

2.3 Typy agentov

Ako sme si definovali v predchádzajúcom texte 2.1, agent príjíma vnemy z prostredia a následne reaguje pomocou aktívnych jednotiek. Agent väčšinou príjíma len aktuálny vnem prostredia, ktorý odzrkadľuje len aktuálny stav jeho okolitého prostredia, pretože žiadne iné informácie nie sú od prostredia dostupné. Na základe týchto informácií agent vyhodnocuje svoje rozhodnutia. Agenti sa na základe príjímania a spracovania vnemov dajú implementovať rôznymi spôsobmi. V knihe Artificial Intelligence: A Modern Approach navrhli delenie implementácie agentov podľa spracovania vnemov nasledovne [10]. Rôzne systémy využívajú špecifické komponenty z týchto implementácií k individuálnym účelom.

Jednoduchý reflexný agent

Tento typ agenta je najjednoduchší zo všetkých spomenutých. Pracuje a rozhoduje sa len na základe najaktuálnejších vnemov a úplne ignoruje všetky doposiaľ nadobudnuté informácie pri rozhodovacom procese. Správania podobného typu sa vyskytujú aj v komplexných prostrediach z dôvodu možnej okamžitej reakcie na situáciu, ktorá nastala.

Typ jednoduchého reflexného agenta je avšak, na druhú stranu limitovaný svojou inteligenciou. Pretože jeho logika bude fungovať len v prípade, že sa dá spraviť správne rozhodnutie na základe všetkých jeho vnemov. Inými slovami povedané, tento typ agenta vyžaduje, aby bolo operačné prostredie plne pozorovateľné. Čo znamná, že je nevyhnutné, aby bolo všetko potrebné pre rozhodovanie obsiahnuté v jednom vneme, napríklad v jednom obrázku z videa. Dá sa teda predpokladať, že agent reaguje na situáciu reflexívne, bez zvažovania. Pozorovateľnosť prostredia sme riešili v sekcii 2.2.

Reflexný agent založený na modele

Ako sme riešili v predchádzajúcom odseku, je pre agentov náročné sa efektívne rozhodovať v prostredí, ktoré nie je plne pozorovateľné. Najefektívnejší spôsob, ako zvládať túto neúplnú pozorovateľnosť prostredia je, aby si agent uchovával časť prostredia, ktorú aktuálne nemôže vidieť. To znamená, že si agent udržuje nejakým spôsobom vnútorný stav, ktorý odzrkadľuje

históriu vnemov a tým pádom vlastne históriu zmien svojho prostredia. Táto história môže zahrňovať aj aktuálne nepozorovateľné aspekty prostredia.

Ďalšie dva typy znalostí sú nevyhnutné pre udržanie vnútorného stavu agenta v aktuálnej podobe. V prvom rade potrebujeme spracovávať informácie týkajúce sa a odzrkadľujúce evolúciu okolitého sveta nezávisle od pozorujúceho agenta. Ale taktiež informácie o spôsobe, akým akcie agenta ovplyvňujú okolité prostredie. Tieto znalosti o spôsobe vplyvu okolitého prostredia na agenta, a teda aj vplyv agenta na okolité prostredia sa nazývajú model prostredia.

V niektorých prípadoch agent nevie s presnosťou určiť, ako svet okolo neho vyzerá v prípade, že nie je v plne pozorovateľnom prostredí. Vtedy sa snaží čo najlepšie odhadnúť stav okolia. Napríklad autonómne auto pri obiehaní veľkej prekážky, cez ktorú nevidí musí odhadnúť stav prostredia na druhej strane. Takže neúplnosti informácie sa v istých prípadoch nedá vyhnúť, agent aj napriek tejto neúplnosti musí rozhodnúť, aké budú jeho nasledujúce akcie.

Agent založný na cieľoch

Ako sme už do istej miery naznačili, iba znalosť o aktuálnom stave prostredia, v ktorom sa agent nachádza nemusí byť dostatočná na rozhodnutie o akcii. V niektorých situáciách agent okrem tohto aktuálneho stavu prostredia potrebuje aj znalosť o jeho cieli, aby sa bol schopný rozhodnúť najlepšie ako môže. Napríklad pri aute zaseknutom v dopravnej zápche, len znalosť prostredia nestačí na rozhodnutie, či má zabočiť vľavo, vpravo alebo ísť rovno. Tieto rozhodnutia záležia od skutočnosti, kam sa chce vodič dostať.

V niektorých situáciach je rozhodovanie sa na základe cieľov jednoduché. Napríklad, keď uspokojenie cieľa zapríčiní vykonanie jedinej akcie. Avšak inokedy je tento proces náročnejší. Napríklad, keď agent musí zvažovať dlhodobé prenasledovanie určitého cieľa bez reálnych výsledkov.

Môže sa zdať, že tento typ agenta je menej efektívny, jeho výhodou je však flexibilita, ktorá vychádza z jednoduchej úpravy explicitných vedomostí. Keďže celý proces rozhodovania je založený na vedomostiach agenta, je jednoduché ovplyvniť jeho rozhodovanie jednoduchou úpravou. Tak isto aj samotný cieľ agenta, ovplyvňujúci rozhodnutie, sa môže jednoducho upraviť. Zmenou cieľa pre vodiča v kolóne môžeme docieliť kompletnú zmenu rozhodnutia o trase, ktorou sa vydá. Na rozdiel od jednoduchého reflexného agenta, kde by sme museli pozmeniť mnoho rôznych podmienených pravidiel.

Agent založený na úžitku

Ciele, ako také sú pre agenta dôležité, ale nie sú dostatočné na vytvorenie vysoko kvalitných správaní vo väčšine prostredí. Ide o to, že špecifikácia cieľa nerieši spôsob jeho dosiahnutia. Takže vodič auta si pri výbere cieľa musí zvoliť, ktorou cestou sa k nemu dostane, pričom niektorá môže byť rýchlejšia, iná bezpečnejšia a tak ďalej. Ciele poskytujú agentovi len binárny rozdiel stavov medzi úspešným(dosiahol svoj cieľ) a neúspešným (nedosiahol svoj cieľ). Vo všeobecnosti by agent mal byť schopný porovnávať rôzne stavy prostredia medzi sebou na základe, ako šťastný by agent bol v prípade dosiahnutia tohoto stavu. Úspešnosť agenta môžeme nazvať aj jeho úžitkom z danej situácie.

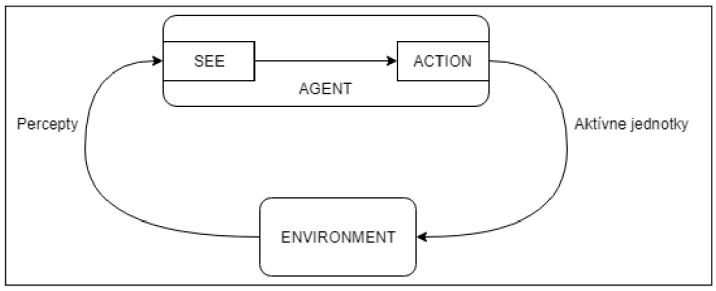
Keď sa už bavíme o úžitku agenta, je dôležité mať opatrenie, ktoré priraďuje sekvenciám stavov prostredia skóre, na základe ktorého sa dajú rozlišovať na priaznivé a nepriaznivé. Toto opatrenie sa vo všeobecnosti nazýva úžitková funkcia. Tento typ agenta nadobúda zmysel v prípade, že sa stretne s konfliktnými cieľmi, z ktorých sa dajú splniť len niektoré

a tie zvyšné sú recipročne výlučné. V našom príklade auta na ceste môže ísť o rýchlosť vs. bezpečnosť. V tomto type agenta práve úžitková funkcia rozhodne o kompromise medzi týmito dvoma potenciálnymi cieľmi. Takže agent je vďaka svojej úžitkovej funkcii schopný uskutočniť rozumné rozhodnutie medzi vzájomne sa vylučujúcimi cieľmi.

2.4 Vnímanie

V predchádzajúcich sekciách sme si načrtli tému o vnemoch a vnímaní agentov. Aký obrovský dopad majú na rozhodovanie agenta a ako reprezentujú praktické poznatky agenta o jeho okolitom prostredí. Vnímanie agenta môžeme považovať za jeden kompletný podsystém rovnakým spôsobom, ako klasifikáciu podsystémov v softvéromov inžinierstve. Za vnímaním agenta je myšlienka funkcie, ktorá je schopná spýtať sa prostredia na aktuálny stav okolo agenta. Táto vnemová funkcia môže byť pre niektoré typy agentov implementovaná v hardvéri samotného agenta. Alebo, ako je tomu v našom prípade, reprezentovaná funkciou, prípadne metódou vysoko úrovňového programovacieho jazyka. Výstupom tejto funkcie je vnem, ktorý obsahuje všetky dostupné informácie o prostredí agenta.

Implementácia tejto funkcie je väčšinou nezávislá od prostredia, od ktorého dostáva odpoveď. Avšak samotná odpoveď je ovplyvnená vlastnosťami prostredia, ako sme spomenuli v sekcii 2.2. Vnemová funkcia vracia agentovi zoznam všetkých dostupných vnemov o jeho okolitom prostredí a je už len na agentovi samotnom, ktoré informácie sú pre jeho proces rozhodovania relevantné. A taktiež ktoré informácie si agent uchová pre prípadné ďalšie rozhodovanie. S touto problematikou sme sa zaoberali v sekcii 2.3.



Obr. 2.2: Vnímanie a reakcie agenta vzhľadom na prostredie, v ktorom sa nachádza.

2.5 Organizácia agentov

Problém s organizáciou agentov nastáva pri potrebe plnenia cieľov, ktoré nedokážu byť uspokojené prácou jediného agenta. Inými slovami, je potrebných viacero agentov rovnakého alebo rôzneho typu na uspokojenie potrieb spoločného cieľa. V tomto prípade je nevyhnutné, aby sa agenti zhlukovali do skupín s rovnakými alebo aspoň podobnými cieľmi. Organizácia medzi skupinami agentov je nevyhnutná aj pre rozdelenie výpočetných síl nad určitým problémom. Balaji a Srinivasan navrhli nasledovné delenie organizácie agentov [9].

Hierarchická organizácia

Organizácia do hierarchie bola aplikovaná v množstve distribuovaných problémov. V hierarchickej architektúre agentov sú agenti rozložení do štruktúry podobnej stromovej štruktúre.

Pričom agenti na rôznych úrovniach tejto štruktúry majú rôzne úrovne autonómie. Dátové toky typicky smerujú od najspodnejších úrovní smerom k horným. Naopak kontrolné a rozhodovacie signály smerujú od agentov umiestnených na vrchole pomyselnej stromovej štruktúry smerom ku spodným, takzvane podriadeným agentom.

Na základe distribúcie moci medzi agentmi sa hierarchická štruktúra dá ďalej klasifikovať na jednoduchú a jednotnú hierarchiu.

- **Jednoduchá hierarchia** - Pri jednoduchej hierarchii existuje v hierarchickom strome práve jeden agent, ktorý vykonáva všetky rozhodnutia. Tieto rozhodnutia následne deleguje ďalej. Avšak zvyšok agentov už nerozhoduje o ničom, len nasleduje tieto, im pridelené rozhodnutia. Rizikom tohto prístupu je vysoká možnosť zlyhania agenta rozhodujúceho o všetkom.

- **Jednotná hierarchia** - Princíp jednotnej hierarchie je distribúcia rozhodovania medzi viacerých agentov, ktorí majú dostatočné informácie. Toto má za účelom zlepšiť odolnosť systému voči chybám a zlyhaniam. Tieto rozhodnutia sa oznamujú nadriadeným autoritám len v prípade konfliktných záujmov.

Koalície

Pri koaličnej architektúre skupina agentov dospeje k záveru, že ich spolupráca prinesie osobný úžitok každému individuálnemu agentovi. Koalície sa vytvárajú len dočasne a akonáhle je spoločný cieľ dosiahnutý, koalícia sa rozpadá. Často sa využíva jeden agent ako iniciátor koalície, pričom sa pomocou distribuovaných výpočtov dosiahla informácia, že koalícia je benefičná. Prienik agentov medzi viacerými koalíciami je možný. Avšak je nevyhnutné, aby agent dedikoval istú časť svojich schopností konkrétnej koalícií pre prípad, ak by tieto zdroje boli obmedzené. Toto prekrývanie koalícií zvyšuje náročnosť stratégie dohadovania medzi agentmi.

Teoreticky zostavenie jedinej koalície pozostávajúcej zo všetkých agentov v prostredí by maximalizoval výkon celého systému agentov. Toto by bolo z dôvodu, že každý agent by mal plný prístup k zdrojom a informáciám potrebným na výpočet podmienok pre ideálnu akciu. Avšak zostavenie takejto koalície je nepraktické z dôvodov nákladov na komunikáciu medzi agentmi, kde počet správ rastie exponenciálne s každým agentom a taktiež zdrojov systému.

Jednotlivé koaličné skupiny vznikajú a rozpadajú sa podľa stanovených cieľov systému.

Tímy

Tímová architektúra multiagentných systémov je podobná tej koaličnej. Odlišnosť je však badateľná, ak sa pozrieme na jednotlivé skupiny. Pri tímovej architektúre sa jednotlivý agenti snažia spolupracovať, aby zvýšili úžitok celej skupiny a nie len samých seba. Najväčšou výhodou takto zostavených tímov je napríklad lepšie skúmanie prostredia, a teda väčšie množstvo relevantných informácií. No však začleňovanie týchto informácií do tímovej štruktúry je náročnejšie. Pričom pri malých tímoch je práca s informáciami jednoduchšia. Kompromis medzi učenlivým a výkonným tímom musí byť ustanovený zavedením optimálnej veľkosti tímov. Táto skutočnosť však kladie vysoké nároky na zložité výpočty zabezpečujúce optimálnosť oproti koaličnej architektúre multiagentných systémov.

2.6 Komunikácia medzi agentmi

Ako sme už naznačili, komunikácia medzi agentmi je nevyhnutná pre ich úspešné nasadenie na riešenie reálnych problémov vo svete. A to hlavne pre výmenu informácií, znalostí a služieb s ostatnými agentmi, prípadne aj ľuďmi začlenenými v procese rozhodovania. Napríklad pri riešení problémov, ktoré nemôžu byť alebo nie sú efektívne riešené individuálne. Pre vykonanie úloh tohoto typu musia byť agenti schopní komunikovať a vyjednávať navzájom medzi sebou.

Vo všeobecnosti, žiaden agent nemôže prinútiť iného agenta k vykonaniu činnosti. To však neznamená, že nemôžu komunikovať. Agenti vykonávajú akciu komunikácie samovoľne, za účelom dosiahnutia svojich cieľov. Týmto spôsobom síce nemôžu priamo meniť vnútorný stav ostatných agentov, avšak sa ich môžu pokúsiť ovplyvniť pomocou zasielania informácií [12].

Koordinácia medzi agentmi je ďalšia dôležitá vlastnosť systému, ktorá je dosahovaná pomocou komunikácie. Z pohľadu agenta, účasť na spoločenskej akcii zahrňuje nároky na agenta. Pretože je vyžadovaná istá úroveň príspevku agenta do spoločnej práce. Ale taktiež výhody v zmysle, že agent by mal byť ohodnotený za svoj príspevok do takejto spoločenskej akcie. V systéme, kde spolupracuje viacero agentov na dosiahnutí jedného cieľa existujú podľa Jennings-a aspoň tri hlavné dôvody, prečo ich akcie musia byť koordinované [8]:

1. Existujú závislosti medzi jednotlivými akciami agentov. Závislosti medzi akciami vznikajú v prípade, že ciele ktoré agenti nasledujú, vzájomne súvisia. Napríklad z dôvodu, že rozhodnutie jedného agenta má vplyv na rozhodovanie ostatných agentov v skupine alebo z dôvodu možnej hrozby iným agentom.

2. Je potrebné dodržať globálne obmedzenia. Tieto globálne obmedzenia môžu byť napríklad v podobe obmedzených zdrojov alebo času. Koordinácia musí nastať z dôvodu individuálnych agentov, ktorí by konali v izolácií bez snahy optimalizovať svoj vlastný prínos do systému. Takto by s veľkou pravdepodobnosťou spolu presiahli uvedené globálne obmedzenia.

3. Žiadny individuálny agent v komplexnom multiagentnom systéme nemá kompetenciu, zdroje a ani informácie na to, aby riešil celé problémy osamote. Mnoho problémov je nemožné vyriešiť samostatne. Jedným z mnohých príkladov môže byť napríklad zdvíhanie ťažkej váhy, kde je potrebných viacero agentov z dôvodu nedostatočnej váhovej kapacity alebo z dôvodu stability ťažkého objektu. Tým pádom budovanie spoločností, ktoré sme rozoberali v sekcii 2.5, je nevyhnutné pre uspokojenie takýchto cieľov. Taktiež v systéme môžu existovať agenti rôznych typov, čo prináša ešte väčšie komplikácie na správu zdrojov jednotlivých agentov.

2.7 Distribuované riešenie problémov

Distribuované riešenie problémov je podmnožinou oboru distribuovanej umelej inteligencie. Rieši otázku, ako prinútiť viacerých agentov k spolupráci pri riešení problému vyžadujúceho spoločné úsilie. Distribuované riešenie problémov funguje na základe rozkladu problému na podproblémy, ktoré sú nezávisle vyriešené a následne ich riešenia sú zložené do jedného celku, ktorý reprezentuje riešenie pre pôvodný problém.

Multiagentný systém sa dá považovať za systém distribuovaného riešenia problémov v prípade splnenia istých predpokladov. Medzi tieto predpoklady podľa autorov Edmund H. Durfee a Jeffrey S. Rosenschein [6] patria:

Benevolenčný predpoklad - Jeden z ponúknutých predpokladov, na základe ktorého sa systém dá klasifikovať ako systém distribuovaného riešenia problémov je, či sa jednotliví agenti dajú považovať za benevolentných. Definícia benevolentnosti v tomto prípade predstavuje schopnosť agentov pomáhať si kedykoľvek je to možné. Agenti nepotrebujú byť podplácaní alebo inak presviedčaní o potrebe pomôcť s vykonávaním nejakej úlohy, ktorú iný agent potrebuje dokončiť. Agenti chcú vykonávať tento typ úloh z dôvodu, že tak boli naprogramovaní.

Napriek benevolentnosti agenta avšak nie je zaručená spolupráca a vzájomná koordinácia medzi skupinami agentov. Aj napriek vlastnosti agentov, chcieť si navzájom pomáhať. Náročnosť načasovania a možnosť vnímania len lokálneho prostredia môže viesť k narušeniu spolupráce a zlej koordinácií akcií agentov. Do tohto stavu sa môže systém dostať napríklad pomalým vykonávaním aktuálnych činností, ktoré sa môže prejavovať ako zanedbávanie plnenia prioritných úloh.

Predpoklad spoločných cieľov - Motivácia pre zhovievavosť medzi agentmi vychádza z vlastnosti skupín v multiagentnom systéme nasledovať nejaký spoločný cieľ, ktorý môže byť globálnym cieľom systému. To znamená, že ak si všetci agenti cenia rovnaký výstup skupinovej aktivity, tak sa každý z nich snaží prispieť akýmkoľvek spôsobom ku globálnemu výsledku. Keďže globálny výsledok vedie k úspešnosti systému.

Nastávajú však aj situácie, keď niektoré lokálne rozhodnutia skupiny môžu viesť ku globálnej nesúvislosti akcií. Bez striktných pokynov o zodpovednostiach a záujmoch sa agenti môžu navzájom zahltiť prehnanými informáciami. V iných prípadoch môžu agenti začať pracovať so vzájomne výlučnými myšlienkami, čo spôsobí zasielanie zavádzajúcich informácií, ktoré môže odchýliť ostatných agentov od spracovania dôležitejších úloh.

Predpoklad centralizovaného návrhára - Predpoklad využitia centralizovaného návrhára v sebe zahrňuje oba predchádzajúce predpoklady, keďže ciele tohto návrhára by sa zaoberali spoluprácou jednotlivých častí systému, čo by pravdepodobne zapríčinilo benevolentnosť týchto častí, teda agentov. Keďže tento centralizovaný návrhár by mal v sebe zhrnutú podstatu systému a tým pádom aj hlavné ciele, ktoré sa systém snaží dosiahnuť. Zameranie agentov by bolo možné kalibrovať a mechanizmy vyjadrovania a konania agentov štandardizovať.

Kapitola 3

Vývoj agentov

Agenti sú v podstate softvérové jednotky schopné vysporiadať sa so zmenami prostredia a rozmanitými požiadavkami na vlastnosti, akými sú napríklad autonómia, mobilita alebo inteligencia, kooperativita a reaktívnosť. Na pokrytie všetkých týchto požiadaviek je nevyhnutné špecifikovať charakteristiky potrebné pre návrh a implementáciu schopného agenta. Avšak zostáva otázkou, aké majú tieto charakteristiky byť. Podľa autorov, ktorí napísali Formálnu sémantiku pre abstraktný agentný programovací jazyk [7] sú práve takými vlastnosťami:

- Agent má komplexný vnútorný stav, ktorý pozostáva z toho, čomu agent verí, z jeho túžob, plánov a zámerov, ktoré sa môžu časom meniť.
- Agenti konajú pro-aktívne na základe zmien v prostredí, táto problematika bola riešená v sekcii 2.1.
- Agenti majú reflektívnu schopnosť zdôvodniť svoje akcie.

Z tohto zoznamu vlastností môžeme vydedukovať, že agent musí byť cieľovo orientovaný. Mať zoznam cieľov, ktoré sa pomocou svojho správania snaží dosiahnuť. Ale taktiež musí vedieť transformovať svoje domnienky o okolitom prostredí. Na základe týchto predpokladov si agent vytvára plány, ktoré využíva ako nástroj k dosiahnutiu svojich cieľov.

3.1 Špecifikácia FIPA

The Foundation for Intelligent Physical Agents, v preklade základy pre inteligentných fyzických agentov, ďalej už len FIPA, definuje referenčný model pre agentné platformy a množinu služieb, ktoré by mali byť poskytované na zabezpečenie systému plne schopného výmeny a využitia informácií. Na začiatok definujú referenčný model platformy. Pričom začínajú identifikáciou dôležitých služieb a kľúčových agentov pre správu platformy a popisujú jazyk správy agentov. Tri povinné role boli identifikované ako nevyhnutné pre každú platformu agentov. **Systém riadenia agentov** (AMS) je agent, ktorý vykonáva dohľad nad prístupom a použitím platformy. Je zodpovedný za udržovanie adresára s prítomnými agentmi a na správu ich životného cyklu. Ďalšou rolou v systéme je **služba prenosu správ** (MTS) poskytujúca spôsob jednoduchého kontaktovania agentov v rámci a aj mimo platformy. ACC je základná komunikačná metóda, ktorá ponúka spoľahlivé, usporiadané a presné smerovacie služby pre správy. Poslednou rolou je **sprostredkovateľ adresárov** (DF), táto rola slúži agentom ako služba žltých stránok v systéme. [5]

Špecifikácia FIPA taktiež definuje komunikačný jazyk agentov (ACL) používaný na výmenu správ medzi agentmi. Tento jazyk popisuje kódovanie a sémantiku správ, avšak nediktuje spôsob na prepravu týchto správ.

Kapitola 4

Nástroj JADE

Nástroj Java Agent Development Framework, ďalej už len JADE, reprezentuje softvérové prostredie schopné budovať agentné systémy v súlade so špecifikáciami FIPA. Tento nástroj je plne vyvinutý v programovacom jazyku Java. Veľkou výhodou tohoto nástroja sú vlastnosti [2] ako napríklad:

- Nájstroj JADE je **jednoduchý na použitie** - Komplexita behu a správy agentov je v nástroji JADE skrytá za sadou jednoduchých a intuitívnych metód aplikačného rozhrania.
- Filozofia, ktorú autori nazývajú **platíš ako ideš** - Čo prakticky znamená, že programátor nemusí využívať všetky možnosti, ktoré nástroj JADE ponúka. Vlastnosti, ktoré týmto spôsobom programátor nevyužíva, nemajú žiaden dopad na potrebu byť s nimi oboznámený.

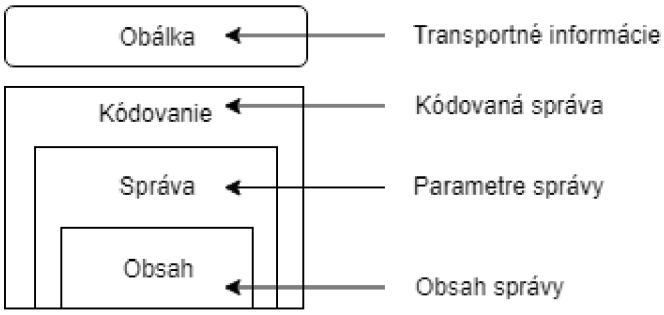
Každý agent je v nástroji JADE implementovaný na vlastnom vlákne. Viac vláknové riešeni pre prípady kooperatívnych úloh ponúka samotné prostredie programovacieho jazyka Java. JADE je vďaka tejto implementácii schopný naplánovať kooperatívne správanie agentov jednoduchým a efektívnym spôsobom. Implementácia tohoto nástroja obsahuje už vopred zopár typov správaní, ktoré sa pravidelne využívajú na urýchlenie základných úloh programátora. [3]

4.1 Služba prenosu správ

Komunikačná architektúra nástroja JADE ponúka efektívne zasielanie správ, pričom zabezpečuje vytvorenie a správu fronty prichádzajúcich správ. Táto fronta je privátna pre každého agenta zvlášť, keďže každý agent príjíma a spracúva len svoju vlastnú poštu. Pri prístupe do svojej schránky má agent na výber z viacerých módov. Nástroj JADE ponúka plnú podporu komunikačného modelu podľa špecifikácie FIPA od obálok jednotlivých správ až po celkový transportný protokol. Transportný mechanizmus, ktorý je využívaný, sa podľa autorov dá prirovnať k chameleónovi, keďže sa prispôsobuje na základe situácie. A to práve vybraním najvhodnejšieho prenosového protokolu. [1]

Táto služba je poskytovaná agentskou platformou za účelom prenášania správ medzi agentmi na ktorejkoľvek platforme alebo medzi agentmi z rôzných platforiem. Správy sú zapúzdrené v transportnej obálke, ktorá obsahuje informácie, ako napríklad príjemcu danej správy. Štruktúra tejto správy je podľa autorov nástroju JADE nasledovná. Najskôr agent

obalí požadovaný obsah správy do štruktúry, ktorá pridáva parametre správy. Následne sa táto štruktúra zakóduje a pridá sa transportná obálka, obsahujúca napríklad údaje o príjemcoch a odosielateľovi 4.1.

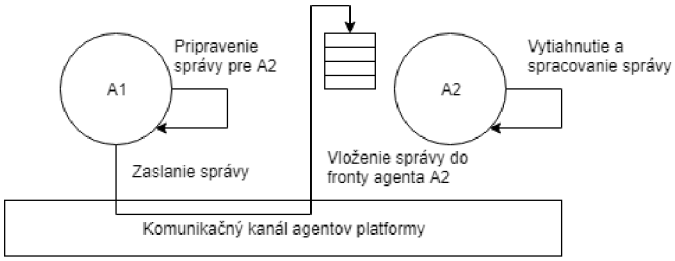


Obr. 4.1: Štruktúra správy agentského komunikačného jazyka podľa špecifikácií FIPA

Služba prenosu správ je realizovaná pomocou komunikačného kanálu agentov, ktorý je povinný všetky správy obdržané v súlade s transportnými inštrukciami obsiahnutými v správach preposlať ďalej. Pričom smie čítať len obálku týchto správ, v žiadnom prípade nie ich obsah. Počas prenášania správ môže byť nevyhnutné pre dokončenie prenosu využitie zvyšných dvoch služieb na úspešnú lokalizáciu a doručenie správy. Každý komunikačný kanál, ktorým správa prechádza môže, ale nemusí, pridať dodatočné informácie do jej hlavičky, ale nikdy nesmie prepísať už existujúce záznamy.

Komunikácia medzi agentmi

Komunikácia agentov tvorí významnú rolu v nástroji JADE a je implementovaná v súlade s FIPA špecifikáciami spomenutými v sekcii 3.1. Komunikačný model v nástroji JADE je založený na asynchrónnom predávaní správ. To v praxi znamená, že každý agent má svoju vlastnú poštovú schránku, do ktorej mu komunikačný kanál vkladá správy určené práve pre neho. O každom doručení správy do fronty je agent oboznámený. Avšak samotné vyzdvihnutie správy z pošty je už len na uvážení programátora. Proces zasielania, doručenia a vyzdvihnutia správy je znázornený na obrázku 4.2.



Obr. 4.2: Princíp asynchrónneho predávania správ medzi agentmi

Každá správa obsahuje nasledovné položky [4]:

- Odosielateľa správy.
- Zoznam príjemcov.

- Komunikatívnum, je vlastne reprezentácia zámeru správy. Odzrkadľuje zámer odosielateľa, čo sa snaží s danou správou dosiahnuť.
- Obsah zahrňujúci informácie, ktoré majú byť danou správou prenesené.
- Jazyk obsahu naznačujúci syntax použitú k vyjadreniu obsahu. Obaja, odosielateľ aj príjemca, sa musia zhodnúť na rovnakom jazyku obsahu.
- Prípadne ďalšie položky.

4.2 Systém riadenia agentov

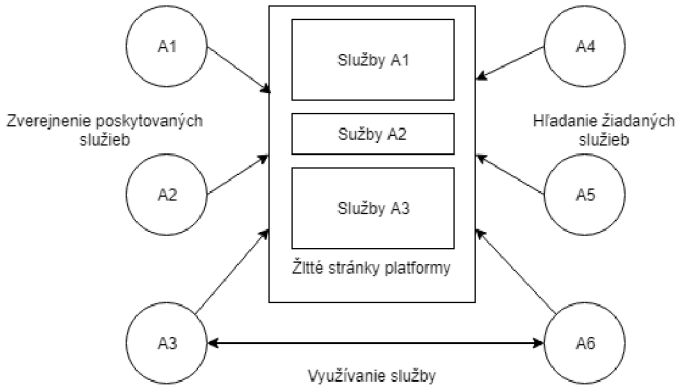
Systém riadenia agentov je nevyhnutným komponentom agentnej platformy. Zabezpečuje správu a operácie agentnej platformy, ako napríklad vytváranie a mazanie agentov. Ale taktiež zabezpečuje migráciu agentov medzi platformami. Každý agent sa povinne musí registrovať do tohto komponentu pre obdržanie identifikátoru. Identifikátor ho následne činí súčasťou platformy, ku ktorej patrí daný systém riadenia agentov. Každá agentná platforma má svôj vlastný systém riadenia agentov a životný cyklus agenta končí odregistrovaním sa z tohoto systému riadenia agentov. Po odregistrovaní agenta sa jeho identifikátor stáva voľným pre konkrétnu platformu a je možné ho prideliť ďalšiemu prichádzajúcemu agentovi do systému.

Systém riadenia agentov je oprávnený vyžiadať vykonanie istých funkcií, nevyhnutných pre správu platformy. Jednou z týchto funkcií môže byť požiadanie agenta o termináciu a v prípade ignorácie takejto žiadosti je systém riadenia agentov oprávnený ukončiť životný cyklus agenta násilne. Na jednej agentnej platforme môže existovať jediná inštancia systému riadenia agentov [4].

4.3 Sprostredkovateľ adresárov

Sprostredkovateľ adresárov je na rozdiel od prechádzajúcich komponentov voliteľný. Jeho úlohou je poskytovať službu žltých stránok pre ostatných agentov. Táto služba si udržuje presný, kompletný a aktuálny zoznam všetkých agentov a musí poskytovať najpresnejšie informácie o týchto agentoch uložených vo vlastných adresároch. Agentná platforma môže podporovať akýkoľvek počet služieb tohto typu, pričom tieto môžu byť registrované medzi sebou a tvoriť federácie.

Pre registráciu svojich služieb si každý agent najskôr musí vyhľadať dostupnú službu a následne požiadať o registráciu svojho vlastného popisovača. Registrujúci agent nie je viazaný tejto registrácii vyhovieť. Agent je následne oprávnený sa odregistrovať od zdieľania svojich služieb. V tom prípade agent poskytujúci službu sprostredkovania nie je povinný hľadať náhradu za odregistrovanú službu a taktiež nie je povinný udržiavať informácie o odregistrovanom agentovi naďalej vo svojich adresároch. Kedykoľvek a z akýchkoľvek dôvodov agent môže požiadať o zmenu svojho popisu registrovaného s touto službou [4].



Obr. 4.3: Sprostredkovanie služieb agentom v systéme pomocou služby DF

Kapitola 5

Multi-Agent Systems Simulation Platform (MASSim)

MASSim je simulačný softvér použitý v súťaži zameranej na multiagentné programovanie. V súťaži proti sebe zápasili dve skupiny agentných programov vo vopred určenej hre. Táto simulácia beží v diskrétnych krokoch. Agenti sa pripájajú vzdialene na server, na ktorom prebiehajú zápasy, z ktorého následne dostávajú vnemy a odosielajú naň svoje akcie. Tieto akcie sú vzápätí vykonané serverom.

5.1 Scenár simulácie

Scenár multiagentnej súťaže pozostáva z dvoch tímov agentov, ktorí sa pohybujú ulicami realistického modelu mesta. Cieľom každého tímu je vybudovanie a udržanie si najväčšieho množstva studní. Studne majú význam generátorov skóre v tejto súťažnej hre. Sú budované pomocou massia, táto entita reprezentuje menu hry a je získavaná pomocou dokončovania prác.

Agent je v tomto scenári charakterizovaný:

- **Batériou** - ktorá reprezentuje vzdialenosť, akú môže daný agent prejsť bez potreby nabíjania.
- **Kapacitou** - táto charakteristika odráža objem, ktorý dokáže agent uniesť.
- **Rýchlosťou** - jednoducho reprezentuje rýchlosť pohybu mestom.
- **Videním** - odzrkadľuje vzdialenosť, ktorú agent dokáže vnímať.
- **Zručnosťou** - určuje rýchlosť, akou je agent schopný vykonávať určité úlohy.

Scenár obsahuje štyri typy agentov: drony, motocykle, autá a nákladné autá. Pričom sa im zostupne zvyšuje kapacita, ale zároveň znižuje mobilita. Agenti, skladovacie zariadenia a skládky sú na začiatku hry náhodne rozmiestnené po mape.

Kapitola 6

Koaličné hry

Zameranie koaličných hier nie je na akcie jednotlivých agentov, ale na spôsob rozdelenia akcií medzi spolupracujúcu skupinu agentov. Myšlienka sa zaoberá množinami agentov, ktorí môžu spolu vytvárať potenciálne koalície agentov na následnú spoluprácu. V skutočnosti nás pri formovaní koalícií nezaujíma, ako si agenti samotnú prácu v nich rozdelia alebo ako bude prebiehať koordinácia v takejto koalícii. Čo nás však zaujíma, je akési ohodnotenie množiny agentov, ktorí by mohli koalíciu vytvoriť v prípade, že by toto ohodnotenie bolo dobré. Otázkou preto zostáva, ktoré sú tie koalície, ktoré sme ochotní vytvoriť, alebo ktoré koalície by mali byť agenti ochotní vytvoriť.

Jednou z možností je vytvorenie veľkej koalície, ktorá zahrňuje všetkých agentov v systéme. Toto riešenie by fungovalo v super-additívnom prostredí, kde prírastok agenta ku koalícií nijako nezvyšuje náklady na udržanie takejto koalície.

6.1 Prideľovanie úloh

Z dôvodu potreby plnenia úloh v multiagentnom prostredí je nevyhnutné zabezpečiť spoluprácu agentov na plnenie týchto úloh, keďže niektoré z nich môžu vyžadovať spoluprácu viacerých agentov na svoje splnenie. Z určitej množiny úloh, priraďujeme jednotlivé úlohy skupinám agentov pre ich splnenie. Práve toto rozdelenie úloh do skupín je nevyhnutné v prípade, že úloha nie je splniteľná jediným agentom, napríklad z dôvodu obmedzenej kapacity tohto agenta alebo iných nárokov na zdroje, ktorými agent nedisponuje.

Zmyslom prideľovania úloh ku skupinám je maximalizácia zisku z ich vykonania. Pre docielenie tejto maximalizácie je potrebné zvážiť optimalizáciu priradenia jednotlivých úloh ku skupinám agentov, ktoré nazývame koalíciami.

6.2 Algoritmus Shehory-Kraus

Algoritmy navrhnuté Onn Shehorym a Sarit Krausom [11] , ďalej už len Shehory-Krausove algoritmy, sú nenásytné distribuované algoritmy, ktoré riešia delenie a pokrytie množín. Sú navrhnuté pre špeciálny prípad autonómnych agentov v nie super additívnom prostredí, ktoré funguje na podobnom princípe ako systém distribuovaného riešenia problémov.

Algoritmy poskytujú výsledky riešenia problému aj v prípade, že sú ukončené predčasne, čiže skôr ako bol ich koniec mienený samotným agentom. A zároveň, toto riešenie je vždy lepšie, alebo pri najhoršom rovnako dobré, ako určoval vnútorný stav agenta pred spustením tohoto procesu.

Algoritmus pozostáva z dvoch častí:

1. V prípravnej fáze algoritmu sú všetky možné koalície distribuovane zostavené a ich počiatočné hodnoty sú vypočítané.
2. Hlavná fáza algoritmu pozostáva z iteratívneho výpočtu, v ktorom nastávajú dve podfázy:
   - Koaličné hodnoty sú znova prepočítané z dôvodu, že algoritmus na vytváranie koalícií vyžaduje aktuálnosť informácií o hodnotách možných koalícií na umožneni výberu tých najlepších.
   - Agent vyhodnotí najlepšie koalície a snaží sa ich vytvoriť.

Náročnosť algoritmu

Delenie a pokrývanie množín v prípade autonómnych agentov nadobúda exponenciálnej komplexnosti z dôvodu počtu všetkých možných koalícií, ktoré je možné vytvoriť. Počet všetkých možných koalícií sa rovná (2^n).

Umiestnením obmedzenia na povolené koalície je možné tento počet znížiť. Obmedzenia závisia od špecifikácií riešeného problému, pričom to môže byť napríklad obmedzenie úlohy špecifikujúcej konkrétny počet potrebný pre jej riešenie. Alebo určením minimálnej a maximálnej veľkosti koalícií, pričom nám toto obmedzenie vytvorí výsek koaličných množín z celého stavového priestoru. Tieto opatrenia vo všeobecnosti znižujú počet koaličných množín, ktoré musia byť prepočítavané v každom kroku algoritmu.

Z dôvodu exponenciálne narastajúcich výpočtových a komunikačných nákladov pri formácii väčších koalícií, je v algoritme implementovaná heuristika, ktorá je reprezentovaná celočíselnou hodnotou k a obmedzuje hornú hranicu veľkosti vytváraných koalícií. Toto obmedzenie redukuje počet koalícií na $O(n^k)$ [11].

Koaličný stav na začiatku algoritmu pozostáva z n samostatných agentov, ktorí začnú vyjednávať medzi sebou a krok za krokom vytvárať koalície. V našom prípade, agent po zlúčení do koalície opúšťa proces vytvárania koalícií a iba zostávajúci agenti pokračujú vo vyjednávaní. Zníženie počtu agentov pokračujúcich vo vyjednávaní má priamy dopad na komunikačné a výpočtové zaťaženie.

Predbežná dohoda o rezdelení výpočtov

Agenti sa ešte pred začiatkom vytvárania koalícií musia dohodnúť na rozdelení výpočtov medzi sebou. Na dosiahnutie rozdelenia, si každý agent musí vytvoriť zoznam všetkých možných permutácií agentov tak, že každá permutácia obsahuje aj jeho. Tieto permutácie sú v algoritme obmedzené na veľkosť k agentov.

Následne agent prechádza zoznam takto vybudovaných možných koalícií a postupne kontaktuje každého agenta potenciálnej koalície. V prípade, že je prvým agentom zaoberajúcim sa vytvorením tejto koalície, stáva sa zodpovedným za opakované prepočítavanie koaličných hodnôt. Agent postupne odstraňuje zo zoznamu potenciálnych koalícií tie koalície, z ktorých agenta kontaktoval iný agent, ktorý sa tým pádom zaviazal výpočtom hodnôt danej koalície. Táto časť sa opakuje do chvíle, kedy agent nemá v zoznam žiadne ďalšie koalície pre kontaktovanie. Agent má v tomto momente všetky potrebné informácie k výpočtu koaličných hodnôt a súčastne všetky potenciálne koalície sú rozdelené medzi agentov.

Každý agent A_i by mal vykonať nasledujúce kroky:

1. Vypočítať všetky kombinácie až do k agentov absahujúce A_i a uložiť ich do P_i, ktorá reprezentuje množinu potenciálnych koalícií agenta A_i.

2. Pokiaľ P_i nie je prázdne urob:

   - Kontaktuj agenta A_j, ktorý je členom potenciálnej koalície z P_j.
   - Zaviaž sa k výpočtu hodnôt podmnožiny S_{ij} spoločných potenciálnych koalícií.
   - Odstráň S_{ij} z P_i. Pridaj S_{ij} do zoznamu záväzkov L_i.
   - Za každého agenta A_k, ktorý ťa kontaktoval odstráň z P_i množinu S_{ki} ku ktorej sa agent A_k zaviazal.
   - Vypočítaj hodnoty pre koalície, ktorým si zaviazaný.
   - Opakovane kontaktuj ostatných agentov až pokiaľ $P_i = \{A_i\}$.

Algoritmický popis prevzatý z [11].

Distribuované, opakované výpočty koaličných hodnôt

Algoritmus na výber koalícií vyžaduje opakované prepočítavanie koaličných hodnôt a to z dôvodu ich aktuálnosti kvôli optimálnemu rozhodnutiu pri výbere koalície. Z tohto dôvodu sa hodnoty musia iteratívne prepočítavať. Každý z agentov by si mal pre zjednodušenie uchovávať zoznam koalícií, pre ktoré je potreba vypočítať nové koaličné hodnoty.

Algoritmus následovne prechádza všetky koalície, pre ktoré treba vypočítať nové koaličné hodnoty. Pričom sa snaží vyčísliť vektor potenciálnych schopností danej koalície pomocou spočítania všetkých nevyužitých schopností agentov v koalícii. Pre každú z týchto koalícií vytvorí zoznam očakávaných výsledkov pri vykonávaní konkrétnych činností. Tento zoznam pozostáva z hodnôt, ktoré reprezentujú prienik vektoru potrebných schopností na dokončenie činnosti a vektoru schopností agentov koalície. Na konci agent vyberie najlepšie ohodnotený pomer činnosti ku koalícii a zapamätá si ho ako koaličnú hodnotu. Na záver, cena danej koalície je vyjadrená vzťahom $1/V_c$, kde V_c reprezentuje hodnotu koalície.

Každý agent A_i by si mal uchovávať aktuálny zoznam koalícií, ku ktorým má vypočítať potrebné koaličné hodnoty označený ako L_i^{cr} na začiatku prázdny. Následne by mal každý agent A_i nasledovať tieto kroky:

Cykliť a pre každú koalíciu C z L_i^{cr} vykonať:

1. Vypočítaj vektor potenciálnych schopností koalície B_c^{pc} spočítaním schopností členov koalície.

2. Vytvor zoznam E_c očakávaných výsledkov prevedenia úlohy T koalíciou C. Pre každú úlohu $t_j \in T$ urob:

   - Skontroluj aké schopnosti B_j sú nevyhnutné pre splnenie úlohy t_j.
   - Porovnaj B_j s B_c^{pc}, čím zistíš, ktoré úlohy sú splnitelné koalíciou C.
   - Keď $\forall i, b_i^j \in B_j \leq b_{pc}^i \in B_c^{pc}$. Čo znamená, že t_j môže byť splnené C. Vypočítaj očakávaný výsledok e_j úlohy t_j vzhľadom na $|C|$. Ulož e_j do E_c.

3. Medzi očakávanými výsledkami uloženými v zozname E_c vyber maximálny. Táto hodnota reprezentuje koaličnú hodnotu V_c.

4. Vypočítaj cenu koalície.

Algoritmický popis prevzatý z [11].

Výber koalícií

V poslednej časti algoritmu sa agenti iteratívne rozhodujú, ktorá koalícia má byť uprednostnená a vytvorená. Na konci predchádzajúcej časti má každý agent v systéme zoznam koalícií, ktorým vyhodnocoval koaličné hodnoty, ako aj samotný zoznam koaličných hodnôt. V tejto časti agenti tvoria medzi sebou koalície na základe vypočítaných koaličných hodnôt.

Z dôvodu zjednodušenia si určíme pomer medzi cenou koalície a jej veľkosti ako váhu koalície vzťahom $w_i = c_i/|C_i|$.

Každý agent si v zozname koalícií nájde koalíciu s najmenšou váhou w_j, ktorú následne oznámi všetkým agentom v systéme ako svoju najlepšie ohodnotenú koalíciu. Agenti si potom spoločne vyberú najnižšiu z oznámených hodnôt. Táto hodnota reprezentuje dvojicu koalície a k nej najlepšie ohodnotenú prácu a odstráni si zo zoznamu kandidátov na koalície všetkých členov tejto vybranej koalície. V prípade, že agent je členom vytváranej koalície, tak sa pridá ku ostatným. Nakoniec sa odstráni úloha zo zoznamu úloh a je nahradená novo vzniknutou koaláciou.

Proces počítania a určovania preferovaných koalícií bude prebiehať až pokiaľ nebudú všetci agenti zadelení do koalícií alebo pokiaľ nebudú existovať ďalšie úlohy, ktoré by mohli byť novo vzniknutým koalíciám pridelené. Niektoré hodnoty musia byť počas procesu vytvárania koalícií neustále prepočítavané z dôvodu, že sú priamo závislé na prideľovaných úlohách a koaličnej konfigurácii. Tým pádom koaličné hodnoty, ktoré boli vypočítané na základe už pridelených úloh, musia byť prepočítané.

Každý agent A_i by mal iteratívne vykonávať nasledujúce kroky:

1. Nájdi v L_i koalíciu C_j s najmenšou w_j.
2. Oznám koaličnú váhu w_j ktorú si našiel všetkým agentom.
3. Vyber najnižšiu z oznámených hodnôť. Táto w_{low} bude zvolená všetkými agentmi. Vyber patričnú koalíciu C_{low} a úlohu t_{low}.
4. Odstráň členov koalície C_{low} zo zoznamu kandidátov pre nové koalície.
5. Ak si členom koalície C_{low} pridaj sa k ostatným agentom a vytvor koalíciu.
6. Z L_i odstráň všetky koalície obsahujúce odstránených agentov.
7. Odstráň z T vybranú úlohu t_{low}.
8. Pridaj do L_i^{cr} koalície z L_i ktorých hodnoty majú byť prepočítané.

Algoritmický popis prevzatý z [11].

Kapitola 7

Implementácia

Implementácia tejto práce, ako bolo už skôr spomenuté v kapitole 4, bola v programovacom jazyku Java. Využitý bol nástroj na budovanie agentných platforiem JADE. Základom práce je implementácia triedy `CoalitionAgent`. Táto trieda obsahuje ďalej privátne triedy, ktoré reprezentujú chovania agenta a pomocné metódy, na špeciálne účely. Ohľad sa bral hlavne na efektivitu komunikácie so simulačným serverom a medzi agentmi navzájom. Keďže algoritmus musí rozhodnúť o vytvorených koalíciach v reálnom čase pričom preskúmava obrovský stavový priestor tvorený rôznymi kombináciami agentov.

Trieda `CoalitionAgent` využíva vlastnosť dedičnosti z objektovo orientovaného programovania a je odvodená od triedy `Agent`, ktorá je definovaná v balíčku `jade.core`. Tento balíček obsahuje základné triedy na budovanie agentnej platformy. Dedenie z triedy `Agent` zaručuje, že naša trieda je následne vnímaná ako agent v systéme. Avšak na využitie poskytovaných vlastností agentov je potrebné zaimplementovať ešte verejnú metódu s návratovým typom `void`, ktorá sa podľa abstraktnej triedy `Agent` nazýva `setup` a nepreberá žiadne parametre. Táto trieda bude vytvorená hneď ako prvá, pri vytvorení nového agenta z našej triedy.

Rozhranie triedy `Agent` nám umožňuje využiť metódu `takeDown`, ktorá je volaná vždy pri ukončení agenta.

7.1 Pomocné metódy

Na zabezpečenie behu programu sú v projekte implementované tri pomocné metódy, ktoré sú počas behu agenta využívané na zabezpečenie zdrojov alebo komunikáciu. Šetria jednak úsilie agenta, ako aj programátora a súčasne znižujú pravdepodobnosť výskytu chýb v projekte.

Metóda `nameToInt`

Na začiatok by sme chceli len spomenúť jednoduchú metódu, ktorá preberá hodnotu a transformuje ju do súčtu ASCII hodnôt znakov. Táto jednoduchá, napriek tomu užitočná funkcionalita je využitá hneď na začiatku celého procesu. A to pri výbere správcu informácií, Kde si medzi sebou agenti navzájom vyberajú správcu informácií na základe hodnoty súčtu znakov v ich názve.

Metóda `searchDF`

Metóda `searchDF` je využívaná v priebehu celého procesu od vzniku agenta až po vytváranie koalícií. Využíva sa, ako už meno naznačuje, na prehľadávanie adresárov služieb. Inými slovami ide o dotazovaciu metódu na žltté stránky platformy.

Metóda preberá objekt `ServiceDescription`, ktorý popisuje hľadanú službu v systéme. Metóda vezme túto pridelenú službu a obalí ju do popisovača agenta. Ďalej zašle žiadosť na žlté stránky systému o vyhľadávanie požadovaných služieb. Následne buď vráti zoznam popisovačov alebo hodnotu null, ktorá reprezentuje nulovú zhodu.

Metóda `broadcast`

Z dôvodu neustálej komunikácie medzi agentmi sa občas využíva zaslanie obsahu všetkým agentom v systéme. Toto zasielanie správ každému agentovi je ekvivalentné všesmerovému vysielaniu zo sietí.

Táto metóda si vyžaduje, aby dostala obsah správy, ktorý má rozoslať a špecifikáciu komunikatíva, ktorá určuje zámer správy. Pre zabezpečenie rozosielania avšetkým agentom si metóda vyžaduje, aby bol každý agent prihlásený na príjmanie tohto typu vysielania. Na vyhľadávanie týchto agentov sa využíva metóda `searchDF`, ktorá bola spomenutá v predchádzajúcej podsekcii. Služba, pod ktorou musia byť agenti zaregistrovaný sa nazýva *broadcast*.

Metóda generuje správu určenú množine agentov, ktorá je vrátená metódou `searchDF` a následne ju zasiela.

7.2 Implementácia agenta

Implementácia agenta začína pri metóde `setup`, ktorou sa začína životný cyklus každého agenta. Považujem za dôležité spomenúť, čo sa v tejto metóde nachádza a dosahuje. Ďalej sa agent skladá z atribútov, ktoré reprezentujú vnútorný stav agenta a v neposlednom rade zo správaní, ktoré určujú akcie agenta. V tejto sekcii si rozoberieme implementáciu jednotlivých správaní agenta, ktoré sú reprezentované ako privátne triedy `CoalitionAgent`.

Metóda `setup`

Ako sme už spomenuli, metóda `setup` je prvou metódou, ktorá je volaná pri začiatku životného cyklu agenta. Z tohto dôvodu je to ideálna metóda na umiestnenie všetkých možných aj nemožných inicializácií. A celkovo rozbehnutie agenta.

V našom prípade sa v metóde `setup` nachádza úplne na začiatku registrácia agenta k príjmaniu všesmerového vysielania ostatných agentov. Toto je dôležité hlavne z dôvodu, aby bol agent schopný čo najskôr prijímať správy zasielané týmto spôsobom. Následne pokračuje registrácia agenta k simulačnému serveru pomocou rozhrania prostredia `EnvironmentInterfaceStandard` a registrácia agenta k entite, ktorú bude v simulácii reprezentovať. Zoznam entít sa načíta zo špeciálneho súboru, v ktorom sú dáta o entitách uložené vo formáte *JSON*.

Poslednou akciou čo sa v metóde `setup` vykonáva, je pridávanie chovaní.

Správanie `ReceiveMessages`

Prvým z implementovaných správaní v tomto projekte bolo práve `ReceiveMessages`. Toto správanie zabezpečuje prijímanie správ a ich zadelenie u každého agenta pomocou metód poskytnutých nástrojom Jade.

Príjmanie a spracovanie správ je kľúčovou vlastnosťou agentov. Vďaka schopnosti komunikácie dokážu vyjednávať a poskytovať informácie ďalším agentom v systéme. Komunikácia prebieha v dvoch fázach. Najskôr sa agent snaží prijať správu zo svojho zásobníka správ, kam mu ich doručuje agentná platforma. Dôležitým prvkom pri príjmaní správ je overovanie, či správa prišla. V prípade, že metóda `receive()` nemá žiadnu správu, ktorú by vytiahla z fronty správ, vracia `null`. Z dôvodu vyhnutia sa chybám pri práci s objektom `null` , je dôležité prerušiť ďalšie spracovanie správ.

Následne sa správy delia na základe typu zámeru a na najnižšej vrstve sa kontroluje inštancia objektu, ktorý je obsiahnutý v tele správy. Agent potom na základe tohto následne vykonáva ďalšie obslužné rutiny.

```
# receiving a message
msg = receive()

# message verification
if msg is Null:
    end

  # decision making based on mssage performative
switch(msg.getPerformative()):

    # performative type ACLMessage.INFORM
    case (INFORM):
        if content instanceof AgentInformation:
            ...

    # performative type ACLMessage.REQUEST
    case (REQUEST):
        if content instanceof HashMap:
            ...
```

Výpis 7.1: Príjem správ

Správanie `HandlePercepts`

Spracovanie vnemov od simulačného serveru potrebujeme najmä z dôvodu udržania aktuálnosti informácií, ktorými agenti disponujú. Na základe týchto znalostí agenti ohodnocujú jednotlivé koalície. Ale taktiež sa agenti dozvedajú o prácach, ktoré systém zadáva na splnenie. Každý agent dostáva a následne spracováva len vlastné vnemy, ktoré sa filtrujú na základe asociovanej entity.

Prvým krokom je spracovanie vnemov. Pri spustení agenta sa so správou typu `SIM-START`, ktorá dorazí od simulačného serveru cez rozhranie `EnvironmentInterfaceStandard`. V tejto správe sú obsiahnuté pod menom agenta dodatočné informácie o entite ktorú reprezentuje. Ako napríklad rola v simulácií, atribúty a pozícia. Tieto informácie sa uložia v triede

`AgentInformation`, ktorá je schopná ich spracovávať a predávať ďalej. Každý agent si udržuje svoju inštanciu tejto triedy aktuálnu na základe ďalších vnemov. Ďalej si tu agent vytvára štruktúru položiek, ktoré v simulácii existujú. V neposlednom rade sa tu z vnemov udržiava zoznam ponúkaných prác v simulácií.

Správanie `GetMan`

Agenti si z dôvodu optimalizácie zasielania informácií medzi sebou volia akéhosi správcu informácií cez ktorého sa rieši distribúcia vlastností agentov v systéme. Práve výber tohto agenta sa vykonáva v správaní `GetMan`.

V prvom rade sa agent pokúsi zistiť, či v systéme už nejaký takýto správca existuje. V prípade, že áno, agent si nastaví referenciu na identifikátor tohto agenta. Inak si agent zistí pomocou metódy `nameToInt` ohodnotenia mien ostatných agentov v systéme a v prípade, že jeho je najmenšie sa prehlási za správcu. Ak však nie je ohodnotený najlepšie, agent sa začne systému dopytovať na správcu a v krajných prípadoch čaká až desať sekúnd na odpoveď. Po tejto dobe sa rozhodne ukončiť čakanie a zaháji proces odznovu pre prípad, že by sa množina agentov v systéme zmenila a práve on bol najlepšie ohodnotený.

Dôvodom pre zavedenie správcu informácií je, že pri vytváraní koalícií agenti vyžadujú informácie o ostatných agentoch kvôli nevyhnutným výpočtom. V tom prípade by si medzi sebou posielali každý s každým informácie navzájom. Pri tomto návrhu to avšak funguje v pomere $1 : N$. Kde N agentov zasiela svoje informácie jednému správcovi, ktorý ich následne distribuuje. Prvá distribúcia tejto kolekcie nastáva, akonáhle správca obdrží informácie od všetkých agentov, teda počet záznamov v kolekcii sa rovná počtu agentov v systéme. Následne pri každom obdržaní žiadosti o zmenu údajov nejakého agenta.

```
# try searching for manager
agents = searchDF(searchForManager)

# manager found, set local refference
if agents is not Null:
    manager = agents.getFirst()
else:
        agents = searchDF(searchForAll)
        bigger = False

        # search through all agents and compare values
        for agent in agents:
            if(nameToInt(myName) > nameToInt(agent.name)):
                bigger = True
                break

        if not bigger:
            # register as manager
            ...
        else:
            # search for manager agent until found (10s max)
            ...
```

Výpis 7.2: Výber správcu informácií

Správanie `SendAgentInformation`

Toto správanie je vykonávané len pri agentoch, ktorí nie sú správcami. Zabezpečuje odoslanie informácií o agentovi smerom ku správcovi, ktorý ich následne spracuje a uchová v patričnej štruktúre. Toto správanie však musí čakať, pokým sa agentovi nepodarí načítať počiatočné vnemy od simulačného servera. Teda sa neukončuje až do momentu odoslania správy. Znovu sa používa až v prípade načítania nových vlastností agenta.

Na ovládanie behu správania sa používa metóda `done`, ktorú sme spomínali na začiatku tejto sekcie 7. Za pomoci atribútu triedy `SendAgentInformation` sa riadi ukončenie behu tohto správania.

Správania `Send` a `RequestAgentCollection`

Správania `SendAgentCollection` a `RequestAgentCollection` sú základom distribúcie informácií o agentoch.

Agent na začiatku svojho životného cyklu potrebuje pre svoje rozhodnutia poznať ostatných agentov prítomných na platforme. Tieto informácie nadobudne požiadaním správcu. Proces prebieha v podobe vytvorenia správy pre správcu informácií, ktorej obsahom je aktuálny stav agentových znalostí o okolitých agentoch.

Správca je na opačnú stranu zodpovedný za udržovanie aktuálnosti informácií a ich distribúciu. Preto je trieda `SendAgentCollection` zodpovedná za prípadnú distribúciu týchto informácií agentom, ktorý o ne požiadali. Je však dôležité, aby správca neposkytol informácie okamžite, ale až po ich zozbieraní o všetkých agentoch prítomných v systéme.

Správanie `GenerateCombinations`

Vytváranie kombinácií je začiatkom tvorenia koalícií. Každý agent si potrebuje vytvoriť zoznam všetkých možných koalícií, ktoré je potrebné prepočítať. Požadované je určenie obmedzenia veľkosti koalície na k agentov, ktoré sa špecifikuje pri vytváraní triedy.

Na vytvorenie všetkých potenciálnych kombinácií využívame bitové polia, v ktorých každý záznam reprezentuje kokrétneho agenta. Ďalej sa nad touto dátovou štruktúrou používajú binárne operácie, čo zaručuje vysokú efektivitu pri vytváraní týchto kombinácií. Algoritmus avšak nie je optimálny, pretože prehľadáva celý stavový priestor.

Pri vytvorení potenciálnej množiny agentov sa pred jej zaradením do agentovej štruktúry kontroluje, či sú splnené nasledujúce podmienky:

- Agent musí byť súčasťou danej množiny.
- Množina musí byť menšia, pri najhoršom rovná hodnote k.
- Množina musí súčastne obsahovať aspoň 2 prvky.

Správanie `CommitToCalculation`

Agenti sa zaväzujú k výpočtom koaličných hodnôt z dôvodu rozdelenia vypočtov medzi viacerých agentov. Týmto spôsobom nikdy nezostane agent, ktorý by len čakal. To znamená, že celá výpočtová záťaž nezáleží len od jedného agenta.

Počas iteratívneho procesu sa agenti medzi sebou dohadujú, kto bude počítať ktorú množinu koalícií. Keďže sa koalícia skladá z viacerých agentov, pričom každý z nich je odhodlaný pre túto koalíciu prepočítavať jej hodnotu. Je nevyhnutné, aby agenti medzi

sebou oznamovali jednotlivé koalície, ku ktorým sa viažu, aby neboli zbytočne prepočítavané viacerými agentmi.

Vždy pred zaviazaním sa agent skontroluje, či nemá vo fronte správ doručenú správu o zaviazaní sa iného agenta k výpočtu nejakej z jeho koalícií. V tom prípade odstráni všetky takto oznámené koalície a pokračuje ďalej vo výpočte.

Agent opúšťa tento proces buď v prípade, že počet potenciálnych koalícií, ku ktorým by sa mohol zaviazať klesne na nulu alebo v prípade, že počet koalícií ku ktorým už je zaviazaný je viac, ako počet všetkých možných koalícií delený počtom všetkých agentov.

```
# check messages
if getMessageQueueSize() > 0:
    msg = recieve(msgTemplate)

    #verify message contents
    if msg.getContent() instanceof Coalition:
        potentialCoalitions.remove(msgContent)

# otherwise continue committing
if potentialCoalitions.size() > 0:
    # create message for all agents in choosen coalition
    msg.setSender(this)

    for agent in coalition:
        msg.addReceiver(aget)

# send the message
send(msg)

# add coalition to list of commited coalitions
commitedCoalitions.add(coalition)
```

Výpis 7.3: Zaviazanie sa k výpočtom

Správanie `CoalitionValueCalculation`

Ohodnotenie všetkých koalícií, ku ktorému sa agent zaviazal je dôležitým krokom pri ich vytváraní. Ku každej z nich sa vytvára zoznam očakávaných výsledkov. Vzťah koalície s agentom sa ukladá ako záznam v tomto zozname, ktorý reprezentuje efektívnosť pri vykonávaní práce danou koalíciou.

Ohodnotenie koalície je maximálna hodnota vzhľadom na všetky práce v systéme. To znamená, že sa zo všetkých hodnôt, ktoré sú vypočítané nasledovným vzťahom:

$$value = 100 - (coalitionSize - roles) * 4 - (jobCapacity - coalitionCapacity)/4 \quad (7.1)$$

vyberie maximálna a táto hodnota sa použije ako ohodnotenie koalície.

Súčasne sa vytvára vektor očakávaných výsledkov koalície a na základe neho sa dá následne v systéme dohľadať práca, pre ktorú bolo vypočítané maximálne ohodnotenie.

Správanie `AnnouceCoalitionalWeight`

Keďže je potrebné vybrať najvhodnejšiu koalíciu agenti sa po ukončení výpočtov propagovať svoj najlepší výsledok ostatným agentom. Takýmto spôsobom si medzi sebou agenti oznámia ohodnotenia koalícií, na základe ktorých sa vyberie koalícia na zostavenie.

Pri prijatí správy s vypočítanou hodnotou koalície si agent túto koalíciu uloží a následne čaká na ostatných agentov. Taktiež v prípade, že on sám vypočíta hodnotu si ju ukladá. Avšak v prípade, že agent nemá žiadnu hodnotu, ktorú by ďalej oznámil, posiela ako hodnotu -1. Táto činnosť priebeha hlavne zo synchronizačných dôvodov.

Správanie `CreateCoalition`

Pri vytváraní koalícií má agent zostavený zoznam oznámených koaličných hodnôt, z ktorých vyberá hodnotu, ktorá najviac pomeru ceny k veľkosti koalície. Každý agent si vyberie práve tú koalíciu, v ktorej je tento pomer minimalizovaný. Tento výber skončí u každého agenta rovnako.

Rozdielom však bude, či sa agent v konečnej koalícií vyskytuje alebo nie. Ďalšie akcie, ktoré ovplyvnia jeho vnútorný stav určuje fakt, či agent musí pokračovať v koaličnom procese alebo koalíciu vytvára. Na základe vybranej koalície a jej ohodnotenia sa následne vyberá práca, ktorú má koalícia vykonať. Keďže ohodnotenie koalície bolo počítané v závislosti na špecifikáciách práce, sa jednoduchým hľadaním cez indexy dá konkrétna práca vyhľadať.

Kapitola 8

Výsledky

V tejto kapitole sú popísané výsleky, ktoré algoritmus dosiahol pri testovaní na agentnej platforme implementovanej v prostredí Jade založenom na programovacom jazyku Java. Využíval rozhranie `eis` pre pripojenie ku simulačnému serveru.

Dôležitými aspektami algoritmu boli:

- **Čas**, za ktorý je algoritmus schopný rozdeliť agentov do koalícií.
- **Počet správ** zaslaných medzi agentmi k vytvoreniu koalícií.

Keďže komunikácia so simulačným serverom nebola podstatou tejto práce, je v časových údajoch vynechaná zložka týkajúca sa registrácie agenta so serverom. Táto zložka trvala približne od jednej do dvoch sekúnd naprieč všetkými konfiguráciami. Na druhú stranu, získavanie a spracovanie vnemov od simulačného serveru považujem za dôležitú súčasť procesu ustanovenia koalícií.

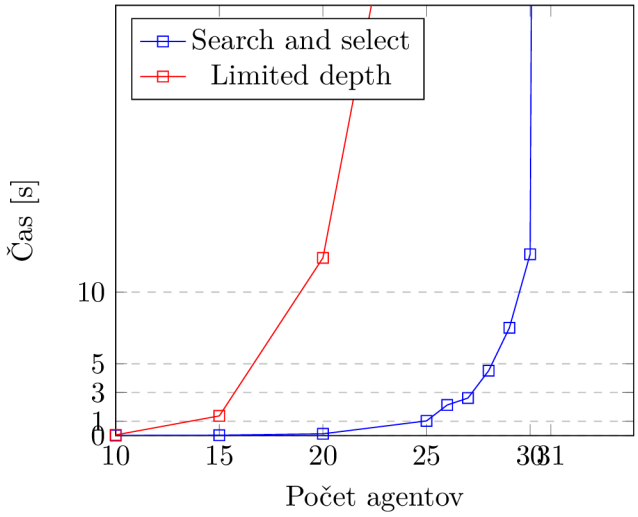
8.1 Prehľadávanie stavového priestoru

S nárastom agentov v systéme nastáva problém s prehľadávaním stavového priestoru všetkých koalícií. Počet všetkých kombinácií agentov na vytvorenie koalície je 2^n, pričom n je počet agentov. Exponenciálne nárastu vo výpočtoch sú nevhodné z dôvodu obrovského nárasty potrebných zdrojov pri pridaní čo i len jedného prvku.

Algoritmus na prehľadávanie stavového priestoru ktorý je použitý v tomto projekte je schopný skončiť v rozumnom čase až do veľkosti tridsiatich agentov. Vzťah medzi počtom agentov a časovou náročnosťou môžeme vidieť v grafe 8.2.

V grafe je jasne vidieť exponenciálny nárast zložitosti výpočtov. Na začiatku pri testovaní s desiatimi agentmi bol výpočtový čas triviálny. Zvyšovaním počtu agentov sme sa dostali až do bodu, kde zhodnotenie stavového priestoru trvalo vyše desiatich minút. Napriek využitiu bitových operácií v algoritme bolo čisté množstvo stavov neúnosné. Všetky časy v grafe sú uvedené jednotkovo, teda samostatne na agenta.

Časová náročnosť prehľadávania stavového priestoru



Riešením komplexnosti stavového priestoru by teoreticky mohol byť algoritmus, ktorý by sa zanáral len do hĺbky k. Avšak žiadna z implementácií obdobného algoritmu nedosahovala porovnateľné výsledky s implementovaným algoritmom.

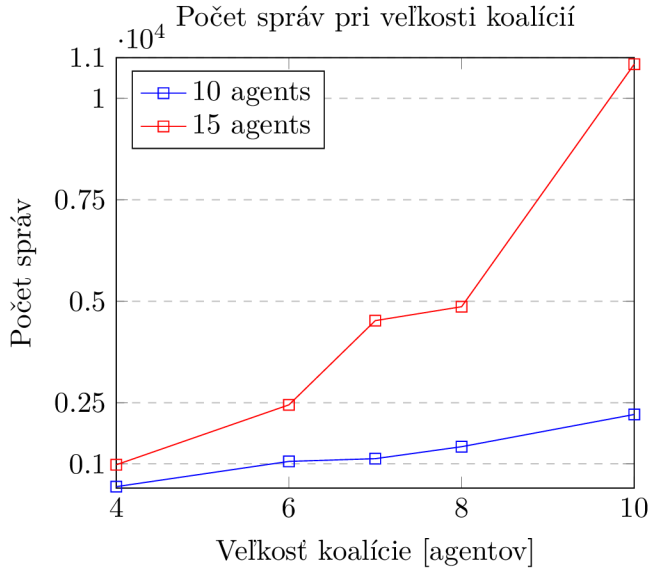
Táto náročnosť vzniká pri použitých dátových štruktúrach. V prípade prehľadávania do obmedzenej hĺbky stavový priestor obsahuje menej prípadov. Náklady na udržanie tohto stavu sú však vyššie.

8.2 Dopad veľkosti koalícií na množstvo správ

Množstvo správ je priamo úmerné množstvu komunikujúcich agentov. Pričom výsledky merania ukazujú postupný nárast správ v systéme.

Tieto výsledky vyplývajú hlavne z princípu dohadovania sa o koaličných hodnotách. Kde pri väčšom množstve agentov v koalícií sa súčasne zvyšuje počat agentov, ktorí sa budú vzájomne dohadovať o zaviazaní sa k výpočtom. Pričom pri malých koalíciach je celková množina koalícií, o ktorých sa dohadujú súčasne menšia.

Na základe týchto poznatkov si dovoľujem tvrdiť, že veľkosť koalícií má byť nastavené s ohľadom na efektivitu prevedenia úloh systému a nie z pohľadu efektívnosti komunikácie medzi agentmi. Nárast správ medzi agentmi nie je exponenciálny. Väčším problémom pre systém je zvyšovanie počtu agentov.


Počet správ pri veľkosti koalícií
$\cdot 10^4$
1.1
1
0.75
0.5
0.25
0.1
Počet správ
10 agents
15 agents
4
6
8
10
Veľkosť koalície [agentov]

Kapitola 9

Záver

Hlavným cieľom tejto práce bolo dosiahnuť úspešné zadelenie agentov do koalícií na základe ponúknutých prác od simulačného serveru.

Tento cieľ sme dosiahli použitím algoritmu Shehory-Kraus, ktorý je popísaný v kapitole 6.2. Prvým dôležitým krokom pri implementácii tohto algoritmu bolo navrhnúť algoritmus, ktorý vytvorí množinu potenciálnych koalícií, z ktorých si agenti následne budú vyberať a zaväzovať sa k výpočtu koaličných hodnôt. Keďže stavový priestor všetkých kombinácií môžeme reprezentovať ako 2^n, bolo naozaj kľúčové dbať na efektivitu a vyhnúť sa rekurzii.

Ďalej bolo treba vytvoriť návrh komunikácie medzi agentmi kde sme využívali typ správy ako hlavný separátor. V ďalšej úrovni sa správy delili na základe obsahu, kedy samotný obsah reprezentoval už konkrétnu správu buď informačného alebo iného charakteru.

V neposlednom rade bolo potrebné vytvoriť rovnicu na ohodnocovanie koalícií v závislosti k jednotlivým prácam z pohľadu, ako keby ich vykonávali. Toto číslo určuje koaličnú hodnotu a preto bolo dôležité, aby sme naozaj jeho maximalizáciou dosahovali zvyšovanie efektivity výkonov práce.

Z pohľadu budúcnosti vývoju projektu by bolo vhodné optimalizovať algoritmus na vytváranie kombinácií agentov. Keďže momentálne je prehľadávaný celý stavový priestor agenta, v ideálnom prípade by sa prehľadával do hĺbky k, avšak s lepšou efektívnosťou. Ďalej by sa dalo zamerať na zlepšenie komunikácie medzi agentmi, keďže momentálne sa zasielajú celé serializované objekty. Tento prístup začína pri veľkom množstve agentov zlyhávať. Jednou z možností by bola implementácia indexov agentov, z dôvodu prenosu koaličných objektov medzi agentmi pri dohadovaní sa o zaviazanie k výpočtom.

Literatúra

[1] *Jade site / Introduction to JADE.* [Online; navštíveno 02.05.2019]. URL `https://jade.tilab.com/documentation/tutorials-guides/introduction-to-jade/`

[2] *Jade site / Java Agent DEvelopment Framework.* [Online; navštíveno 02.05.2019]. URL `https://jade.tilab.com/`

[3] *Jade site / Technical description.* [Online; navštíveno 02.05.2019]. URL `https://jade.tilab.com/technical-description/`

[4] Bellifemine, F.; Caire, G.; Greenwood, D.: *Developing Multi-Agent Systems with JADE.* John Wiley & Sons Ltd, 2007, ISBN 978-0-470-05747-6.

[5] Bellifemine, F.; Poggi, A.; Rimassa, G.: Developing multi-agent systems with JADE. 2001, ISBN 3540424229, ISSN 03029743, s. 89–103.

[6] Durfee, E.; Rosenschein, J.: Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples. 08 1995: str. 12.

[7] Hindriks, K.; S. de Boer, F.; Hoek, W.; aj.: Formal semantics for an abstract agent programming language. 10 1998, s. 215–229, doi:10.1007/BFb0026761.

[8] Jennings, N. R.: Coordination techniques for distributed artificial intelligence. 1996.

[9] Parasumanna Gokulan, B.; Srinivasan, D.: *An Introduction to Multi-Agent Systems*, ročník 310. 07 2010, s. 1–27, doi:10.1007/978-3-642-14435-6_1.

[10] Russel, S. J.; Norvig, P.: *Artificial Intelligence: A Modern Approach.* Prentice Hall, tretie vydanie, 2010, ISBN 0-13-604259-7.

[11] Shehory, O.; Kraus, S.: Methods for Task Allocation via Agent Coalition Formation. *Artificial Intelligence*, ročník 101, 05 1998: s. 165–200, doi:10.1016/S0004-3702(98)00045-9.

[12] Wooldridge, M.: Reasoning About Rational Agents. *J. Artificial Societies and Social Simulation*, ročník 5, 01 2002.

[13] Wooldridge, M.; Jennings, N.: Intelligent agents: theory and practice. *The Knowledge Engineering Review*, ročník 10, č. 2, June 1994.