UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

DECENTRALIZOVANÁ A DISTRIBUOVANÁ KOMUNIKAČNÁ PLATFORMA

Bakalárska práca

Študijný program: Aplikovaná informatika
Študijný odbor: 2511 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: RNDr. Jozef Šiška, PhD.

Bratislava **Adam Horváth**

Čestne vyhlasujem, že bakalársku prácu som vypracoval samostatne s použitím uvedenej literatúry a pod dohľadom môjho školiteľa.

.....

V Bratislave dňa 29.5.2014

Ďakujem môjmu školiteľovi RNDr. Jozefovi Šiškovi PhD. za jeho obrovskú ochotu, pomoc, cenné rady a nesmiernu trpezlivosť počas písania tejto bakalárskej práce.

Abstrakt

LCP je platforma, ktorá zabezpečuje komunikáciu agentov na lokálnej sieti a cieľom tejto práce je rozšíriť ju o možnosť vzájomného objavenia agentov, ktorí sa nachádzajú na rôznych lokálnych sieťach a možnosť posielať si medzi sebou správy. Toto chceme dosiahnuť pomocou takzvaného *Gateway Agenta*, ktorý bude preposielať správy od svojich lokálnych agentov smerom k ich príjemcom na vzdialenej lokálnej sieti. Pre tieto účely v práci implementujeme komponenty *Discovery Service* slúžiaci na objavovanie agentov v systéme a *Message Transport Service* zodpovedný za posielanie správ medzi hocijakými dvomi agentami v systéme.

Kľúčové slová:

Multiagentové systémy, komunikačná platforma, LCP, Gateway Agent

Abstract

The main goal of this work is to expand LCP by allowing mutual discovery of agents in different LANs and allowing them to send messages to each other. We want to accomplish this by the means of a *Gateway Agent* which will resend messages from its local agents towards the recipients of the messages on remote LANs. For this purpose, we will implement components *Discovery Service*, which will handle agent discovery in the system, and *Message Transport Service* responsible for sending and receiving messages among any two agents in the system.

Keywords:

Multiagent Systems, communication platform, LCP, Gateway Agent

Obsah

Úvod

Napriek tomu, že v súčasnosti existujú multiagentové systémy viac-menej v pozadí verejného záujmu, v praktickom využití nachádzajú svoje miesto. Spomenúť môžeme napríklad tému na vzostupe - „inteligentné domy“, kde navzájom prepojené agenty starajúce sa o rôzne súčasti domu sú schopné konať na základe stavu podmienok vo svojom prostredí.

Objektom záujmu tejto práce bude decentralizovanosť týchto systémov. Od platformy LCP sa očakáva, že bude multiagentová a taktiež má existovať bez centrálnej autority, v tomto prípade servera. Pre našu komunikačnú platformu to znamená, že má byť schopná nakonfigurovať sa sama od seba.

Cieľ práce

Cieľom tejto práce je rozšírenie Jednoduchej Komunikačnej Platformy (ďalej LCP, podľa „Lightweight Communication Platform“) o možnosť komunikácie agentov v jednej lokálnej sieti s agentami v inej lokálnej sieti. Východiskovým riešením tohto problému je takzvaný „Gateway Agent“, ktorý je schopný posielať správy aj za iných agentov v jeho lokálnej sieti a prakticky sa správa ako virtuálny router.

Štruktúra práce

Táto práca je rozdelená do piatich kapitol. Prvá kapitola bude obsahovať prehľad o agentoch a multiagentových systémoch. Pri týchto systémoch spomenieme aj počítačové siete, routovanie v nich a technológie, ktoré možno použiť na implementáciu nášho riešenia. Na konci kapitoly uvedieme komunikačnú platformu LCP. V druhej kapitole špecifikujeme požiadavky na funkčnosť komponentov nášho riešenia. Tretia kapitola bude návrh nášho riešenia. V nej navrhneme jednotlivé funkcie komponentov, celkovú funkčnosť riešenia a možné problémy plynúce z návrhu. V ďalšej kapitole – implementácia popíšeme aké technológie sme v riešení použili a ako fungujú jednotlivé komponenty. V záverečnej kapitole zhrnieme čo bolo cieľom práce, aké problémy neboli vyriešené a ponúkneme možné rozšírenia nášho systému.

1 Prehľad

Aby sme dokázali plne pochopiť, čo chceme v tejto práci dosiahnuť a pre celkové porozumenie problematiky, sa budeme v tejto časti práce venovať prehľadu poznatkov z oblasti agentov a multiagentových systémov. Najskôr si zadefinujeme pojem agent a vysvetlíme teoretické základy multiagentových systémov a ich potenciál, ktorý sa budeme snažiť načrtnúť na príkladoch ich praktického využitia vo svete.

1.1 Inteligentný agent

Klasickú definíciu agenta nám ponúkajú Russel a Norvig [1]:

„Agent je všetko, na čo sa dá pozerať ako na niečo, čo vníma svoje okolie senzormi a reaguje na toto okolie pomocou aktuátorov.“

Agent je podľa nich zložený z architektúry a agentového programu, ktorého vytvorenie je úlohou práve umelej inteligencie. Fyzická architektúra nás v tejto práci nebude zaujímať, preto ďalej v tejto kapitole budeme rozumieť pod pojmom agent práve agentový program. Existujú agenty, ktoré sú veľmi jednoduché, ale aj také, ktoré sú komplexné. Podľa ich vnemovej inteligencie a schopnosti ich opäť Russel a Norvig rozdelili do týchto piatich kategórií, ktoré vzápätí rozoberieme podrobnejšie:

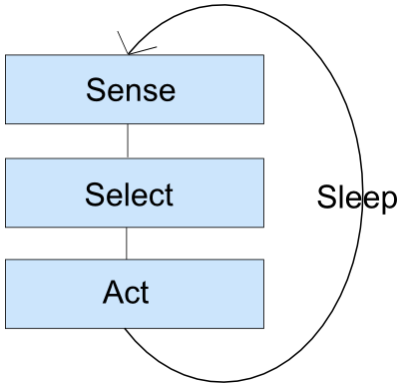
- ***Simple reflex agent***
- ***Model-based agent***
- ***Goal-based agent***
- ***Utility-based agent***
- ***Learning Agent***

1.1.1 Simple Reflex Agent

Tento druh agenta je úplne najjednoduchší. Reaguje totiž len na aktuálny stav jeho prostredia a pritom si neuchováva žiadnu históriu stavov daného prostredia. Tento agent je plne úspešný iba v úplne pozorovateľnom prostredí. Keď sa jedná o čiastočne

pozorovateľné prostredie, vo väčšine prípadov je tento agent v nekonečnom cykle a jeho správanie je zväčša podmienené. Ak je splnená podmienka, tak na ňu programovo reaguje (Obrázok č. 1) .

Obrázok č. 1 Diagram cyklu jednoduchého reflexného agenta v čiastočne pozorovateľnom prostredí



1.1.2 Model-based Agent

Na rozdiel od jednoduchého reflexného agenta je v tomto type agenta uložená štruktúra, ktorá má reprezentovať súčasný stav jeho sveta. Ten je ovplyvnený históriou vnímania. Podľa zmien je schopný sa naučiť „ako svet funguje“, a teda je schopný čiastočne zachytiť aj stránky sveta, ktoré nie je schopný vnímať. Táto skutočnosť má za následok, že *model-based* agent je úspešný aj v čiastočne pozorovateľnom prostredí. Svoje reakcie na stav sveta vyberá ako jednoduchý reflexný agent.

1.1.3 Goal-based Agent

Jeho základ je v podstate rovnaký ako základ *model-based* agenta, je však rozšírený o informáciu o jeho cieli. Táto informácia v sebe zahŕňa popis požadovaných situácií a stavu sveta. Vďaka tomu je *goal-based* agent schopný z viacerých možností vybrať takú, ktorá ho dostane alebo priblíži k požadovanému stavu. Ako vybrať správnu reakciu je otázka plánovania a prehľadávania. Sú to poddisciplíny umelej inteligencie. Niekedy je tento agent menej efektívny, ale za to je viac flexibilný, pretože jeho cieľ je udaný explicitne a dá sa meniť.

1.1.4 Utility-based Agent

Na rozdiel od *goal-based* agenta, ktorý je schopný rozoznávať iba medzi stavom, ktorý je požadovaný a ktorý nie je, je *utility-based* agent schopný určiť, nakoľko sú

jednotlivé stavy požadované pomocou *úžitkovej funkcie.* Táto funkcia mapuje stav na veľkosť úžitku tohto stavu. Agent zhodnocuje možné stavy podľa toho, koľko ťažkostí (alebo úžitku) mu daný možný stav prinesie. Takéto zhodnocovanie mu pomáha vybrať najlepšiu postupnosť činností, keďže racionálny agent sa snaží o to, aby dosiahol čo najviac úžitku a teda bol čo najviac „šťastný (spokojný)".

1.1.5 Learning Agent

Jeho výhodou je, že je schopný fungovať aj v neznámom prostredí. Čím dlhšie sa v prostredí nachádza, tým je schopnejší vyberať lepšie rozhodnutia, pretože sa učí. Skladá sa z troch častí:

- **Učenlivý element** – je zodpovedný za zlepšovanie sa agenta. Odozvu na jeho činy, ktorú získa z prostredia posúva do výkonnostného elementu, aby mu pomohol v budúcnosti konať lepšie rozhodnutia.

- **Výkonnostný element** – má za úlohu vybrať najlepšiu akciu

- **Generátor problémov** – navrhuje akcie, ktoré by mohli viesť k novým skúsenostiam a teda novým poznatkom

1.2 Teoretické základy multiagentových systémov

Aj keď je agent sám o sebe schopný riešiť určité problémy, môže sa stať, že narazí na problém, na ktorého vyriešenie nemá dostatočné prostriedky. Práve preto vznikla nutnosť existencie multiagentových systémov. Sú to také systémy, ktoré v sebe zahŕňajú viacero agentov, ktoré navzájom spolupracujú a ich prostredie. Katia P. Sycara [2] popísala ich charakter takto:

1. Každý agent má neúplné informácie alebo nedostatočné schopnosti vyriešiť problém a teda má naň limitovaný pohľad
2. Neexistuje nad nimi žiadna globálna kontrola.
3. Dáta sú decentralizované
4. Výpočty sú asynchrónne

1.2.1 Typy multiagentových systémov

Klasifikácií multiagentových systémov existuje mnoho. Môžeme ich deliť podľa typu prostredia, v ktorom sa agenty nachádzajú, podľa komplexnosti alebo akého typu sú, podľa komunikácie medzi nimi a tak ďalej.

1.2.1.1 Typy multiagentových systémov podľa vzájomnej komunikácie

Je možné rozdeliť multiagentové systémy podľa toho, akým spôsobom agenty medzi sebou komunikujú. Existujú tri typy multiagentových systémov podľa ich vzájomnej komunikácie a to [5]:

- **Bez priamej komunikácie** – tento typ komunikácie môže byť realizovaný napríklad pomocou prostredia
- **Jednoduchá komunikácia** – väčšinou je jednosmerná aj keď kontaktovaný agent môže reagovať na požiadavku
- **Komplexná komunikácia –** taktiež ju môžeme nazývať podmienková, príkladom môže byť „Pohnem sa, ak sa pohneš ty“

1.2.1.2 Typy multiagentových systémov podľa prostredia

V tejto podkapitole popíšeme typy prostredí, v ktorých sa agentyi môžu nachádzať. Toto rozdelenie bolo navrhnuté Russelom a Norvigom [1]:

- **Plne pozorovateľné a čiastočne pozorovateľnému** – Ak v hocijakom čase môže agent získať celkový stav svojho prostredia, hovoríme o takom prostredí ako o plne pozorovateľnom. Úlohové prostredie je plne pozorovateľné, pokiaľ dokáže agent pomocou svojich senzorov zachytiť všetky informácie, ktoré sú **relevantné** pre jeho rozhodnutie. Čiastočne pozorovateľné prostredie môže byť vtedy, keď je v prostredí vysoký šum, nepresné senzory a teda senzory nedokážu zachytiť presné informácie alebo jednoducho dáta zo senzoru chýbajú.

- **Deterministické a stochastické prostredie –** Deterministické prostredie je také prostredie, v ktorom hocijaký čin má jediný možný výsledok. Naproti tomu v stochastickom prostredí si nemôžeme byť istí, aký účinok náš čin vyvolá.

- **Statické a dynamické prostredie –** Ak prostredie dokáže byť ovplyvnené iba činmi agenta a inak zostane nezmenené, také prostredie nazývame statické. Dynamické prostredie sa však mení odhliadnuc od činov agenta.

- **Diskrétne a spojité prostredie –** Napríklad hra šachu má konečný počet pozorovateľných stavov a vnemov. Takémuto prostrediu hovoríme diskrétne. Prostrediu hovoríme spojité, ak stavy a vnemy závisia napríklad od času alebo polohy agenta. Príkladom môže byť automatizovaný šofér.

- **Známe a neznáme –** Nejde tu ani tak o prostredie samotné ako o to či agent alebo jeho tvorca vie, ako prostredie funguje. Ak nevie, ide o neznáme prostredie a agent sa musí zákonitosti a fungovanie takéhoto prostredia naučiť. Agent na druhej strane môže vedieť ako prostredie funguje, neznamená to však, že prostredie je plne pozorovateľné. Ako príklad môžeme uviesť hru hľadania mín. Vieme zákonitosti a pravidlá hry, avšak nevieme, kde sa míny nachádzajú. Takéto prostredie je potom čiastočne pozorovateľné.

1.2.2 Komunikácia v multiagentových systémoch

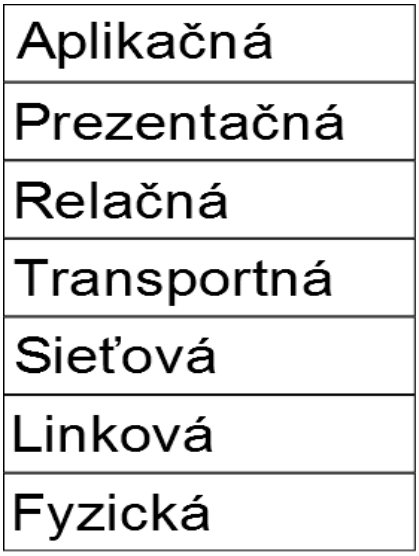
Komunikácia agentov je jednou z najzákladnejších podmienok v multiagentových systémoch. Je esenciálna k tomu, aby si agenty vedeli vymieňať informácie, koordinovať svoje úlohy a takýmto spôsobom spolupracovať na dosiahnutí ich cieľa. Ak by komunikácie neboli schopní, stratil by sa celý zmysel multiagentového systému. Jazyk, ktorým sa agenty dorozumievajú sa nazýva „Agent Communication Language“, skrátene ACL. Dva najpoužívanejšie agentové jazyky sú FIPA-ACL [3], a KQML [4]. Oba jazyky boli inšpirované teóriou rečových aktov.

1.2.3 Multiagentové systémy a počítačové siete

Od multiagentových systémov zvyčajne očakávame, že agenty v rámci systému neexistujú na jednom zariadení, ale vo viacerých. Od týchto zariadení už principiálne vyžadujeme, aby boli vzájomne prepojené. Väčšinou sú pripojené do lokálnej siete (od „Local Area Network“) alebo do rozsiahlej siete WAN („Wide Area Network“). Aby však agenty boli schopné po takejto sieti komunikovať, musia byť špecificky naprogramované. Tomuto programovaniu sa tiež hovorí sieťové programovanie.

Jednotlivá funkčnosť siete je rozdelená do sieťových vrstiev v OSI modeli Open System Interconnection) špecifikovaným organizáciou ISO (International Organization for Standardization). Tento model obsahuje 7 vrstiev (Obrázok č. 2) a každá z nich využíva vrstvu pod ňou a slúži vrstve nad ňou. Protokoly na rovnakej vrstve sú schopné spolu komunikovať [6].

Obrázok č. 2 Vrstvy OSI modelu



1.2.3.1 Routovanie

Routovanie je určovanie cesty paketov v sieti. Zaoberá sa ním tretia, sieťová, vrstva. Existujú mnohé spôsoby ako určiť cestu, ktorou bude paket poslaný. Môže byť vybraná staticky podľa routovacích tabuliek, vopred dohodnutá v rámci jedného dialógu alebo určená dynamicky. V súčasnom internete sa používa prevažne dynamické routovanie. Príkladmi protokolov takéhoto routovania je napríklad RIP [7] a OSPF [8].

V dynamickom routovaní, taktiež nazývanom aj adaptatívne, je dôležitá takzvaná routovacia metrika. Je to hodnota, niekedy jej hovoríme aj cena cesty, ktorá je rozhodujúcim faktorom v rozhodovaní o výbere najlepšej cesty.

Metrika môže byť [9]:

- **Aditívna –** celková cena cesty je sumou cien jednotlivých úsekov celej cesty
- **Konkávna –** celková cena cesty je minimom cien jednotlivých úsekov celej cesty

- **Multiplikatívna** – celková cena cesty je súčinom cien jednotlivých úsekov celej cesty

Celková hodnota metriky je určená viacerými aspektmi, napríklad počtom hopov, rýchlosťou prenosu, odozvou, stratovosťou paketov cesty alebo aj MTU. Spomínaný RIP patrí do rodiny *distance-vector* protokolov. Tie využívajú rôzne algoritmy pre výpočet najvhodnejšej cesty. Ukladajú si minimálne vzdialenosti do všetkých uzlov v sieti. Vzdialenosťou sa nemyslí fyzická vzdialenosť, ale cena za dosiahnutie určitého uzla. Z toho vyplýva, že čím menšia cena, tým menšia vzdialenosť.

1.3 Prehľad technológií

Táto sekcia sa zaoberá technológiami, ktoré možno využiť pri riešení našej úlohy. Ponúkajú sieťové komponenty a mechanizmy, ktoré uľahčujú sieťové programovanie, ktoré budeme využívať v našej práci. V krátkosti popíšeme frameworky Qt a Tufão.

1.3.1 Qt

Qt je využiteľné v grafických, ale aj konzolových aplikáciách. Tento framework je cross-platformový [14] a ponúka funkcionalitu, ktorá dokáže uľahčiť programovanie inak zdĺhavých sieťových metód. Jeho jednoduché použitie pre pripojenie na multicastovú skupinu, posielanie a prijímanie datagramov a HTTP requestov z neho robí dokonalého kandidáta aj kvôli asynchrónnosti jeho sieťového API. Bohužiaľ, priamo neimplementuje funkcionalitu HTTP servera.

Ďalšou výhodou Qt je používanie signal-slot mechanizmu, ktorý je jednou z jeho hlavných čŕt. Slúži na komunikáciu medzi objektmi. Použitie je veľmi jednoduché. Pripojíme signál jedného objektu na slot toho istého alebo iného objektu. Pokiaľ objekt tento signál vyšle, príslušný objekt ho spracuje svojím slotom, kde slot je normálna funkcia C++, ktorá sa ale navyše dá pripojiť na signál. Aj toto si vyžadovalo vznik takzvaného *Meta-Object Compiler*-u, v skratke *moc*. Ten spracováva hlavičkové súbory C++ a keď nájde triedu, ktorá obsahuje makro Q_Object, vytvorí pre ňu *meta-object* kód.

1.3.2 Tufão

Tufão je ďalší cross-platformový framework, ktorý implementuje HTTP server, jednoduché spracovávanie požiadaviek a automatizovanie niektorých častí

odpovedí servera pomocou signal-slot mechanizmu Qt a implementácie HTTP request a HTTP response. Výrazným plusom tohto frameworku je aj jeho prehľadná dokumentácia [15].

1.4 Existujúce návrhy multiagentových systémov

V súčasnosti existujú mnohé návrhy využitia multiagentových systémov. Je ich možno využiť v širokom spektre odvetví od turizmu, cestnej premávky a armády až po medicínu.

- **PalliaSys [10]–** Návrh multiagentového systému, ktorý má pomôcť pri paliatívnej starostlivosti tým, že pomocou agentov bude monitorovať stav pacienta a bude schopný ponúknuť detailné, aktuálne informácie o pacientovi doktorom.

- **MokSAF [17] –** Systém, ktorý má pomôcť armáde pri kritickom rozhodovaní a ponúka virtuálne prostredie na plánovanie a koordináciu cesty. Spolupracujú v ňom dva typy agentov – *Path Planner* agent, ktorý vedie ľudí po ceste, ktorú vyhodnotí ako najkratšiu a berie do úvahy iba fyzické prekážky, ktoré môžu byť zadané aj manuálne do mapy. *Critique Agent* analyzuje nakreslenú cestu a pomáha doladiť detaily, berie pritom do úvahy fyzikálne, ekonomické a sociálne aspekty. Na rozdiel od *Path Planner* agenta nevytvára cestu automaticky, ale oznámi, či daná cesta je možná.

- **MAS/LUCC [18] –** je označenie pre multiagentové systémy zaoberajúce sa modelovaním využitia pôdy a jej nasledovného využitia.

- **Java Agent DEvelopment Framework (JADE) [12] -** Middleware, ktorý zjednodušuje implementáciu multiagentových systémov, ktoré sú kompatibilné s FIPA špecifikáciou. Celý software je naprogramovaný v Jave. Obsahuje grafické nástroje, ktoré pomáhajú najmä v štádiách debuggingu a deplymentu. Tento software je v súčasnosti zadarmo.

- **FIPA-OS [13] -** Prvýkrát vydaný v roku 1999 pre širokú verejnosť *Royalty Free*. Zároveň je prvou Open Source implementáciou FIPA a na jeho vývoji sa podieľa

mnoho vývojárov. Podarilo sa im vydať cez 10 oficiálnych vydaní. Podporuje väčšinu experimentálnych FIPA špecifikácií. Taktiež je implementovaný v Jave a pre jeho návody je vhodný pre začínajúcich vývojárov FIPA kompatibilných systémov.

1.5 LCP

Jednoduchá komunikačná platforma LCP používa na prenos správ medzi agentami REST TCP/IP požiadavky. Sú založené na FIPA ACL. Oproti zvyčajnej FIPA implementácii sú jej cieľom heterogénne systémy obsahujúce viacero jednoduchých agentov, ktoré môžu byť vytvorené v hocijakom jazyku a sú schopné komunikovať bez centrálneho manažmentu [11].

Agenty v takomto systéme sú schopné objaviť ostatných agentov na svojej lokálnej sieti, ako aj identifikovať, aké služby tieto agenty ponúkajú. Umožňuje to existenciu viacerých heterogénnych agentov rozmiestnených po lokálnej sieti.

Všeobecný názov pre komponent, ktorý sa v platforme stará o objavovanie členov systému sa nazýva *Discovery Service*. Myšlienka LCP je navrhnutá tak, aby časom podporovala viacero druhov *Discovery Service* podľa prostredia, v ktorom existujú. Jeho úlohou je taktiež poskytnúť potrebné informácie o agentoch, za ktorých je zodpovedný ostatným *Discovery Service* svojho druhu.

Okrem *Discovery Service* existuje v systéme komponent *Message Transport Service*. Tak ako pri *Discovery Service,* LCP v budúcne bude podporovať viaceré druhy podľa podporovaného protokolu. Stará sa o prijímanie a doručovanie správ medzi agentami.

2 Špecifikácia zadania práce

V tejto kapitole sa budeme venovať funkcionalite, ktorú má LCP ponúkať. Popíšeme hlavné časti LCP – *Discovery Service a Message Transport Service.* Za účelom testovania bude pre nás zaujímavá aj časť *Platform*, ktorá bude zastrešovať tieto dva komponenty. Agenty v LCP majú medzi sebou komunikovať pomocou správ. Pokiaľ sa príjemca nenachádza na rovnakej lokálnej sieti, musí sa správa preposlať cez *Gateway Agenta*, ktorého popis taktiež uvedieme v tejto kapitole. V tejto kapitole uvedieme aj špecifikáciu funkčnosti posielania týchto správ.

2.1 Discovery Service

Súčasťou LCP bude *Discovery Service* (ďalej len DS), ktorý bude zodpovedný za objavovanie agentov, ich služieb a možností ako ich kontaktovať. Okrem objavovania bude mať aj opačnú úlohu, a to poskytovať tieto informácie o agentoch, za ktorých je zodpovedný ostatným DS.

2.2 Message Transport Service

Message Transport Service (ďalej len MTS), bude zodpovedný za odosielanie správ najlepšou cestou pomocou dostupných informácií od DS. Taktiež sa bude starať o preposielanie správ ich príjemcom, pokiaľ sa nenachádza v konečnom uzle trasy, ktorou je správa poslaná.

2.3 Platform

Táto súčasť obsahuje DS aj MTS a správa sa ako rozhranie medzi nimi. V kontexte tejto práce vznikol tento komponent pre potreby testovania a doplnenia funkčnosti z dôvodu budúcej integrácie do iného systému. Jeho úlohou bude spravovať aktuálny zoznam aktívnych agentov ako aj potrebné informácie o nich.

2.4 Preposielanie správ

Od LCP očakávame, že aj agenty zo vzdialených lokálnych sietí, FIPA platforiem alebo rôznych implementácií sa budú schopné objavovať a taktiež si medzi sebou posielať správy cez MTS. Na to potrebujeme router, ktorý bude posielať správy od agentov na svojej lokálnej sieti nejakému vzdialenému routeru. Ten bude schopný prijať takúto správu a odovzdať ju správnemu agentovi na svojej lokálnej sieti. Navyše bude schopný oznamovať ostatným routerom, akých agentov zastupuje a spracovávať takéto informácie od ostatných routerov. Preto v našom systéme vznikol pojem *Gateway (GW) Agent*. Virtuálne plní úlohu spomínaného routera v sieti. Na jednej lokálnej sieti môže byť viac ako jeden a musí byť aspoň jeden GW agent. Aby bol GW agent schopný fungovať podľa našich požiadaviek, musí mať verejnú IP. Taktiež si bude ukladať perzistentný zoznam ostatných aktívnych GW agentov ako aj zoznam všetkých GW agentov podľa stanovených kritérií z toho dôvodu, aby ich bol schopný kontaktovať aj po jeho vypnutí a premiestnení na inú lokálnu sieť.

Dôležitá poznámka: Aby sa predišlo nedorozumeniam, je nutné napísať, že Gateway Agent nie je správne pomenovanie v kontexte tejto práce. Názov pochádza ešte z pôvodného návrhu systému a preto je tento pojem viac menej historický. V našom systéme bude úlohy GW agenta plniť platforma. Preto ju budeme pri tejto príležitosti nazývať aj GW platforma.

3 Návrh riešenia

V nasledovných sekciách popíšeme spôsob, akým by mali fungovať komponenty, *Discovery Service, Message Transport Service* a *Platform.* Taktiež tu navrhneme štruktúry, s ktorými budú tieto komponenty pracovať.

3.1 Transport Address

Transport adresa je spôsob, akým sa budú agenty medzi sebou kontaktovať. Je to adresa, na ktorú sa budú posielať správy agentom. Bude definovaná pomocou URL. Pre potreby udržiavania aktuálnych informácií bude obsahovať aj informáciu o tom, dokedy je platná. Za účelom vyberania vhodnej cesty bude obsahovať aj informáciu o metrike danej adresy. Pre účely routovania bola k transport adrese pridaná aj položka *origins* a *sourceDs.*

3.2 Agent

Každý agent bude jednoznačne definovaný menom. Musí byť v systéme unikátne. Ďalej bude obsahovať informácie o službách (*services*), ktoré ponúka a jeho príznakoch (*flags*). Tieto dve informácie v našom systéme budeme oznamovať, ale nemajú žiadny dopad na funkčnosť našich komponentov. V neposlednom rade si v agentovi budeme udržiavať informácie o transport adresách, ktorými bude dosiahnuteľný.

3.3 Správy

V našom systéme budeme rozlišovať medzi dvomi typmi správ. Systémovými a štandardnými. Okrem toho budeme striktne rozdeľovať správy určené pre DS a pre MTS.

Systémové správy v sebe budú obsahovať informácie potrebné na správnu funkčnosť systému. Budú to:

- **Status správy –** správy typu *Hello* a *Bye* sa budú posielať vtedy, keď sa platforma zapne (*Hello*) alebo vypne (*Bye*)
- **Notify správa –** správa, ktorá bude obsahovať informácie o agentoch a GW agentoch

Štandardné správy sa budú posielať len medzi jednotlivými agentami a nebudú určené pre systém.

3.4 Platform

Medzi úlohy platformy bude okrem úloh rozhrania medzi DS a MTS patriť udržiavanie zoznamu všetkých agentov v systéme a informácií o nich. V tomto ohľade bude spolupracovať s DS, ktorý jej poskytne aktuálne informácie. Ak budú nejaké informácie o agentovi neaktuálne, aktualizuje ich. Pokiaľ ide o transport adresy, aktualizuje ich alebo ich vymaže. Ak potom nebude mať agent informáciu o žiadnej transport adrese, platforma ho bude považovať za neexistujúceho, pretože ho nebude mať ako kontaktovať a zo zoznamu ho vymaže. Platforma bude kontrolovať platnosť transport adries v pravidelných intervaloch.

V súvislosti s MTS bude mať platforma na starosti odovzdávanie správ MTS od agentov na nej bežiacich spolu s agentami, ktorým majú byť tieto správy doručené. Adresy na týchto agentov mu bude poskytovať zo svojho zoznamu. Táto úloha platí aj spätne. Platforma prevezme správu od MTS pre svojich agentov a odovzdá im ju na spracovanie.

Platforma bude pracovať v dvoch módoch – normálny a GW mód. Normálny mód pracuje podľa tohto návrhu, kým GW mód má dodatočné požiadavky na funkčnosť MTS a DS. Vtedy budeme platformu označovať ako GW platformu.

3.5 Discovery Service

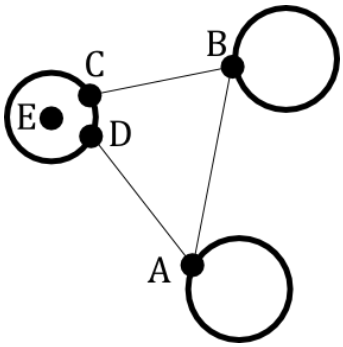
DS ihneď po spustení oznámi svoju prítomnosť na lokálnej sieti pomocou správy *Hello* čím u ostatných DS spustí odoslanie správy *Notify* (viď nižšie). DS oznamujúca svoju prítomnosť týmto krokom zistí existenciu agentov na svojej lokálnej sieti. V prípade, že sa mu nepodarí nikoho kontaktovať a nedostane žiadny zoznam aktívnych agentov, predpokladá, že sa v systéme nachádza sám. V takomto prípade musí počkať, kým sa do systému pripojí pre neho viditeľný agent, ktorý by mu bol schopný tieto informácie poskytnúť alebo sa premiestni do lokálnej siete s aktívnymi členmi.

Počas svojho behu bude v pravidelných, pevne stanovených intervaloch posielať správu *Notify*, aby informoval ostatné DS o súčasnom stave aktívnych platformových

agentov. Pokiaľ je DS súčasťou GW platformy, oznamuje na svojej lokálnej sieti aj agentov zo vzdialených lokálnych sietí a naopak, vzdialené agenty oznamuje agentom na svojej lokálnej sieti. Od DS očakávame, že zistí všetky možné cesty ku všetkým agentom, ktoré podporujú rovnaký typ DS bez toho, aby cesta bola výsledkom cyklu. Táto požiadavka vznikla preto, aby keď nastane situácia, že zanikne cesta k nejakému agentovi a reálne existuje iná, tak by mala byť ihneď použiteľná, nie až po ďalšej *Notify* správe.

Príklad (Obrázok č. 3): Nech kružnice vyjadrujú lokálne siete. Body na jej obvode nech sú GW agenty a body vnútri kružnice, nech sú štandardné agenty. Úsečky medzi GW agentami nech vyjadrujú schopnosť navzájom sa kontaktovať. Predpokladáme, že GW agent vie principiálne kontaktovať štandardného agenta na svojej lokálnej sieti, a preto odpadáva potreba úsečiek. Nech agent A chce kontaktovať agenta E. Vyberie cestu cez agenta s najlepšou metrikou. Nech je to počet hopov. Agent A teda vyberie cestu cez GW agenta D (metrika má hodnotu 2). Teraz predpokladajme, že agent D sa odpojí a teda zanikne pôvodná cesta A-D-E. Nech A chce znovu poslať správu E. Ak by si ukladal iba práve jednu cestu k agentovi, tak by musel čakať, kým sa mu nejaká nová cesta oznámi, ale keďže si ukladá všetky možné cesty, vie, že existuje cesta cez B a C a preto správu ihneď odošle po tejto ceste.

Obrázok č. 3 Príklad siete s viacerými možnými cestami medzi agentami



Úlohou DS samozrejme bude *Notify* správy aj spracovávať. Ak je v zozname oznámených agentov taký, ktorý v zozname aktívnych agentov ešte nie je, pridá ho do zoznamu agentov s prislúchajúcimi informáciami. Ak sa už daný agent v zozname nachádza, informácie o ňom aktualizuje.

Keď sa DS zo systému odpája, pošle ostatným DS správu typu *Bye*. Podľa nej vedia, že transport adresy, ktoré tento DS oznamuje už nebudú platné a preto ich platforma náležite spracuje.

3.5.1 DS správy

DS správy sú určené pre všetkých agentov na lokálnej sieti odosielateľa. Keďže sa odosielajú na multicastovú skupinu, dostanú sa priamo k príjemcovi, a preto obsah správy je hneď prístupný a netreba k správe pridávať informácie o jej ceste, odosielateľovi a príjemcoch. Medzi DS správy patria správy *Hello*, *Bye* a *Notify*. DS sa nestará o štandardné správy. DS správy posielame vo formáte JSON.

3.5.1.1 Štruktúry DS správ

Štruktúra *Notify* obsahuje položky:

- type - typ správy
- agents – tu sa nachádza zoznam agentov, ktorých oznamujeme. V prípade, že je odosielateľ aj Gateway agent, v tejto položke oznamuje aj agentov, o ktorých preposielanie sa stará. Položka obsahuje samozrejme aj informácie o všetkých agentoch okrem *origins* pri transport adresách.
- gwAgents – je to zoznam adries Gateway agentov, ktoré sú aktívne v systéme.

Štruktúra *Hello* a *Bye* je rovnaká:

- type - ako u *Notify*
- address – adresa platformy. Podľa typu správy sa pridá do zoznamu alebo sa zo zoznamu odstránia všetky položky s ňou súvisiace (okrem prípadu zoznamu GW agentov).

3.6 Message Transport Service

MTS bude zodpovedné za spracovávanie štandardných správ v prípadoch:

- prijímanie,
- odosielanie
- preposielanie (ak je súčasťou GW platformy)

Najprv preberieme odosielanie správ. MTS prevezme od platformy správu na odoslanie, k nej dostane informácie o tom, kto ju posiela a mená agentov, ktorým ju posiela. MTS vyberie transport adresu agenta podľa najlepšej metriky podľa routovacieho protokolu. Pre každú rôznu transport adresu sa vytvorí obálka, ktorá obsahuje informáciu o odosielateľovi a príjemcoch tejto obálky. Ak MTS vyberie pre rôznych agentov rovnakú transport adresu, príjemcovia tejto obálky budú všetky agenty s touto adresou.

Príklad: Pre správu majme príjemcov A, B, C. MTS vyberie pre A, B adresu T1 a pre C adresu T2. Potom pre adresu T1 vytvorí obálku, kde príjemcovia sú A a B. Pre adresu T2 vytvorí druhú obálku, kde príjemca je iba C.

Obálku, ktorá bude obsahovať taktiež aj správu odošle na príslušnú adresu. Na druhej strane sa táto obálka spracuje podľa príjemcov a rozhodne sa, či správa patrí nejakému z platformových agentov alebo treba správu preposlať ďalej. Ak ide o prípad preposielania, správu vložíme do obálky s upravenými príjemcami a pošleme na adresu, ktorá je vyberaná obdobne ako v prvom prípade. Ak ide o prípad prijímania, MTS odovzdá správu pre platformového agenta platforme, ktorá ho odovzdá agentovi, pre ktorého je správa určená.

3.6.1 MTS správy

Správy MTS budeme formátovať do XML. Z dôvodu možnosti putovania týchto správ po internete vznikla potreba komplikovanejšieho spôsobu posielania týchto správ oproti DS správam. V prvom rade sa tieto správy budú posielať v XML obálke (*Envelope).* MTS obsluhuje hlavne štandardné správy. Tieto správy MTS dostáva od platformy už v takom tvare, aby sa jednoducho vložili do obálky. V tejto práci sa ich štruktúre nevenujeme, pretože to je mimo rámca tejto práce. Pokiaľ bude MTS súčasťou GW platformy, bude spracovávať aj správy typu *Hello, Bye* a *Notify*, ktoré sa budú posielať iba ostatným GW platformám vo formáte MTS správ. Štruktúra správ *Hello* a *Bye* je obdobná s DS správami tohto typu. Správa *Notify* sa líši v tom, že pri jednotlivých transport adresách agentov, ktoré budú v tejto správe oznamované pribudla položka *origins*. Navyše celá štruktúra *Notify* bude vložená do XML elementu *notifyInfo.*

3.6.1.1 Envelope

Štruktúra obálky bude pre všetky typy správ rovnaká až na jednu položku. Keďže v prípade MTS sa budú všetky správy okrem štandardných správ posielať iba GW

agentom, budú sa líšiť tieto správy v tom, že štandardné správy budú mať navyše položku *recipients*. Štruktúra obálky bude vyzerať nasledovne:

- messageType - to isté, ako type u DS správy.
- sender – odosielateľ správy. V prípade *Standard Message* je to identifikátor agenta (meno), u ostatných je to adresa platformy
- recipients – mená agentov, ktorým má byť doručená táto obálka. Tento zoznam nie je ekvivalentný s menami agentov, ktorým má byť doručená správa.
- message – do tejto položky sa ukladá samotná správa

3.7 Gateway Agent

Gateway Agent bude zodpovedať za to, aby boli agenty na vzdialených lokálnych sieťach oznámené na jeho lokálnej sieti a aby agenty z jeho lokálnej siete boli oznámené na tých vzdialených. Navyše sa bude starať o posielanie a doručovanie správ medzi navzájom vzdialenými lokálnymi sieťami. Kvôli tomu, že agenty na lokálnej sieti nie sú schopné kontaktovať vzdialené lokálne siete, vzniká potreba využívať routovanie. A pretože jednotlivé agenty a hlavne gateway agenty nemusia byť stále pripojené do systému, je nutné, aby toto routovanie bolo dynamické.

3.8 Routovanie

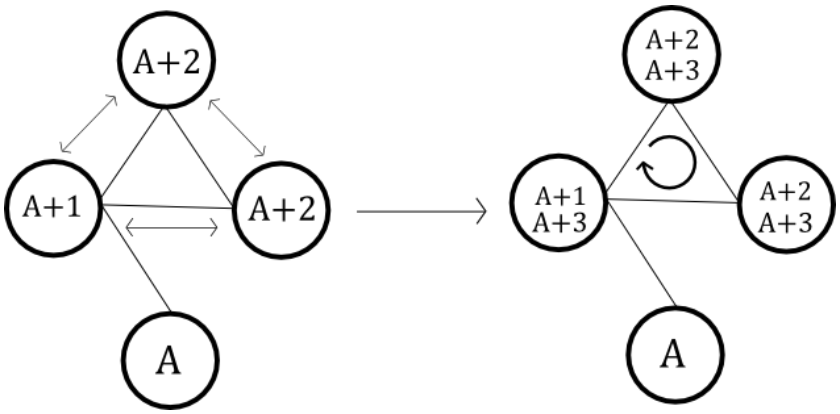
Routovanie správ v našom systéme bude mierne inšpirované routovacím protokolom RIP. Za metriku budeme považovať počet skokov a všetkým agentom sa bude posielať zoznam GW agentov a informácie o tom, za ktoré agenty je odosielateľ zoznamu zodpovedný.

Napriek tomu, že by obmedzenie na počet hopov, ktoré implementuje RIP, riešilo cykly v systéme, je to v našom prípade zlé riešenie, pretože by sa systémom šírili nepravdivé informácie. Preto sme vymysleli iný spôsob ako sa vyhýbať cyklom. V transport adrese sme pridali informáciu *origins.* Hovorí o tom, ktoré GW agenty túto adresu ohlasujú. Cykly v routovaní by mali za následok, že by sa adresa ohlasovala vždy s o jedno väčšou metrikou a musela by byť uložená, lebo by sa považovala za úplne novú adresu (viď. obrázok 4). Takto je každá adresa jednoznačne definovaná pomocou zoznamu *origins*. Adresu neohlasujeme GW agentovi, ktorý sa nachádza v tomto zozname. Týmto

zabezpečíme, že sa ostatní dozvedia všetky možné cesty k agentovi a zároveň v nich nevznikne cyklus.

Príklad cyklu: Po tom, ako agent oznámi ostatným adresu s metrikou A+1, bude stav vyzerať ako na ľavej strane Obrázok č. 4. V ďalšom kole správ *Notify*, ak by prebiehali súčasne, bude vyzerať stav ako na pravej strane. Tam môžeme pozorovať, že jeden agent má adresy s metrikami A+1 a A+3. To je chybné, pretože adresa A+3 je dôsledkom spomínaného cyklu.

Obrázok č. 4 Stav oznámených transport adries po dvoch správach *Notify*



Cyklus môže vzniknúť aj na lokálnej sieti. V prípade, že na jednej lokálnej sieti existuje viac ako jeden GW agent si tieto GW agenty oznamujú pomocou MTS aj agenty, ktoré patria do ich spoločnej lokálnej siete. V takejto situácii by si agenty ukladali zbytočné informácie, pretože agentov na svojej lokálnej sieti dokážu kontaktovať vždy priamo. Tento problém je jednoducho riešiteľný tak, že transport adresy platformových agentov ohlasujeme s metrikou 0. Tým ich identifikujeme ako lokálnych. Ak bude teda agentovi oznámená transport adresa s metrikou aspoň 1 a daný agent je kontaktovateľný pomocou transport adresy s metrikou 0, nová transport adresa sa neuloží.

Ďalší cyklus by mohol vzniknúť ak by existovali viaceré GW agenty na jednej lokálnej sieti. Potom by cez DS ohlasovali aj transport adresy, ktoré im cez neho boli oznámené. Vznikol by podobný problém ako v prvom prípade a posielali by sa nielen nepravdivé, ale aj zbytočné informácie. Práve preto sme k transport adrese pridali aj položku sourceDs, ktorá jednoznačne identifikuje, z ktorého DS bola transport adresa oznámená. Keď DS oznamuje transport adresy neplatformových agentov, vynechá také,

ktoré sám sprostredkoval. Tento spôsob používame kvôli budúcim plánom, keď budú existovať viaceré DS na jednej platforme.

3.9 Problémy plynúce z požiadaviek

V tejto časti sa budeme venovať problémom, ktoré sú priamym následkom návrhu. Pri každom probléme navrhneme aj jeho riešenie.

3.9.1 Problém prepojenia vzdialených lokálnych sietí

Ak máme dve vzdialené siete, v ktorých ani jeden z agentov z jednej siete nemá žiadny kontakt na hocijakého agenta z druhej siete, potom nie je možné, aby sa agenty vzájomne objavili cez internet bez nejakého centralizovaného prvku. V decentralizovaných systémoch ako implementácia DHT (*Distributed Hash Table*) [16] - Kademlia sa používa proces nazývaný *bootstrap*. Nový člen, ktorý sa pripojí do systému kontaktuje vopred určený, známy *bootstrap node*, ktorý mu odovzdá adresu na aspoň jedného člena siete, cez ktorého potom postupne získa adresy na ostatných členov. Napriek tomu, že DHT sa považuje za decentralizovaný systém, my v *bootstrap node* vidíme centralizovaný prvok, ktorý nie je ideologicky plne realizovateľný, pretože agenty sa v našom systéme môžu ľubovoľne vypínať a zapínať. V takomto prípade pre nás existuje málo možností ako naše dve vzdialené siete prepojiť. Jednou z možností je premiestniť agenta z jednej siete do siete druhého agenta. Tým pádom bude mať vo svojom zozname adresy agentov zo svojej predchádzajúcej siete spolu s adresami na Gateway agentov v systéme, ktoré v tomto prípade budú čiastočne plniť funkciu *bootstrap node*. Objavenie ostatných účastníkov na jeho súčasnej lokálnej sieti nebude problém, pretože samotný DS tieto prostriedky ponúka.

3.9.2 Problém neaktívneho agenta

V ideálnych podmienkach platforma pri odchode zo systému oznámi svoj odchod správou typu *Bye*. Po prijatí správy sa informácie súvisiace s ňou odstránia. Čo ak platforma odíde zo systému nečakane následkom výpadku prúdu alebo zlyhania techniky? Riešenie tohto problému je jednoduché. Určíme dobu platnosti transport adries agentov. Dĺžka tejto doby závisí od toho či je to agent zo vzdialenej lokálnej siete (dlhšia) alebo „domácej“ (kratšia). V určitých intervaloch platforma spustí kontrolu platnosti adries agentov. Adresy, ktoré sú neplatné sa zo zoznamu odstránia. Ak agent nemá žiadne platné adresy, odstráni sa zo zoznamu aktívnych agentov.

4 Implementácia

Obsahom tejto kapitoly bude popis implementácie nášho návrhu. Uvedieme technológie, ktoré sme v našom riešení použili a takisto popíšeme riešenia jednotlivých komponentov a ich algoritmov.

4.1 Použité technológie

Napriek pôvodným plánom nezávislosti systému od vonkajších knižníc bolo riešenie tejto práce implementované za pomoci frameworku Qt, ktorý pracuje nad programovacím jazykom C++. Z dôvodu zjednodušenia práce sme použili vývojové prostredie Qt Creator pod linuxovou distribúciou Ubuntu 14.04. Pre potreby použitia funkcionality HTTP servera sme navyše použili webový C++11 framework Tufão, ktorý využíva funkcie Qt.

4.2 Riešenia jednotlivých komponentov

Táto sekcia obsahuje podrobnejší opis implementácie komponentov. Budeme sa venovať aj algoritmom, ktoré boli vytvorené pre toto riešenie.

4.2.1 AgentInfo

AgentInfo je zložený dátový typ, ktorý obsahuje dva druhy informácií o agentovi:

- Zložený dátový typ AgentDescription
- Hashmapu z reťazca na TransportAddressProperties.

4.2.1.1 AgentDescription

AgentDescription Obsahuje informácie, ktoré definujú agenta. Obsahuje tieto položky:

- Reťazec name – jednoznačný identifikátor agenta, používame ho pri odosielaní štandardných správ
- Zoznam reťazcov services – je to pole reťazcov, ktoré obsahuje informáciu o tom, aké služby vie tento agent ponúknuť
- Zoznam reťazcov flags – pole reťazcov, ktoré obsahujú príznaky (flags) agenta

4.2.1.2 TransportAddressProperties

Tento zložený dátový typ ukladá informácie o jednej adrese, ktorou možno kontaktovať agenta. Keďže ich môže byť viac, ukladajú sa v AgentInfo v hashovacej tabuľke. Jeho položky sú:

- celé číslo metric – metrika danej adresy.
- čas validUntil – informácia o tom, do kedy je adresa platná. Platnosť adresy platformových agentov pri posielaní správy *Notify* cez DS nastavujeme na 30 sekúnd od jej poslania a pri správe *Notify* cez MTS na tri minúty od jej odoslania.
- DiscoveryService *sourceDs – referencia na DS, ktoré transport adresu sprostredkovalo. Je to jediná informácia, ktorá sa neoznamuje v žiadnom variante správy *Notify*.
- Zoznam reťazcov origins – obsahuje zoznam GW agentov, ktoré ohlasujú túto adresu.

4.2.2 DiscoveryService

DS si ukladá referenciu na platformu, ktorej je súčasťou kvôli zoznamu agentov, a socket ktorý využíva na pripojenie do multicastovej skupiny. Jeho metódy slúžia na odosielanie, prijímanie a spracovávanie správ typu *Hello, Bye* a *Notify*.

Vo svojom konštruktore najskôr získa adresu na platformu. Potom socket pripojí na nami zvolený port, kde počúva na všetkých adresách. Následne sa pripojí do multicastovej skupiny, na ktorú chodia vyššie spomenuté správy. Po tom, ako je pripojený na túto skupinu, pošle na ňu správu *Hello*. Prichádzajúce dáta, ktoré sú vo formáte JSON, sa spracujú za pomoci tried QJsonDocument a z neho sa vytvorí mapa z reťazca na variant. Potom rozhodne, ako sa správa spracuje podľa jej typu získaného z mapy, ktorá je vstupný parameter do metód, ktoré spracovávajú obsah správ. *Notify* správu spracováva samotný DS, ale *Hello, Bye* správy spracováva platforma, pretože tieto správy sa spracovávajú rovnako pri DS aj MTS, kým spracovanie *Notify* prebieha na nich inak.

Spracovávanie *Notify* správ prebieha tak, že nové informácie o agentoch sa pridajú a staré sa aktualizujú. O spracovávaní správ *Hello* a *Bye* budeme hovoriť v sekcii *Platform*.

Pri odosielaní správy *Notify* najprv naplníme štruktúru tejto správy informáciami o platformových agentoch. Tento krok robíme vždy. Ak je ale platforma v GW móde,

v tom prípade DS oznámi aj agentov, ktorých nezískal z lokálnej siete a teda sú zo vzdialených lokálnych sietí. Ďalšou úlohou DS je ohlásiť ostatných GW agentov v systéme. Jednotlivé položky, ktoré sa posielajú sú popísané v odseku o spracovávaní tejto správy.

O posielanie správ *Hello* a *Bye* sa stará metóda, ktorá má za vstupné parametre typ správy, adresu odosielateľa správy a informáciu o tom, či správa prišla na DS alebo MTS.

4.2.3 MessageTransportService

MTS obsahuje referenciu na platformu z rovnakého dôvodu ako pri DS. Medzi jeho dôležité členské premenné patria HTTP server, request a response od Tufão. Jeho funkčnosť sa orientuje na prijímanie a odosielanie správ pomocou HTTP POST. Okrem toho však virtuálne ponúka funkčnosť DS, keďže práve v MTS sa riešia správy medzi GW agentami *Hello, Bye* a *Notify* Štandardné správy sa posielajú cez MTS aj obyčajným agentom.

Jediné, čo MTS vo svojom konštruktore spraví je, že si inicializuje premennú týkajúcu sa platformy a zapne server, aby počúval na porte 22222 na hocijaké adresy. Na rozdiel od DS, vo svojom konštruktore nepošle správu *Hello*, pretože GW agenty sa načítavajú až v konštruktore platformy, takže by sa táto správa nikam neposlala, pretože by nemala prijímateľov.

Správy, ktoré sa posielajú v MTS pochádzajú buď od platformy už v konečnom tvare (štandardné správy), alebo ich zloží samotné MTS (správy *Hello, Bye, Notify*) podľa požiadaviek na ich štruktúru. Správa je vstupný parameter pre metódu, ktorá ju vloží do obálky. Na konci tejto metódy sa obálka odošle správnym agentom cez HTTP POST, podľa adries na nej. Pri odosielaní správy *Notify* sa najprv ubezpečíme, že neoznamujeme transport adresy agentov takým GW agentom, ktoré sa nachádzajú v zozname *origins* danej transport adresy.

Keď príde na náš server kompletný HTTP Request, podľa typu a informácií v obálke sa MTS rozhodne ako so správou naloží:

- *Standard Message* – ak je správa podľa obálky určená pre nejakého platformového agenta, vyšle signál, že je správa preňho pripravená a odstráni ho zo zoznamu

príjemcov. Ak zoznam príjemcov nie je potom prázdny, ostávajúcich príjemcov a správu odovzdá vyššie metóde na odosielanie správ, ktorá s ňou korektne naloží.

- *Hello/Bye* – správa sa posunie platforme, aby si upravila zoznam agentov alebo aj GW agentov podľa jej obsahu

- *Notify* – telo správy sa spracuje podľa jej štruktúry a aktualizuje alebo pridá agentov či GW agentov, podľa jej obsahu

4.2.4 Platform

Platforma obsahuje samotné DS a MTS, obsahuje aj tri časovače. Ďalej sa stará o štyri zoznamy agentov. Prvé dva zoznamy sú typu hash z reťazca na AgentInfo a sú to zoznamy platformových agentov a ostatných obyčajných agentov v systéme. Tretí zoznam je zoznam reťazcov, ktorý obsahuje adresy gateway agentov za určité časové obdobie a v obmedzenom počte (*m_knownGatewayAgents)*. Tieto agenty nemusia byť aktívne. Posledný zoznam je tiež zoznam reťazcov, kde sú uložené adresy aktívnych GW agentov v danej dobe.

Platforma vo svojom konštruktore inicializuje DS a MTS. V tomto kroku získajú na ňu referenciu. Po inicializácii sa zo súboru načíta zoznam všetkých GW agentov *m_knownGatewayAgents*. Potom, ako je zoznam načítaný, nastavia sa časovače a udalosti, keď vyprší na nich čas. Všetky časovače sa periodicky opakujú. Sú to:

- Časovač kontroly aktuálnosti transport adries agentov – keď je čas v položke validUntil v transport adrese menší (skorší) ako čas v priebehu kontroly, transport adresa sa vymaže, ak agent, ktorému sa transport adresa vymaže neobsahuje žiadnu inú adresu, vymaže sa aj on. Časovač je nastavený, aby sa opakoval každých 5 sekúnd.
- Časovač posielania správy *Notify* cez DS – tento časovač je nastavený na každých desať sekúnd (tretina platnosti adresy od jej ohlásenia cez DS) a DS pošle správu *Notify* na svoju lokálnu sieť.
- Časovač posielania správy *Notify* cez MTS – z dôvodu menšieho zaťažovania siete je tento časovač nastavený, aby spustil posielanie správy *Notify* každú minútu (tretina platnosti adresy od jej ohlásenia cez MTS)

Na konci konštruktora GW platforma pošle cez MTS správu *Hello* ostatným GW agentom.

Okrem posielania pravidelných správ *Notify* a kontroly platnosti transport adries sa platforma stará o spracovávanie správ *Hello* a *Bye* od DS aj MTS. Ak správa *Hello* pochádza od DS, tak na ňu reaguje iba poslaním správy *Notify* cez DS. Ak pochádza od MTS a odosielateľ sa nenachádza v zoznamoch o GW agentoch, tak ho do nich pridá podľa potreby. Potom platforma povie MTS, aby poslalo správu *Notify* GW agentom, medzi ktorými už je aj odosielateľ správy *Hello*. Keď platforma spracuje správu *Bye*, vymaže transport adresy súvisiace s odosielateľom zo zoznamu agentov. Ak nejaký agent nemá žiadnu inú adresu, ktorou by sa dal kontaktovať, odstráni sa celý. Pokiaľ správa *Bye* pochádza od MTS, platforma vymaže jej odosielateľa zo zoznamu aktívnych GW agentov. V perzistentnom zozname však zostane.

Záver

V tejto práci sme rozšírili LCP o možnosť vzájomnej komunikácie agentov na rôznych lokálnych sieťach pomocou *Gateway* agenta, ktorý plní virtuálnu úlohu routera. Je schopný kontaktovať GW agentov, ktorých mal vo svojom zozname aj po presunutí do inej lokálnej siete. Ďalej sme LCP rozšírili o komponenty *Discovery Service* a *Message Transport Service* a čiastočne aj *Platform*. Vďaka DS sú sa agenty schopné vzájomne objaviť na lokálnej sieti. MST slúži na posielanie správ medzi agentami v systéme. MTS čiastočne implementuje aj funkcionalitu DS, ktorá pomáha plniť povinnosti GW agentom. Hlavným výsledkom tejto práce sú protokoly, ktoré tieto komponenty využívajú. Najdôležitejším je protokol, ktorý rieši routovanie v našom systéme a prevencia cyklov, ktorých následok by bol šírenie prebytočných a chybných informácií v systéme pri samotnom routovaní. Napriek tomu, že cieľ tejto práce bol splnený, máme ďalšie návrhy do budúcnosti pre riešenie určitých problémov, ktoré neboli vyriešené z dôsledku časovej tiesne a máme nápady na vylepšenia implementovaných komponentov.

Plány do budúcnosti

V tejto sekcii zhrnieme zopár problémov, ktoré sa môžu v systéme vyskytovať a ich možné riešenie. Ďalej v tejto sekcii spomenieme zopár plánovaných návrhov na doplnenie do systému.

Problém nedoručených správ

Tento problém v implementácii nie je vyriešený. Medzi tým, ako platforma vypadne zo systému a vypršaním platností transport adries jej agentov môže v najhoršom prípade vzniknúť hluché miesto až niekoľko minút. Správy sa jej agentom budú napriek tomu stále odosielať na platné adresy, ale nebude ich mať kto prijať. Jedným z možných riešení je ukladanie správ, ktoré sa nepodarilo doručiť, aby im boli odoslané pri ďalšom pripojení do systému. Pri tom musíme brať do úvahy aj to, že daná platforma sa nemusí pripojiť na tej istej lokálnej sieti a tak isto v tej chvíli nemusí byť platforma, ktorá ma správu preňho uloženú, aktívna. Ďalším zlepšením situácie by mohol byť takzvaný „*Proxy Bye“*, ktorý vychádza z toho, že platnosti adries vypadnutých agentov, ktoré sa ohlasujú na svojej lokálnej sieti vypršia skôr. Ak vyprší platnosť adresy na lokálnej sieti, môžeme predpokladať, že ani žiadna iná adresa na tohto agenta nebude platná, a preto by mohlo byť

úlohou GW agenta danej siete ohlásiť neprítomnosť spadnutých agentov ostatným. Toto riešenie môže v sebe niesť isté riziká, ktoré bude treba doriešiť, ako napríklad čo ak sa vzápätí spadnutá platforma pripojí, ohlási svoju prítomnosť správou *Hello* a následne na to ostatným príde správa *Proxy Bye,* ale riešenie tohto problému by nemalo byť také komplikované. Samotná riešenie *Proxy Bye* vie zmenšiť hluché miesto až na pol minúty.

Dynamické GW agenty

Predpokladáme, že v LCP vzniknú situácie, kedy bude treba, aby po vypnutí nejakého GW agenta, ho iný agent zastúpil. Momentálne je status GW agenta daný explicitne. Preto navrhujeme, aby sa vymyslel mechanizmus, ktorý bude mať za úlohu dohodu platforiem o tom, ktorá z nich bude plniť úlohy GW agenta v prípade potreby.

Úprava správ *Hello* a *Bye*

V súčasnosti sa tieto správy týkajú iba platformy, ktorá oznamuje či sa vypína alebo sa zapína. V budúcnosti by mali byť tieto správy závislé od pripojenia agentov, pretože teraz predpokladáme, že agenty na platforme sú statické, ale chceme docieliť to, aby sa mohli ľubovoľne vypínať a zapínať. Dokonca má zmysel, aby bola aktívna aj platforma bez agentov, ktorí na nej bežia. Stále môže plniť úlohy GW platformy. Preto by správy *Notify* cez MTS mali slúžiť aj ako oznámenie aktívnosti GW platformy, ktorá túto správu odoslala, aby sa mohol udržiavať aktuálny zoznam aktívnych GW agentov. Ak by sa tento nápad uskutočnil, bolo by treba pridať položku *validUntil* aj k zoznamu aktívnych GW agentov. V prípade, že by náš systém fungoval tak ako sme popísali, museli by sme upraviť správy *Hello* a *Bye* tak, aby obsahovali mená agentov, ktoré na platforme končia. Ak by správa neobsahovala žiadne meno, považovalo by sa to za situáciu, kedy končí celá platforma.

Literatúra

[1] Norvig, P. – Russell, S.: Artificial Intelligence: A Modern Approach. 3. vydanie. Englewood Cliffs, Prentice Hall, 2009. ISBN 13 978-0136042594

[2] Sycara, K. P.: Multiagent Systems. In: AI magazine, ročník 19, Leto 1998

[3] Foundation For Physical Intelligent Agents: FIPA ACL Message Structure Specification. Technická správa, FIPA, 2002

[4] Labrou, Y. – Finin T.: A Proposal for a new KQML Specification. Discourse, (TR CS-97-03), 1997.

[5] Ngobye M. - de Groot W. - van der Weide T.: Types and Priorities of Multiagent System Interactions. [online] 2010 [cit. 20.1.2014] Dostupné z <http://www.indecs.eu/2010/indecs2010-pp49-58.pdf>

[6] Tannenbaum A. S. – Wetherall D. J.: Computer Networks. 5. vydanie. Prentice Hall, 2010. ISBN 13 978-0132126953

[7] Malkin G.: Rip Version 2. [online] 1998 [cit. 21.1.2014] Dostupné z <http://tools.ietf.org/html/rfc2453>

[8] Moy J. : OSPF Version 2 [online] 1998 [cit. 21.1.2014] Dostupné z <http://www.ietf.org/rfc/rfc2328.txt>

[9] Rao S. D. – Murthy C. S. R.: Distributed dynamis QoS-aware routing in WDM optical networks [online] [cit. 22.1.2014] Dostupné z <http://202.114.89.42/resource/pdf/1278.pdf>

[10] Moreno A. – Valls A. – Riaño D.: PalliaSys: agent-based proactive monitoring of palliative patients [online] [cit. 22.1.2014] Dostupné z <http://deim.urv.cat/~itaka/Publicacions/iwpaams05.pdf>

[11] Dziuban V. – Čertický M. – Šiška J. – Vince M.: Lightweight Communication Platform for Heterogeneous Multi-context Systems: A Preliminary Report. Proceedings of the 2nd Workshop on Logic-based Interpretation of Context: Modelling and Applications (Log-IC 2011). 2011

[12] Jade - Java Agent DEvelopment Framework [online] [cit. 25.5.2014] Dostupné z <http://jade.tilab.com/index.html>

[13] Publicly Available Implementations of FIPA Specifications [online] [cit. 25.5.2014] Dostupné z <http://fipa.org/resources/livesystems.html>

[14] qmake Manual [online] [cit. 25.5.2014] Dostupné z <http://qt-project.org/doc/qt-4.8/qmake-manual.html>

[15] tufao Documentation [online] 1.3.2014 [cit. 25.5.2014] Dostupné z <http://vinipsmaker.github.io/tufao/ref/1.x/>

[16] Loewenstern A. – Norberg A.: DHT Protocol [online] 29.2.2008 [cit. 25.5.2014] Dostupné z <http://bittorrent.org/beps/bep_0005.html>

[17] The MokSAF Agents [online] [cit. 22.1.2014] Dostupné z <http://www.cs.cmu.edu/~./softagents/moksaf/agents.html>

[18] Parker D.: MAS/LUCC Resource Page: Introduction [online] [cit. 22.1.2014] Dostupné z <http://www.csiss.org/resources/maslucc/>