НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК “Інститут прикладного системного аналізу”
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

“___”_____2017 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код та назва спеціальності)

на тему: Аналіз мов програмування мультиагентних систем

Виконав: студент _4__ курсу, групи __ДА-31_
(шифр групи)

Атаманчук Віталій Богданович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник професор, Рогоза В.С. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант економічний __доцент к.е.н Рощина Н. В._____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль _____ст. викладач Бритов О.А._____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2017 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК “Інститут прикладного системного аналізу”
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський))

Спеціальність 7.05010102, 8.05010102 Інформаційні технології проектування
7.05010103, 8.05010103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2017 р.

ЗАВДАННЯ

на дипломний проект (роботу) студенту

Атаманчуку Віталію Богдановичу

(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) Аналіз мов програмування мультиагентних систем
керівник проекту (роботи) Рогоза Валерій Станіславович, професор
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Строк подання студентом проекту (роботи) 07.06.2017

3. Вихідні дані до проекту (роботи) _____

1. Мови програмування – Jason, Java, Go, Платформа – Jade.
2. Еталонні тести та порівняльні графіки

3. Форма реалізації – власна платформа створенна на мові Go.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Дослідження особливостей мов програмування агентів та мов комунікації між агентами в мультиагентній системі.
2. Створення прикладової мультиагентної систем из використанням однієї або кількох мов, які були вибрані автором бакалаврської роботи для досліджень.
3. Представлення та опис сценаріїв роботи мультиагентної системи на прикладах.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників, плакатів тощо)

1. UML діаграма платформи
2. Симуляція Smart Grid
3. Таблиця порівняння агентних мов програмування

6. Консультанти розділів проекту (роботи)[*]

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н. В., доцент.		

7. Дата видачі завдання 27.03.2017

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	27.03.2017	
2	Збір інформації	10.04.2017	
3	Вивчення варіантів реалізації та вибір варіанту для розробки	15.04.2017	
4	Вибір способу реалізації	17.04.2017	
5	Розробка плану тестування	20.04.2017	
6	Розробка програмної моделі	25.04.2017	
7	Розробка опису системи	25.04.2017	
8	Тестування прототипу	10.05.2017	
9	Оформлення дипломної роботи	31.05.2017	
10	Отримання допуску до захисту та подача роботи в ДЕК	06.06.2017	

Студент _____ В.Б. Атаманчук

(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ В.С. Рогоза

(підпис) (ініціали, прізвище)

[*]* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

АНОТАЦІЯ

Бакалаврської дипломної роботи Атаманчука Віталія Богдановича

на тему: «Аналіз мов програмування мультиагентних систем»

Особливу увагу в цій роботі на мовах програмування агентів, які можуть ефективно і підтримувати реалізацію многоагентних систем.

У статті дається порівняння між мовами програмування агентом. Для того, щоб визначити придатність мов для реалізації многоагентних систем. Це було зроблено за рахунок впровадження системи багатоагентна в Go мовою, а також шляхом проведення теоретичних досліджень в функціонуванню Джейд і Джейсона. Після цього, мови були зрівняні в теоретичній формі по аспектам простоти програмування, абстракції, повторне використання, і потенціал. Цілі, щоб об'єднати дослідників і практиків як з боку наукових кіл і промисловості до розвитку многоагентних систем.

Результати цього аналізу показують, що Go, Jason і Jade цілком можуть бути використані для впровадження системи багатоагентній. Це показує, що концепції многоагентних систем можуть бути набагато більш широким застосуванням, так як не знання конкретного мови програмування агента і інтеграція декількох мов програмування не потрібно, якщо програмісти відчувають в програмуванні за допомогою одного функціонального мови.

Загальний обсяг роботи – 78 сторінка, 13 рисунків, 7 таблиць, 14 посилань.

Ключові слова: МАС, агент, агент-оріентованна мова, Jason, Jade.

AN ABSTRACT OF THE THESIS OF

Atamanchuk Vitalii for the degree bachelor of the computer science in NTUU “KPI”

Title: “Analysis of multi-agent programming languages”

The specific focus of this paper is on agent programming languages that can effectively and efficiently support the implementation of multi-agent systems.

The paper gives a comparison between agent programming languages. In order to determine the suitability of languages for the implementation of multi-agent systems. This was done by implementing a multi-agent system in Go language, and by performing theoretic research into to the functioning of Jade and Jason. After this, the languages were compared in a theoretical manner on the aspects of ease of programming, abstraction, reusability, and potential. Aims to bring together the researchers and practitioners from both academia and industry to the development of multi-agent systems.

The results of this analysis indicate that Go, Jason and Jade could very well be used for implementing multi-agent system. This shows that the concepts of multi-agent systems could be much more widely used, since no knowledge of a specific agent programming language and integration of several programing languages are necessary, if the programmers is experienced in programming with a single functional language.

The total volume of work - 78 pages, 13 figures, 7 tables, 14 bibliographic references.

Keywords: Multi-agent system, agent, agent-oriented language, Jason, Jade.

АННОТАЦИЯ

Бакалаврской дипломной работе Атаманчука Виталия Богдановича

На тему: «Анализ языков програмирования мультиагентных систем»

Особое внимание в этой статье уделяется языкам операторского программирования, которые могут эффективно и эффективно поддерживать внедрение многоагентных систем.

В документе дается сравнение между языками программирования агентов. Чтобы определить пригодность языков для реализации мультиагентных систем. Это было сделано путем внедрения многоагентной системы на языке Go, а также путем проведения теоретических исследований по функционированию Jade и Jason. После этого языки теоретически сравнивались с аспектами простоты программирования, абстракции, повторного использования и потенциала. Цель состоит в том, чтобы объединить исследователей и практиков из академических кругов и промышленности в развитие многоагентных систем.

Результаты этого анализа показывают, что Go, Jason и Jade вполне могут быть использованы для реализации мультиагентной системы. Это показывает, что концепции многоагентных систем могут быть гораздо более широко использованы, поскольку знание программирующего языка конкретного агента и интеграция нескольких языков программирования не требуются, если программисты имеют опыт программирования с одним функциональным языком.

Общий объем работы - 78 страницы, 11 рисунка, 8 таблиц, 16 библиографических наименований.

Ключевые слова: МАС, агент, агент-ориентированный язык, Jason, Jade.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ,СКОРОЧЕНЬ І ТЕРМІНІВ

MAS - Multi-agent system

FIPA - Foundation for Intelligent Physical Agents

UML - Unified Modeling Language

RTT - round trip time

APL - agent programming language

ACL - Agent communication language

AMS - Agent management system

DF - Directory facilitator

KQML - Knowledge query and manipulation language

RMI - Remote method invocation

RPC - Remote procedure call

XML - eXtensible Markup Language

ВСТУП

Після багатьох років успішної експлуатації в даний час існує величезна кількість агентно-орієнтованих систем, орієнтовані на використовується в сотнях компаній по всьому світу, щоб вирішувати складні проблеми в численних областях. Однак, як тільки агентна технологія розмножилась і окремі системи збільшилися в розмірах і складності, нові проблеми і обмеження були відзначені в їх маштабованості, універсальність, можливість багаторазового використання, крихкості і суперечливості. Грубо сказано: (одинокі) агенти були занадто великими, щоб добре працювати. Для того, щоб подолати ці проблеми, була прийнята політика «розділяй і володарюй», і епоха багатоагентних систем(MAS) народилася.

Ідея цієї нової парадигми було розподіл завдання системи між різними агентами, так щоб загальна продуктивність системи покращилася з боку як швидкість обробки так і якості результату. Нові апаратні і програмні засоби, дозволили здійснення таких багатоагентних систем і довели, що це припущення було вірно.

В область систем на базі агентів продовжує швидко розвиватися, одна з найбільш істотних проблем полягають в можливості порівняти і оцінити відносні переваги та недоліки різних підходів до розробки багатоагентних систем з використанням різних мов програмування агента, акцентуючи увагу на створенню практичних розподілених систем.

Поняття складності пов'язане зі ступенем складності, щоб зрозуміти систему. Це поняття є ключовим фактором в оцінці витрат на розробку і зусиль. Крім того, воно впливає на зрозумілість створеного програмного продукту. Отже, складність програмного продукту впливає на зусилля з технічного обслуговування і вартості. Таким чином, вимірювання складності

програмного продукту може бути використана в якості індикатора для оцінки необхідного зусилля під час фази супроводу.

Гнучкість агентів робить їх поведінку непередбачуваною і зрозумілість розробленої системи стає більш важким. Таким чином, фаза супроводу стає все більш складною.

1 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ МОВ ПРОГРАМУВАННЯ АГЕНТІВ ТА МОВ КОМУНІКАЦІЇ МІЖ АГЕНТАМИ В МУЛЬТИАГЕНТНІЙ СИСТЕМІ

У цьому розділі розглядаються пов'язані проекти, що утворюють ідею цього диплома. Просуваючись через даний розділ, мотивація і обгрунтування даної дипломної роботи також будуть мати додатково пояснені.

1.1 Агент

Єдиного визначення агента немає (дивіться, наприклад, Genesereth і Ketchpel, 1994; Wooldridge і Jennings, 1995; Russell і Norvig, 2003), всі визначення згоджуються, що агент є по суті спеціальним програмним компонентом, який має автономію, яка забезпечує сумісний інтерфейс для будь-якої системи і / або веде себе як людський агент, що працює для деяких клієнтів в гонитві за свій власний порядок денний. Навіть якщо системний агент може бути оснований на одиночному агенті, що працює в середовищі і при необхідності взаємодіє з користувачами, як правило, вони складаються з безлічі агентів. Ці багатоагентні системи (MAS) можуть моделювати складні системи і ввести можливість агентів, що мають спільні або суперечливі цілі. Ці агенти можуть взаємодіяти один з одним як побічно (шляхом впливу на навколишнє середовище) або безпосередньо (за допомогою комунікації та переговорів). Агенти можуть прийняти рішення співпрацювати для взаємної вигоди або можуть конкурувати, щоб служити своїм власним інтересам.

Таким чином, агент є автономним, оскільки він працює без безпосереднього втручання людини або інших і має контроль над своїми діями і внутрішнім станом. Агент соціальний, оскільки він взаємодіє з людьми або іншими засобами для досягнення поставлених завдань. Агент реактивний, так

як він сприймає свою навколишнє середовище і реагує своєчасно на зміни, які відбуваються в навколишньому середовищі. І агент є активним, тому що він не просто діє у відповідь на навколишнє середовище, але може демонструвати цілеспрямовану поведінку, беручи ініціативу.

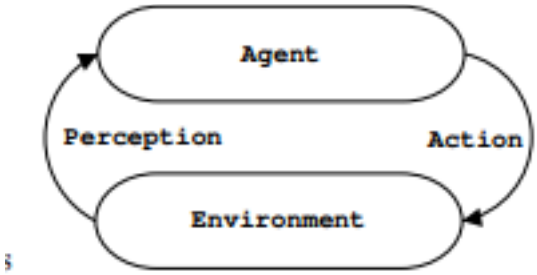

Рисунок 1.1 — Абстрактний вид агента[1]

Діаграма вище, являє собою абстрактне уявлення агента. На цій діаграмі ми можемо побачити результат дії викликаним агентом для того, щоб вплинути на його оточення. У більшості областей розумної складності, агент не матиме повний контроль над своїм середовищем. Він буде мати в кращому випадку частковий контроль, в тому, що він може впливати на нього.

Ключова проблема агента та, щоб вирішити, які з його дій він повинен виконати для того, щоб найкращим чином задовольнити свої закладенні завдання. Архітектура агента, з якої ми побачимо кілька прикладів далі в цій главі, реальна архітектура програмного забезпечення для систем прийняття рішень, впроваджених в навколишньому середовищі. Складність процесу прийняття рішень може залежати від цілого ряду різних властивостей навколишнього середовища. Russell і Norvig пропонують таку класифікацію властивостей середовища:

- Доступний проти недоступний
- Детермінований проти недетермінірованного
- Епізодичний проти неепізодичного
- Статичний проти динамічних

- Дискретний проти безперервного

1.2 Архітектури агентів

До сих пір ми розглядали агента тільки в абстрактному вигляді. Таким чином, в той час як ми досліджували властивості агентів, які підтримують і не підтримують стан, ми не зупинилися на том, щоб розглянути, який стан може приймати агент. Ми розглянемо чотири класи агентів:

- logic based агенти — в основі якого прийняття рішення реалізуються через логічний висновок;
- reactive агенти — в якому прийняття рішень здійснюється в тій чи іншій формі прямого відображення від ситуації до дії;
- belief-desire-intention агенти — в якому прийняття рішень залежить від маніпуляцій структур даних, репрезентуючи переконання, бажання і наміри агента;
- layered архітектура — в якому прийняття рішення реалізуються за допомогою різних програмних шарів, кожен з яких більш-менш явно міркувань про навколишнє середовище на різних рівнях абстракції.

У кожному з цих випадків, ми віддаляємося від абстрактного зору агентів, і починає робити цілком конкретні зобов'язання про внутрішню структуру і діяльності агентів. Кожна секція пояснює природу цих зобов'язань, припущень, на яких архітектури залежать, і відносних переваг і недоліків кожного з них.

1.2.1 Logic-based архітектура

Логіка на основі архітектура також відома як символічна або дорадчий архітектура є одним з ранньої архітектури агента, який ґрунтується на припущенні про системах фізико-символ (Newell & Simon, 1976). Ця класична

архітектура заснована на традиційному штучному символічному підході уявлення і моделювання навколишнього середовища і поведінки агента з символічним поданням. Таким чином, поведінка агента на основі маніпулювання символічного уявлення.

Перевага такого підходу полягає в тому, що людське знання є символічним, таким чином кодування легше, і вони можуть бути побудовані, щоб бути обчислювально повним, що робить його легше для людей, для того щоб зрозуміти логіку. До недоліків можна віднести, те що важко перевести в реальний світ в точний, адекватний символічний опис, а символічне уявлення і маніпулювання може зайняти значний час, щоб виконати з результатами які часто доступні занадто пізно, щоб бути корисним.

1.2.2 Reactive-based архітектура

Реактивна архітектура це агент на основі прямого відображення ситуації до дії. Вона відрізняється від архітектури логічної основи, де не використовуються центральна символічна модель світу і складне символічне міркування. Агент реакції на зміни в навколишньому середовищі в стимул-реакція на основі. Реактивна архітектура реалізуються за допомогою набору датчиків і ефекторів, де перцептивний вхід відображених на ефектори до змін у навколишньому середовищі. Ймовірно, найвідоміший реактивна архітектура архітектура категоризація Брукса (Brooks, 1991).

Перевага такого підходу полягає в тому, що він буде працювати краще (тобто реагувати швидше, але не причина, тим краще) в динамічних умовах, а також те, що вони часто простіше по конструкції, ніж logic-based агент. Однак, той факт, що реактивны агенти не використовують моделі своїх результатів навколишнього середовища в деяких недоліках.

1.2.3 Belief-Desire-Intention (BDI) архітектура

BDI (віра, бажання, намір) архітектура, ймовірно, найбільш популярна архітектура агента. Вони мають своє коріння у філософії і пропонують логічну теорію, яка визначає розумові відносини віри, бажання і наміри, використовуючи модальну логіку.

Один з найвідоміших BDI архітектур є Procedural Reasoning System (PRS). Ця архітектура заснована на чотирьох основних структур даних: переконання, бажання, наміри і плани, і перекладач (рис. 1.2).

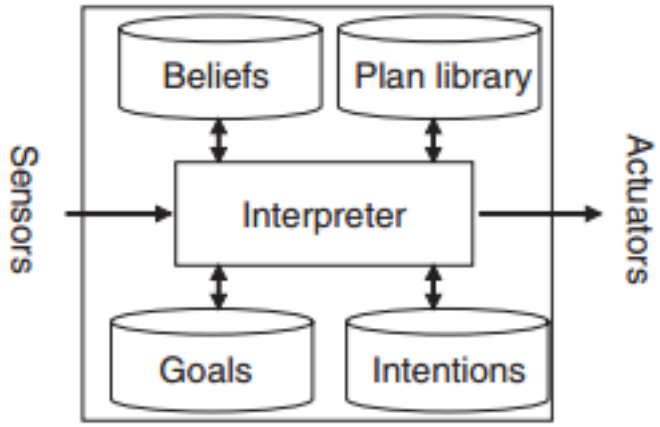

Рисунок 1.2 - Процедурна розмірковуюча система[1]

В системі PRS, вірування представляють інформацію агент має про своєму середовищі, яка може бути неповною або неточною. Бажання представляють завдання, покладені на агент і так відповідають цілям, або цілям, він повинен виконати. Наміри представляють бажання, що агент прагне до досягнення. Нарешті, плани вказати кілька курсів дій, які можуть бути потім агентом для досягнення своїх намірів. Ці чотири структур даних управляються інтерпретатором агента, який відповідає за оновлення переконання зі спостережень, зроблених з навколишнього середовища, створюючи нові бажання (завдання) на основі нових переконань, і вибір з

безлічі поточних активних бажань деякого підмножини, щоб діяти як наміри. Нарешті, інтерпретатор повинен вибрати дію для виконання на основі поточних намірів агента і процедурних знань.

Переваги архітектури BDI є те, що дизайн архітектури є простим і інтуїтивно зрозумілим. Проте, з BDI архітектурою питання про те, як ефективно реалізувати функціональність в підсистемі не ясно, і тому агенти повинні досягти балансу між зобов'язаннями і переосмисленням. Якщо агент не переставав переглядати, то, можливо, намагається досягти наміри, що не досяжні або більше не діє. Якщо агент часто переосмислює, він може зіткнутися з ризиком їх не досягти через нестачу часу роботи над завданням.

1.2.4 Layered архітектура

Layered (hybrid) архітектура являє собою архітектуру агента, який дозволяє як реактивне і навмисне поведінку агента. Багаторівнева архітектура поєднує в собі перевагу реактивної і логічній основі архітектури та в той же час усуває проблеми в обох архітектур. Підсистеми розкладаються в шар ієрархічної структури, щоб мати справу з різною поведінкою. Є два типи взаємодій, які течуть між шаром, а саме горизонтальним (рис. 1.3) і вертикальним (рис. 1.4). В архітектурі горизонтального шару, кожен шар безпосередньо з'єднаний з сенсорним введенням і виведенням дій. Кожен шар, як засіб відображення введення в дію має бути виконане.

Основна перевага цього є простотою конструкції, так як, якщо агент повинен п різних типів поведінки, то архітектура вимагає тільки п шарів. Однак, так як кожен шар діє агент, їх дії можуть бути несумісними пробуджуючи потреба в функції посередника, щоб контролювати дії. Ще одна складність є велике число можливих взаємодій між горизонтальними шарами. Архітектура вертикального шару усуває деякі з цих проблем.

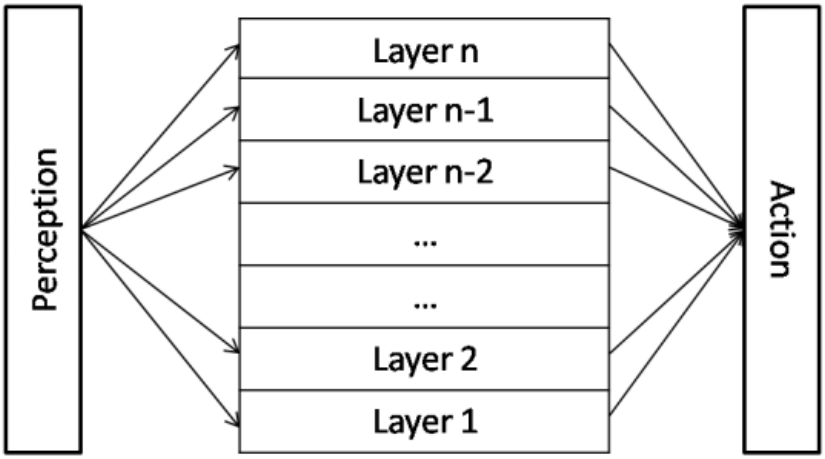

Рисунок 1.3 — Дані та управління потоками в горизонтальній архітектурі

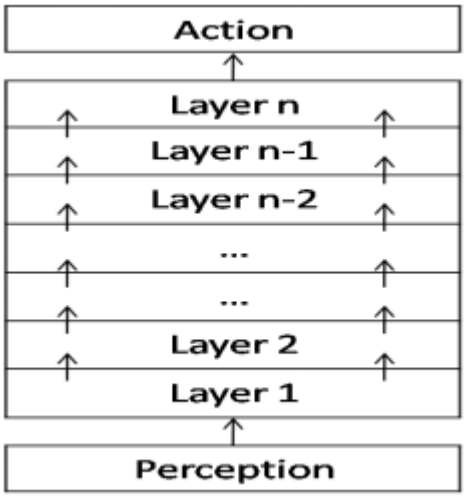

Рисунок 1.4 - Дані та управління потоками в вертикальній архітектурі

1.3 Комунікація і координація

Це одна з визначальних характеристик системи мультиагентной. У традиційному лінгвістичного аналізу, зв'язок береться мати певну форму (синтаксис), щоб нести певний сенс (семантика), а також під впливом різних обставин спілкування. Як ми побачимо, більш пильний погляд на комунікації додає до складності історії. В інформаційному повідомленні, агенти просто інформують одна одну про різні факти.

Зокрема, агенти взаємодіють один з одним за допомогою деяких спеціальних мов зв'язку, званих мовами спілкування агентів, які засновані на теорії мовних актів і які забезпечують поділ між комунікативними актами і мовою контенту.

В даний час найбільш широко використовуваний і вивчав мову агента зв'язку є FIPA ACL. Основні особливості Апія ACL є можливістю використання різних мов контенту та управління розмов через зумовлені протоколи взаємодії.

Координація є процес, в якому агенти займаються, щоб гарантувати, що спільність окремих агентів актів в когерентної Манні. Є кілька причин, чому кілька агентів повинні бути скоординовані в тому числі:

- Агенти мети можуть призвести до виникнення конфліктів між діями агентів;
- Агенти мети можуть бути взаємопов'язані;
- Агенти можуть мати різні можливості і різні знання;
- Агенти цілі можуть бути досягнуті більш швидко, якщо різні агенти працюють на кожному з них.

Координація між агентами може бути оброблена з різними підходами, включаючи організаційне структурування, укладення договору, планування багатоагентного і переговори.

1.4 The Foundation for Intelligent Physical Agents (FIPA)

The Foundation for Intelligent Physical Agents (FIPA) є міжнародною некомерційною асоціацією компаній і організацій, які поділяють зусилля для створення специфікацій загальних технологій агента. FIPA не сприяють технологіям тільки одного домена додатку, але набір загальних технологій для різних областей застосування, які розробники можуть інтегрувати створювати

складні системи з високим ступенем взаємозалежності працездатності. Процес стандартизації FIPA спирається на два основних припущення. Перше, що час, необхідний для досягнення консенсусу і завершити цей стандарт повинен бути якомога коротшим, не повинні перешкоджати прогресу, але повинен виступати в якості пропагандиста сильного промислового зобов'язання в області технологій агента. Друге припущення полягає в тому, що тільки зовнішня поведінка компонентів системи повинно бути вказано, в результаті чого деталі реалізації і внутрішні архітектури для розробників платформи. Справді, внутрішня архітектура JADE, Jason і призначеної для користувача платформа GO мов є власністю, навіть якщо він відповідає інтерфейсів, описаних FIPA.

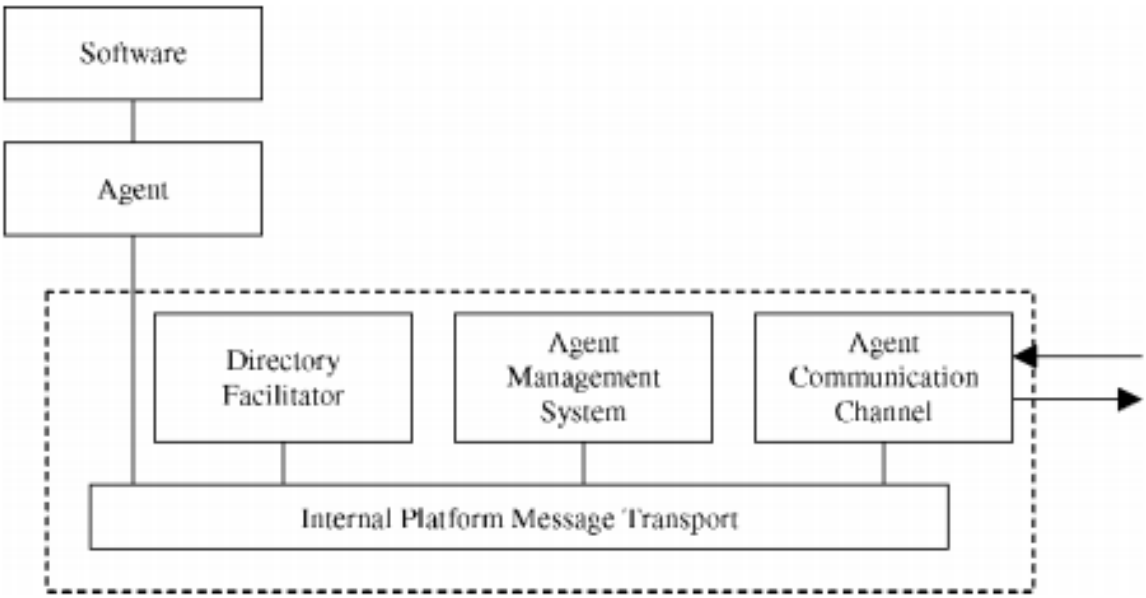

Рисунок 1.5 - Опорна модель платформи

Перші вихідні документи FIPA, названі FIPA97 специфікації, державні нормативні правила, що дозволяють суспільству агентів існувати, працювати і управлятися. Перш за все, вони описують еталонну модель платформи агента, як показано на рис. 1.5. Вони визначають ролі деяких ключових агентів, необхідних для управління платформою, і описати управління агентом мови

контенту і онтологію. Три обов'язкових ролі були визначені для платформи агента. Система агент управління (AMS) є агентом, який надає диспетчерський контроль над доступом і використанням платформи; він несе відповідальність за підтримання каталогу резидентних агентів і для обробки їх життєвого циклу. Канал зв'язку агента (ACC) забезпечує шлях для основного контакту між агентами всередині і зовні платформи. ACC спосіб зв'язку за замовчуванням, який пропонує надійне, впорядковане і точне обслуговування маршрутизації повідомлень. FIPA97 мандатів AKK підтримки міопії для того, щоб між працювати з іншими сумісними платформами агента. Довідник посередник (DF) є агентом, який проходить на жовтих сторінках послуг на платформу агента. Зверніть увагу на те, що ніяких обмежень не приділяється фактичної технології, використовуваної для реалізації платформи: платформа електронної пошти на основі, на основі CORBA один, багатопотокові програми Java і т.д. може все бути FIPA сумісних реалізацій.

Згідно з визначенням FIPA, агент є основним актором в домені. Вона здатна об'єднати цілий ряд сервісних можливостей, щоб сформувати єдину і інтегровану модель виконання, яка може включати в себе доступ до зовнішнього програмного забезпечення, права користувачів і засобів зв'язку.

Звичайно, технічні характеристики також визначити агент зв'язок Language (ACL). Агент зв'язок на основі передачі повідомлень, де агенти спілкуються, посилаючи окремі повідомлення один з одним. FIPA ACL є стандартною мовою повідомлення і встановлює кодування, семантику і прагматик повідомлень. Вона не встановлює конкретний механізм для транспортування повідомлень. Оскільки різні агенти можуть працювати на різних платформах і використовувати різні мережеві технології, повідомлення кодуються в текстовій формі. Передбачається, що агент має деякі засоби передачі цієї текстовій формі. Синтаксис ACL дуже близько до широко використовується мови спілкування KQML. Однак, незважаючи на синтаксичної схожість, існують принципові відмінності між KQML і ACL,

найбільш очевидним будучи існування формальної семантики FIPA ACL, який повинен усунути будь-яку невизначеність і плутанину з використанням мови.

Решта частини специфікації FIPA справу з іншими аспектами, зокрема, з інтеграцією агента програмного забезпечення, мобільності агента, безпеки агента, онтологія служби та комунікації людського агента. Однак, вони не зазначені тут, тому що вони ще не були розглянуті в реалізації платформи GO мов.

2 СТВОРЕННЯ ПРИКЛАДНОЇ МУЛЬТИАГЕНТНОЇ СИСТЕМИ

Там існує багато агентних мов програмування (APLs). Крім того, існує також платформи, які зосереджені на надання певних функцій - такі, як комунікація і загальна інфраструктура (наприклад, білі / жовті сторінки).

У цьому розділі ми зосередимося на агент орієнтованих мов програмування для визначення поведінки окремих агентів в системі багатоагентній. Загалом, в так званий «когнітивний мовах програмування агента», акцент робиться на тому, як описати поведінку агента з точкою зору конструкцій, такі як плани, подія, переконань, мети та шляхів сполучення. Хоча існує відмінність між різними пропонованим APLs, вони також мають суттєві загальні риси. Корисно відзначити, що в деяких з цих мов середовища не захоплені, що взаємодія агента здійснюються на рівні відправки окремих повідомлень - протоколи взаємодії, наприклад, не представлені - і що в багатьох мовах мета не передбачена, але вони апроксимується поняттям подій замість цього.

Більшість когнітивних мов програмування агент приходять зі своїми платформами. Ці платформи забезпечують загальну інфраструктуру (наприклад, система агента управління, біла і жовті сторінки), шар зв'язку, і інтегроване середовище розробки (IDE). Існуючі IDE, надають редактор з підсвічуванням синтаксису об'єктів, включити набір програм агентів, які буде виконуватися паралельно в різних режимах, таких як крок за кроком, або безперервними, надають інструменти для перевірки внутрішніх станів агентів під час виконання, і розглядати повідомлення які обмінюються агентами. Крім того, деякі з платформ, які приходять з когнітивними мов агента програмування, таких як Jadex і 2APL, побудованих на існуючій платформі

агента, такі як JADE. Отримувані платформи використовують функціональні можливості існуючих платформ, такі як загальна інфраструктура, засоби зв'язку і інструменти контролю.

Грубо кажучи, агент-орієнтовані мови програмування можуть бути класифіковані або як «теоретичне» (тобто, маючи формальну семантику, але, можливо, непрактичним для серйозного розвитку) або «практичний» (тобто, будучи практичним для серйозного розвитку, але не вистачає формальної семантики) , Однак, слід зазначити, що ця класифікація є спрощеною в тому, що мовах з формальної семантикою не виключається з практичності. Крім того, існує цілий ряд більш пізніх мов (або розширення існуючих мов, таких як Джейсон, який простягається AgentSpeak (L)), які прагнуть бути практичним, але мають формальну семантику. Зокрема, ми згадуємо 2APL, Jason і SPARK як приклади мов, які прагнуть бути практичними ще мають формальну семантику.

2.1 Об'єктно-орієнтоване програмування

В ООП основний акцент робиться на думки про розв'язуваної задачі в термінах реальних елементів і що представляють проблему в термінах об'єктів і їх поведінки. Класи зображують абстрактні уявлення об'єктів реального світу. Класи, як креслення і шаблони, які збирають подібні предмети або речі, які можуть бути згруповані разом. Класи мають властивості називаються атрибутами. Атрибути реалізовані у вигляді глобальних і змінні екземпляра. Методи в класах уявлення, або визначають поведінку цих класів. Методи і атрибути класів, називаються членами класу. Примірник класу називається об'єктом. Таким чином, об'єкт являє собою структуру даних, яка дуже нагадує якийсь реальний об'єкт.

Є кілька важливих концепцій ООП, такі як абстракції даних, інкапсуляція, поліморфізм, повідомлення, модульність і успадкування. Як правило, герметизація досягається шляхом атрибути приватної, в той час як створення спільних методів, які можна використовувати для доступу до них атрибутів. Спадкування дозволяє користувачеві розширити класи (звані підкласи) з інших класів (так звані супер-класів). Поліморфізм дозволяє програмісту замінити об'єкт класу замість об'єкта суперкласу. Як правило, імена, знайдені у визначенні проблеми, відразу ж стають класами в програмі. І точно так само, дієслова стають методами. Деякі з найбільш популярних мов об'єктно-орієнтованого програмування в Java.

2.2 Агент-орієнтоване програмування

АОП парадигма програмування введена Yoav Shoham. Мета АОП технології полягає в створенні систем, які можна застосувати до реального світу, що можна спостерігати і діяти на зміни в навколишньому середовищі. Такі системи повинні бути в змозі вести себе раціонально і автономно в завершенні своїх призначених завдань. Агент-орієнтована технологія являє собою підхід для побудови складних реального часу розподілених додатків. Ця технологія заснована на переконанні, що комп'ютерна система повинна бути спроектована, щоб демонструвати раціональні цілеспрямовану поведінку, аналогічного тому, що людських істот.

У АОП об'єкти, відомі як агенти взаємодіють для досягнення індивідуальних цілей. Агенти можуть бути автономними утвореннями, вирішуючи їх наступний крок без втручання користувача, або вони можуть бути керовані, що виступає в якості посередницького між користувачем і іншим агентом. програмування АОП виконується на абстрактному рівні. Агент-орієнтоване Software Engineering (AOSE) описуються як нова парадигма

для дослідницької області програмної інженерії. Але для того, щоб стати новою парадигмою для індустрії програмного забезпечення, надійної і простий у використанні методології та інструменти повинні бути розроблені.

2.3 Агент-орієнтоване програмування в порівнянні з ООП.

Агентно-орієнтований підхід є окремим випадком (спеціалізацією) об'єктно-орієнтованого програмування (ООП). В ООП обчислювальний процес розуміється досить широко як система, зібрана з об'єктів, які взаємодіють один з одним через повідомлення.

Таблиця 2.1 – Порівняння між ООП і АОП

	ООП	АОП
Базовий блок	об'єкт	агент
Параметри, що визначають стан основного блоку	невимушений	переконання, зобов'язання, можливості, вибір, ...
Процес обчислення	передача повідомлень і відповідей методи	передача повідомлень і відповідей методи
Типи повідомлень	невимушений	повідомляння, запит, пропозиція, обіцянка,
Обмеження по методам	-	чесність, послідовність, ...

АОП специализирует ці поняття, встановлюючи стан (зване психічним станом) об'єктів (званих агентами), що складаються з компонентів таких як вірування (переконання) (включаючи переконання про світ, про себе, і про один одного), здатності, і рішення, кожне з яких володіє певним синтаксисом.

2.3.1 Базовий блок

Точки, перераховані нижче, показує, що диференціюють між агентами і об'єктами:

1. Агенти можуть здійснювати зв'язок з використанням агента зв'язку мови ACL, тоді як об'єкти взаємодіють через інтерфейси фіксованих методів.
2. Агенти мають якість волі. Тобто, з використанням методів штучного інтелекту, інтелектуальні агенти здатні судити про їх результати, а потім змінити їх поведінку (і, таким чином, свою власну внутрішню структуру), щоб покращити сприйняття придатності.
3. Об'єкти абстракції речей, як рахунки-фактури. Агенти є абстракціями розумних істот - вони, по суті, антропоморфні. Зверніть увагу, що це не означає, що агенти розумні в людському сенсі, тільки те, що вони моделюють антропоморфні архітектури, з переконаннями, бажаннями і т.д.

2.3.2 Параметри, що визначають стан основного блоку

Агент АОП містить переконання, зобов'язання, вибір, і тому подібне, і спілкується один з одним за допомогою обмеженого набору типів мовних актів, таких, як повідомляють, прохання, обіцянку, відмовитися від стану агента називається його психічним станом.

2.3.3 Процес обчислення

Повідомлення об'єкта може запросити тільки одну операцію, і ця операція може тільки б е запитано через повідомлення, відформатований в дуже вимогливому способі. ОО Брокер повідомлень має справу відповідність кожне повідомлення точно один виклик методу рівно один об'єкт.

Агент-зв'язок може також використовувати виклик методу ОО. Однак, вимоги, що багато програм агента розміщувати на зміст повідомлення багатшими, ніж ті, які зазвичай використовуються технології об'єкта. У той час як мови агента зв'язку (ACL) формальні і недвозначні, їх формат і зміст сильно розрізняються. Коротше кажучи, повідомлення агента може складатися з рядка символів, форма може варіюватися ще підпорядковується формальний синтаксис, в той час як звичайний метод ГО повинен містити параметри, кількість і послідовність є фіксованим.

2.3.4 Тип повідомлення

Агенти, як правило, розглядаються як автономні одиниці, тому що вони можуть стежити за свій власний набір внутрішніх обов'язків. Крім того, агенти є інтерактивними об'єктами, які здатні використовувати багаті форми повідомлень. Ці повідомлення можуть підтримувати виклик методу, а також інформування агентів конкретних подій, з проханням щось агента, або отримання відповіді на більш ранній запит. І, нарешті, тому, що агенти є автономними, вони можуть ініціювати взаємодію і відповідати на повідомлення в будь-якому випадку вони вибирають. Іншими словами, агенти можна розглядати як об'єкти, які можуть сказати «ні», а також «так».

Завдяки інтерактивному і автономності агентів, мало чи ні інтеграції фізично не запустити додаток.

Об'єкти, з іншого боку, зазвичай пасивно-з їх методами, які викликаються при потоці абонента контролю. Термін автономія ледь відноситься до суті якого виклик залежить виключно від інших компонентів в системі.

2.3.5 Обмеження по методі

Як правило, класи об'єктів призначені бути передбачуваними для того, щоб полегшити покупку і продаж повторно використовувані компонентів. Агенти, як правило, призначені для визначення їхньої поведінки в залежності від індивідуальних цілей і станів, а також станів поточних розмов з іншими агентами. Хоча реалізації ГО можуть бути розроблена, щоб включити недетермінірованного поведінку, це поширене в Агентно мисленні.

Поведінка Агент також може бути непередбачуваним, тому що в наступному. По-перше, знання як агента може бути представлена таким чином, що не легко переводиться в набір атрибутів. Навіть якщо держава є агентом були загальнодоступними, це може бути важко розшифрувати або зрозуміти. По-друге, запитані поведінку, яке виконує агент може навіть не бути відомі в активній системі. Це явне відміну від об'єктних систем, оскільки існуючі мови ОО тільки дозволяють задати об'єкт, які інтерфейси він підтримує. Так як програміст повинен мати певне уявлення про те, що інтерфейс просити, це робить кодування важко. У ГО, не існує будь-яких положень діючих мов для об'єкта «рекламують» свої інтерфейси. На відміну від цього, агент може використовувати інші механізми, такі як публікація / підписка, реєстрація протоколу, жовтий сторінки і білі сторінки каталогів. Інший поширений механізм надає спеціалізовані брокерські агенти, які інші агенти можуть зробити самі по собі відомі для різних цілей, але в іншому Unlisted до іншої частини населення агента. І, нарешті, модель основної агента зв'язку, як правило, асинхронна. Це означає, що не існує зумовлене потік управління від одного агента до іншого. Агент може автономно ініціювати внутрішнє або зовнішнє поведінка в будь-який час, а не тільки тоді, коли він

відправив повідомлення. Там немає визначеного потоку управління від одного агента до іншого. Агент може автономно ініціювати внутрішнє або зовнішнє поведінка в будь-який час, а не тільки тоді, коли він відправив повідомлення.

2.4 Jason

Jason перекладач для нашої розширеної версії AgentSpeak, що дозволяє агентам бути розподілені по мережі за рахунок використання SACI. Jason є одним з найвідоміших систем многоагентних платформ на базі агента-орієнтоване програмування. Різні автори включили Jason в своїх порівняннях і аналізі мов програмування агента.

У Jason, агент є юридичною особою, що складається з набору переконань, що представляє поточний стан агента і знання про навколишнє середовище, в якій він знаходиться, набір цілей, які відповідають завданням агент повинен виконати / досягти, набір наміри, які є завданнями агента здійснила досягти, і безліч планів, які є курсами дій, викликаних подіями.

Події можуть бути пов'язані зі зміною або переконання бази агента або його метою. Агент реагує на події, створюючи нові наміри, за умови, існує застосовний план для цієї події. Тому кожне намір являє собою особливий «центр уваги» для вирішення різних завдань, в даний час здійснюється агент: всі вони конкурують за вибір агента про намір надалі виконуватися на цьому етапі виконання. Нарешті, ми повинні згадати «пул потоків» функціональність Jason, оголошені з допомогою (басейн, x) поряд з інфраструктурою вибору. Включення пулу нитки означає, що замість того, щоб створювати потік для кожного агента, Jason створює тільки фіксовану кількість потоків, які агенти конкурують за (якщо вони не мають нічого спільного). У наших експериментах ми вибрали x бути кількістю ядер, що використовуються з метою підвищення продуктивності на багатоядерних процесорах.

Деякі з функцій, доступних в Jason, є:

- мовної акт на основі зв'язок між агентом (і вірою анотація на інформаційних джерелах); анотації плану на основі міток, які можуть бути використані складними (наприклад, рішення теоретичного) функціями вибору;
- можливість запускати систему з безліччю розподілених агентів по мережі (з використанням SACI);
- повністю настроюється (в Java) функцій вибору, цільових функцій і архітектури в цілому агент (сприйняття, віра-перегляд, зв'язок між агентом і дії);
- проста розширюваність (і використання застарілого коду) за допомогою визначеного користувача «внутрішніх дій»;
- чітке уявлення про середовище багатоагентній, який реалізується в Java (це може бути моделювання реального середовища, наприклад, для цілей тестування, перш ніж система насправді розгорнута).

2.4.1 Jason інфраструктури

Jason в даний час надає дві інфраструктури для виконання MAS: централізована і JADE. У той час як централізована інфраструктура розміщує всі компоненти MAS в тому ж хості, можна також поширити ці компоненти в декількох хостів з використанням технології JADE. Як уже згадувалося, ця стаття зосереджена на централізовану інфраструктуру при виконанні на багатоядерних процесорах, як перший необхідний крок для характеристики цієї мультиагентной платформи на розподілених інфраструктур.

2.4.2 Режими виконання

Крім інфраструктури, платформа Jason пропонує три різних режиму виконання: асинхронний, синхронний і налагодження. В асинхронному

режимі, що він використовується за умовчанням, агент переходить до його наступного циклу міркування, як тільки він закінчить свій поточний цикл. Замість цього, в синхронному режимі всі агенти виконують один цикл міркування на кожному «кроці глобального виконання». Цей режим чекає, поки всі агенти не закінчили свої цикли міркування, а потім посилає «нести на" сигнал до них. Нарешті, режим виконання налагодження аналогічний синхронного режим, за винятком того, що Jason, поки користувач не натисне кнопку пульта «Крок» перед відправкою «звістку» сигнал до агентам. Режим виконання є параметром, який може мати значний вплив на продуктивність мультиагентной платформи.

2.5 JADE

JADE (Java Agent Development framework) є основою програмного забезпечення для полегшення розробки додатків агента у відповідності зі специфікаціями FIPA для між справної інтелектуальними багатоагентного системами. Метою JADE є спрощення розробки, забезпечуючи при цьому відповідно стандарту на основі всеосяжного набору системних служб і агентів. Для досягнення такої мети, JADE пропонує наступний перелік функцій програміста агента

- FIPA-сумісний Агент платформа, яка включає в себе AMS (System Management Agent), за замовчуванням Д.Ф. (Довідник викладач) і ACC (Агент канал зв'язку). Всі ці три агента автоматично включаються в агент платформи запуску.
- Розподілений агент платформа. Платформа агента може бути розділена на кілька хостів. Як правило, тільки один Java додатків, і тому тільки одна віртуальна машина Java, виконується на кожному хості. Агенти реалізовані у вигляді однієї Java потоку і Java події

використовуються для ефективного і легкого спілкування між агентами на той же хост. Паралельні завдання можуть бути виконані ще одним агентом, і графіки JADE ці завдання кооперативним способом (дивіться розділ «Розподілена платформа агента» для докладної інформації).

- Ряд FIPA-сумісного додаткового ДФХ (Координатор каталогу) може бути запущений під час виконання для того, щоб побудувати середовища мульти-домен, де домен являє собою логічний набір агентів, послуги яких оголошуються через загальний посередник.
- Java API для відправки / отримання повідомлень в / з інших агентів; Повідомлення ACL представлені як звичайні об'єкти Java.
- FIPA97-сумісний протокол IIOP для підключення різних платформ агента.
- Легкий перенесення повідомлень ACL в межах однієї і тієї ж платформи агента, так як повідомлення передаються закодовані як Java об'єкти, а не рядки, щоб уникнути маршалинга і немаршалінга процедур.
- Бібліотека протоколів Апіі взаємодії готових до використання.
- Графічний користувальницький інтерфейс для управління кілька агентів з того ж самого агента. Активність кожної платформи може контролюватися і реєструватися. Всі операції життєвого циклу на агентах (створення нового агента, призупинення або припинення існуючого агента і т.д.) можуть бути виконані за допомогою цього адміністративного GUI.

Система JADE може бути описана з двох різних точок зору. З одного боку, JADE часу виконання система для FIPA-сумісних многоагентних систем, що підтримують агентів додатки щоразу, коли вони повинні використовувати деяку функцію, покриту стандартної специфікації FIPA (передача повідомлень, управління життєвим циклом агента і т.д.). З іншого боку, JADE є основою для

розробки Java Fipa-сумісних програм агента, що робить стандартні FIPA активів, доступні для програміста через об'єктно-орієнтовані абстракції. Два наступних підрозділах будуть представлені JADE з двох точок зору, намагаючись виділити основні проектні рішення слідують команди розробників JADE. У заключному розділі обговорення прокоментуєте сильних і слабких сторін JADE і описати майбутні удосконалення, передбачені в розробці дорожньої карти JADE.

2.5.1 JADE архітектура системи реального часу

Працюючий платформа агент повинен надати кілька послуг для додатків: при погляді на частини 1 і 2 з FIPA97 специфікацій, то можна побачити, що ці послуги поділяються на два основних напрямки; передачі повідомлень підтримки з FIPA ACL і управлінням агентом з життєвим циклом, білим і жовтими сторінками і т.д.

2.5.2 Розподілена платформа

JADE Agent Platform відповідає FIPA97 специфікаціям і включає в себе всі системні агенти, керівники платформи; ACC, то AMS і DF за замовчуванням. Все засіб зв'язок здійснюється шляхом передачі повідомлень, де Апія ACL є мовою, використовуваним для подання повідомлень.

Архітектура зв'язку JADE намагається запропонувати гнучкі і ефективні передачу повідомлень, прозоро вибираючи кращий транспорт доступні і з використанням технології розподіленого об'єкта впровадженої впровадженої в середовищах виконання Java. Незважаючи на те, з'являючись як єдине ціле з зовнішнім світом, платформа агент JADE сам по собі є розподіленою системою, так як вона може бути розбита на кілька хостів з одним з них виступає в якості інтерфейсу для платформи між міопії зв'язку. Система JADE містить один або кілька агенти контейнерів, кожен з яких проживають в

окремій віртуальній машині Java і поставляючи підтримку середовища часу виконання для деяких JADE агентів. Java RMI використовується для обміну даними між контейнерами та кожен з них може виступати в якості клієнта міопії направити вихідні повідомлення на платформах іноземного агента. Особливий, Front End Контейнер також сервер міопія, слухаючи офіційна адреса ACCA агента платформи для вхідних повідомлень від інших платформ. Два обов'язкових системних агентів, тобто AMS і за замовчуванням DF, працювати в межах переднього кінця контейнера. Малюнок 2.5 показує архітектуру JADE агента платформи.

Нові контейнери агента можуть бути додані до працюючої платформі JADE; як тільки починається ні-передній кінець контейнер, то простий реєстраційний протокол з переднім кінцем, і додає себе в таблицю контейнера, підтримуваної переднім кінець. Користувачі можуть використовувати прості параметри командного рядка, щоб сказати на який хост і порт переднього кінець прослуховує реєстрації нових контейнерів.

Java Remote Method Invocation API використовується в якості комунікаційного середнього вироби, що робить JADE приймати вид єдиної платформи, навіть якщо вона поширюється. Кожен контейнер агента експортує віддалений інтерфейс RMI для забезпечення розподілених об'єктів інфраструктури для операцій платформи, таких як управління життєвого циклу агента. Наприклад, коли AMS просять призупинити агент, він часто може бути випадок, коли агент працює на інший контейнер, і це невідомо все агентів, AMS включено, тому що агент рівень адресація бачить JADE платформи в цілому, у відповідності зі специфікаціями іменування FIPA. З цієї причини, AMS може не довіряти, що лежить в основі інфраструктури JADE знайти відповідний агент. З цієї причини він просто викликає метод suspendAgent (), передаючи ім'я агента в якості аргументу. Цей метод буде виробляти локальний виклик, якщо агент призупинити життя в передньому кінці контейнера, або, якщо це не так, то віддалений виклик RMI.

2.5.3 Підсистема доставки повідомлення

Модель агента зв'язку FIPA є тимчасової мережі, хоча контекст мульти-повідомлення забезпечуються протоколами взаємодії і ідентифікатори розмови. Нефриту, з іншого боку, використовує транспортні технології, такі як RMI, CORBA і диспетчеризацію подій, які, як правило, пов'язану з клієнтами / сервером і реактивними системами. Очевидно, що існує розрив подолати для того, щоб зіставити ім'я в явному вигляді передачі повідомлень моделі, такий як FIPA ACL в зв'язку орієнтований, запит / відповідь моделі на основі зв'язку розподілених об'єктів. Ось чому в JADE звичайних агентів не розподілені об'єкти, тоді як контейнери агента.

Більш докладний опис необхідно різних методів, які JADE можна використовувати для пересилання повідомлень ACL і шляхи JADE вибирає з їх числа, щоб краще прояснити це важливе питання дизайну. Програмний агент, відповідно до моделі агента FIPA, має глобальний унікальний ідентифікатор (GUID), який може бути використаний будь-яким іншим агентом або програмного об'єктом для її вирішення з повідомленнями ACL. Крім того, агент поставить його GUID в: слот відправника повідомлень ACL він посилає навколо визнати себе в якості відправника повідомлення. Так, JADE повинні з'ясувати місце розташування приймача, просто подивившись на: приймач слот повідомлень. Оскільки FIPA-97 GUID нагадує адресу електронної пошти, він має вигляд: <agent_name> @ <PLATFORM_NAME> (де платформа адреса може бути або URL міопії або строковою CORBA IOR для платформи агента, розглядається як міопія досяжності CORBA об'єкт), і тому досить легко відновити ім'я агента і адреса платформи. JADE порівнює платформу адреса приймача зі своєю власною адресою платформи. Якщо вони відрізняються, приймач знаходиться на будь-якій іншій платформі, можливо, типу, що не JADE, і стандарт міопії повідомлень буде використовуватися. Таким чином, об'єкт ACLMessage Java, що представляє фактичне

повідомлення перетвориться в рядок і відправляється по каналу міопії, створений з платформою приймача.

З іншого боку, якщо приймач і передавач перебувають на тій же платформі агента, нефриту використовує диспетчеризацію події, коли два агента знаходяться в одному контейнері і Java RMI, коли вони знаходяться в різних контейнерах однієї і тієї ж платформі агента. Коли Java події використовуються, об'єкт ACLMessage просто клонували і передається агенту приймача. При використанні RMI, з іншого боку, об'єкт ACLMessage сериализации і десеріалізациі прозоро RMI під час виконання.

Тому, коли повідомлення про ACL відправляються до програмного агенту, є три варіанти.

- Приймач в тому ж контейнері тієї ж платформи: Java події використовуються; вартість є клонування об'єкта ACLMessage і локального виклику методу.
- Приймач в іншому контейнері тих же платформи: Java RMI використовується, вартість є сериализацией повідомлення на стороні відправника, виклик віддаленого методу і unserialization повідомлення на стороні приймача.
- Приймач на іншій платформі: CORBA IIOP використовується; вартість є перетворенням об'єкта ACLMessage в об'єкт струнного і міопія сортувальний на стороні відправника, віддалений виклик способу і міопія демаршаллінг з подальшим ACL синтаксичного аналізом на стороні приймача.

2.5.4 Інструменти для управління платформою і моніторингу

У доповненні до бібліотеки часу виконання, JADE пропонують деякі інструменти для управління біжить платформи агента і для моніторингу і суспільства налагодження агента. Всі ці інструменти реалізуються в якості

агентів FIPA себе, і вони не вимагають спеціальної підтримки для виконання своїх завдань; вони просто покладаються на JADE AMS. Загальна консоль управління для платформи агента JADE називається RMA (Remote Agent Management). RMA отримує інформацію про платформу і виконує графічний інтерфейс команди для зміни статусу платформи (створення нових агентів, закриття периферійних контейнерів і т.д.) через AMS. З одного боку, RMA просить AMS, щоб отримувати повідомлення про зміни в стані платформи агентів; з іншого боку, він передає в AMS запити для створення, видалення, припинення і перезапуску, отримані користувачем. Агент каталог посередник також має графічний інтерфейс своєї власної, за допомогою якої DF можна вводити, щоб додати або вилучити агентів і настройки їх рекламованих послуг.

Два графічних інструментів, з якими користувачі можуть JADE налагоджувати їх агенти є пустушки агент і Sniffer агент.

Пустушки Агент являє собою простий, але дуже корисний, інструмент для перевірки обміну повідомлень між агентами. Пустушки Agent полегшує перевірку обміну шаблону повідомлення агента до його інтеграції в систему мульти-агент і полегшує інтерактивне тестування агента. Графічний інтерфейс забезпечує підтримку для редагування, створення і відправки повідомлень ACL для агентів, отримувати та читати від агентів, і, в кінцевому рахунку, для збереження / завантаження повідомлень на / з диска.

Sniffer Agent дозволяє відстежувати повідомлення обмінені на платформі агента JADE. Коли користувач вирішує обнюхує один агент або групу агентів, кожне повідомлення, спрямовані до або з цього агента або група відстежується і відображається у вікні сніффер, використовуючи позначення, схожі на UML діаграми послідовності. Кожне повідомлення ACL може бути розглянуто користувачем, який може також зберігати і завантажувати кожне повідомлення треку для подальшого аналізу.

2.5.5 Jade agent development model

FIPA характеристики стану нічого про внутрішній устрій агента: це був явний вибір стандартного консорціуму, після того думки, що сумісність може бути надано тільки обов'язкове зовнішнє поведінка агента через стандартний ACL, протоколи, мови контенту і онтології, що дозволяє платформі внедренцев приймати різну конструкцію і шляхи реалізації. Незважаючи на те, що це правильно в принципі, коли конкретна реалізація таких як JADE повинна бути спроектована і побудована там ще дуже багато питань, які необхідно вирішити. Одним з найбільш важливих з цих питань, як агенти виконуються в межах своєї платформи. Це не погіршує сумісність, оскільки різні агенти спілкуються тільки через обмін повідомленнями. Проте, модель виконання для платформи агента є одним з основних дизайн проблеми, яка впливає як на продуктивність під час виконання і накладає певні стилі програмування на розробників додатків агента. Різні конкуруючі сили повинні бути прийняті до уваги при вирішенні такої проблеми. Як буде показано нижче, рішення JADE виникає з ретельного балансу цих сил. Деякі з них приходять від традиційних принципів інженерії програмного забезпечення, в той час як інші випливають з властивостей теоретичного агента і, отже, властиві системам інтелектуальних агентів.

2.5.6 Від теорії агента до класу дизайну

Відмітна властивість програмного агента є його автономність. Агент не обмежується реагувати на зовнішні подразники, але і в стані розпочати нові комунікативні акти самостійно. Дії, що виконуються агентом не тільки в залежності від отриманих повідомлень, але і від внутрішнього стану і накопиченої історії програмного агента.

Програмний агент, крім того, що автономно, називаються соціальними, тому що він може взаємодіяти з іншими агентами для досягнення своїх цілей і навіть може розробити загальну стратегію разом зі своїми однолітками.

Стандарт FIPA засновує Agent Communication Language (ACL) на теорії мовного акту і використовує менталістскій модель для побудови формального семантичного для різних видів повідомлень (перформативов) агенти можуть посилати один одному. Такий підхід сильно відрізняється від типу клієнт / сервер, а потім розподілених об'єктних систем і корениться в області дизайну за контрактом, де виклики виробляються на експортованих інтерфейсів. Фундаментальна відмінність полягає в тому, що, в той час як виклики на інтерфейсах можуть або успіх або невдача, запит мовного акту, крім згідно і невдач, можна також отримати покидьки перформативном, висловлюючи небажання агента приймача виконати потрібні дії.

Теоретичні агентні рішення вимоги дизайну є автономними ⇒ Агентами є активними об'єктами агентів є соціальним ⇒ Intra-агент паралелізму необхідних повідомленнями є мовними акти ⇒ асинхронного обміну повідомлень для використання агенти не можуть сказати «ні» ⇒ потрібна модель зв'язку Рівних-рівному

Властивість автономії вимагає, щоб кожен програмний агент, щоб бути активним об'єктом; кожен агент повинен мати принаймні один Java нитка для того, щоб мати можливість активно почати нові переговори і будувати плани і мати на меті. Абстрактна потреба в соціальність має практичний результат дозволяє агенту брати участь в декількох розмовах одночасно; тому програмний агент повинен мати справу зі значною кількістю паралельності.

Третя вимога передбачає асинхронні передачу повідомлень в якості здійсненного способу представлення обмінів інформацією між двома незалежними суб'єктами, які також мають додаткову перевагу отримання більш багаторазове використання взаємодій. Аналогічним чином, остання вимога підкреслює, що в системі з безліччю агента відправник і одержувач

мають рівні права (що не у випадку з системами клієнт / сервер, де приймач передбачається підкорятися відправнику). Крім того, будучи в стані сказати «ні», автономний агент повинен також мати можливість сказати: «Я не дбаю», не звертаючи уваги на отримане повідомлення, поки він хоче. Це додало вимога виключає чисто реактивні обробник повідомлень і пропагандист з використанням моделі обміну повідомлень тягнути споживач, де вхідні повідомлення буферизованная, поки їх приймач не вирішує їх читати.

2.5.7 Використання поведінки для створення складних агентів

Розробник, який хоче побудувати агент повинен розширити клас агента і здійснювати агенту конкретні завдання, написавши один або кілька Behavior підкласів, їх екземпляри і додавати об'єкти поведінки до агента. Призначені для користувача агенти успадковують від класу Agent основних можливостей реєстрації та скасування реєстрації своєї платформи і базовим набором методів, які можуть бути викликані для реалізації призначеного для користувача поведінки агента (наприклад, відправлення та одержання повідомлень ACL, використовують стандартні протоколи взаємодії, зареєструвати з декількома доменами). Крім того, призначені для користувача агенти успадковують від свого агента суперкласу два методів: addBehaviour (поведінку) і removeBehaviour (поведінку), які дозволяють керуючий списком поведінки агента.

JADE містить готові моделі поведінки для найбільш поширених завдань в програмуванні агентів, такі як відправка та отримання повідомлень і структурування складних завдань, як сукупності простих (Малюнок 4.1 показує анотований клас UML діаграми для JADE поведінки). Наприклад, JADE пропонує так звану Jess-поведінку, яке дозволяє повну інтеграцію з JESS. JESS є середовищем сценаріїв для правил програмування на основі написаного на Java, пропонуючи двигун, використовуючи алгоритм RETE для обробки правил. Таким чином, в той час як JADE забезпечує оболонку агента і

гарантує відповідність Fipa, JESS дозволяє використовувати правила-орієнтоване програмування для визначення поведінки агентів і використовує його двигун, щоб виконати їх.

Поведінка є абстрактним класом, який забезпечує скелет елементарної завдання повинні бути виконані. Він надає три методи: дія () метод, який представляє «справжню» завдання бути досягнуто шляхом конкретних класів поведінки; метод зроблено (), який використовується планувальником агента, який повинен повертати вірно, коли поведінка закінчило і може бути видалено з черги, помилково, коли поведінка ще не закінчені, і дії () метод повинен бути виконаний знову; спосіб скидання (), використовується для перезапуску поведінки з самого початку.

Через непріоритетної моделі багатозадачною обраної для поведінки агента, агента програмісти повинні уникати використання нескінченних петель і виконують довгих операцій всередині дій () методи. Це відбувається тому, що, коли певним чином впливати поведінку () не працює ніякого іншого поведінки не може тривати до кінця методу (звичайно, це справедливо тільки для поведінки одного і того ж агента: поведінка інших агентів працюють в різних Java потоків і все ще може тривати незалежно один від одного).

Крім того, оскільки ніякого контексту стека не зберігається, кожен раз, коли метод дії () запускається з самого початку: немає ніякого способу, щоб перервати поведінку в середині його дії (), перейти на CPU на інші види поведінки, а потім почати початкове поведінка туди, звідки вона була перервана. Наприклад, припустимо, конкретної операції оп () занадто довго, щоб бути запущена в одну стадію і, отже, розбита на три суб-операцій, названий OP1 (), OP2 () і op3 (); для досягнення бажаної функціональності необхідно викликати OP1 () в перший раз поведінку запускається, OP2 () вдруге і OP3 () в третій раз, після чого поведінка повинна бути позначені як припинено.

При роботі зі складною поведінкою агентів (наприклад, протоколи взаємодії агента) використання явного змінного стану може бути громіздким. З цієї причини, JADE слід композиційний підхід, який дозволяє розробникам додатків створювати свою власну поведінку на основі більш простих, прямо передбачених рамках. Цей клас також визначає два методи addSubBehaviour (Behavior) і removeSubBehaviour (Behavior), що дозволяє визначити рекурсивне агрегування декількох СУБів-поведінки. Так як ComplexBehaviour розширює поведінку, агент автор має можливість реалізації структурованого дерева, що складається з поведінки різних типів (включаючи самі ComplexBehaviours). Агент планувальник розглядає тільки топ-більшість завдань для її планування політики. Протягом кожного кванта часу «» (який, на практиці, відповідає одному виконання методи дії ()), призначеного до задачі агента тільки один подзадача виконується. Кожен раз, коли топ-більшість завдань повертаються, агент планувальник призначає керівництво до наступної задачі в готової черзі.

Крім ComplexBehaviour, структура JADE визначає деякі інші прямі підкласи поведінки: клас SimpleBehaviour може використовуватися розробником агента для реалізації атомних кроків роботи агента. Поведінкареалізується підклас SimpleBehaviour виконується за допомогою планувальника JADE в одному кадрі часу. Ще два підкласу виконують конкретні дії: SenderBehaviour і ReceiverBehaviour. SenderBehaviour дозволяє відправити повідомлення, в той час як ReceiverBehaviour дозволяє отримати повідомлення, яке може бути зіставляється з шаблоном. Самі поведінкові блоки (без зупинки всіх інших дій агента), якщо немає відповідних повідомлень не присутній в черзі.

Два класу ComplexBehaviour і SimpleBehaviourleave до їх подклассам вибір їх умов завершення, і для складного поведінки, фактичні діти планування політики. Для атомного поведінки, передбачені два підкласу. OneShotBehaviouris абстрактний клас, моделі поведінки, які повинні бути

виконані тільки один раз. CyclicBehaviour є абстрактним класом, моделі поведінки, які ніколи не закінчуються і повинні виконуватися безперервно. Клас ComplexBehaviour виконує відповідальність за організацію своїх дітей, але вона відкладає дитина про планування політики на підкласи. JADE надає готові підкласи із загальною політикою, але програмісти можуть вільно реалізовувати свої власні.

SequentialBehaviour є ComplexBehaviour, який виконує його суб-форми поведінку послідовно, він блокує, коли його поточний дитина блокується, і закінчується, коли всі його підпункти поведінку зроблено.

NonDeterministicBehaviour є ComplexBehaviour, який виконує свої дитина поведінка недетермінірованного, він блокує, коли всі його діти будуть заблоковані, і закінчується, коли певну умову на його підрозділи поведінки задовольняється. Наступні умови були реалізовані: закінчується, коли всі його поведінку суб-виконуються, коли який-небудь один з його суб-поведінки закінчується, або коли щонайменше, N суб-поведінки закінчили.

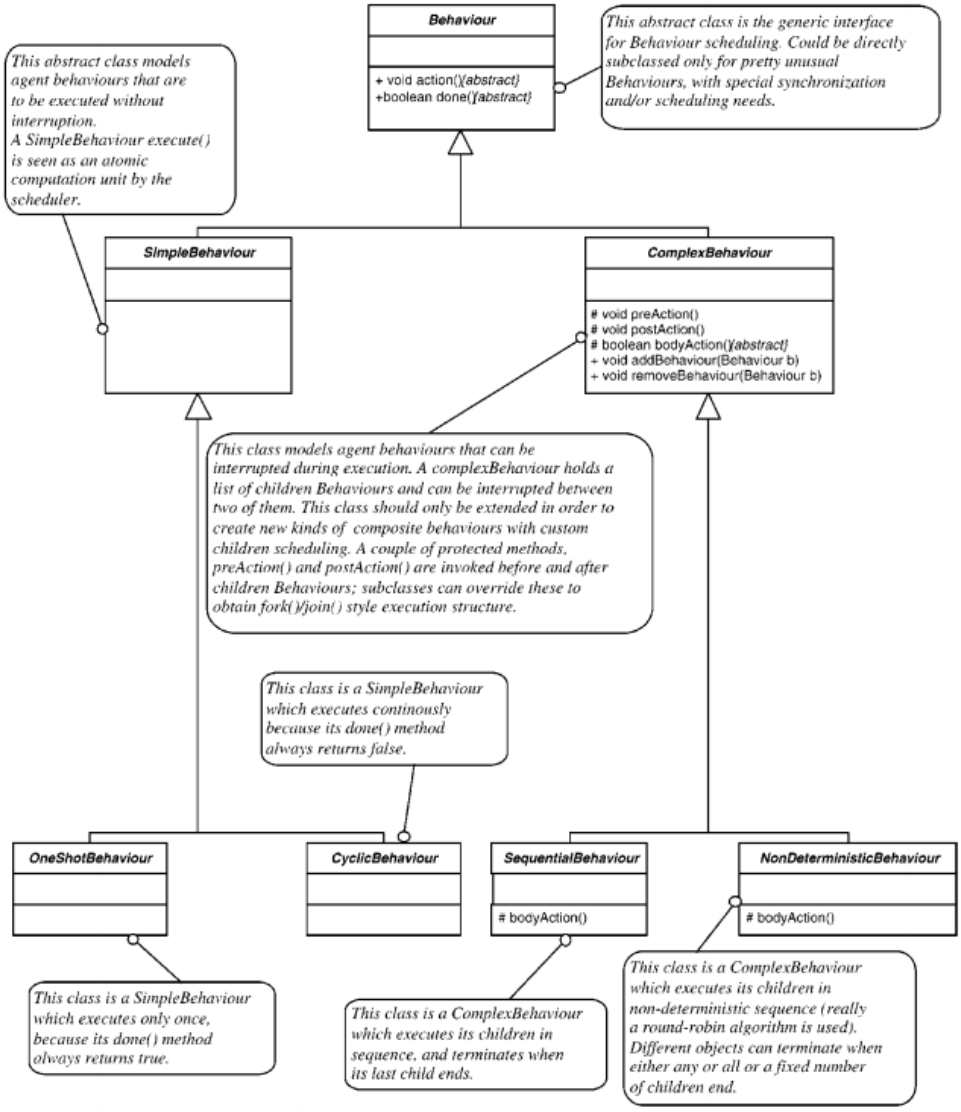

Рисунок 2.1 - UML модель ієрархії класів поведінки

JADE рекурсивний агрегування об'єктів поводження нагадує техніку, використовувану для графічних призначених для користувача інтерфейсів, де кожен інтерфейс віджета може бути листочок дерева, чиї проміжні вузли спеціальні контейнери віджетів, з обох візуалізації і управління дітьми особливостей. Важлива відмінність, однак, існує: JADE поведінки є матеріалізацією завдань виконання, тому планування завдань і підвіски повинні бути розглянуто, теж.

2.5.8 Продуктивність JADE

Незважаючи на те, JADE був розроблений з ефективністю на увазі, отримання «високу продуктивність за рахунок конструкції» трохи більше, ніж видавати бажане за дійсне, а реальні вимірювання повинні бути виконані, щоб знайти вузькі місця в продуктивності і вибрати оптимізацію з найвищими виграшами. Протягом перших двох років, JADE був більше стурбований відповідністю стандартів і з простотою використання, тому не були прийняті повні вимірювання продуктивності: JADE просто досить швидко для додатків, розроблених на ньому.

Однак, як частина більш загальної оцінки JADE і інших FIPA сумісних агентів платформ, деякі вимірювання були виконані на JADE, порівнюючи стандартний IIOP транспорт з локальним подією диспетчерського механізму (PMI не було взято до уваги, з тим щоб мати справедливий порівняння між JADE і іншими агентами платформами FIPA, що не розподілені). Результати показали, що локальна доставка повідомлень завжди була більш ніж в 30 разів швидше, ніж міопія, з рівнем достовірності 99% (тобто щонайменше 99 разів з 100). Крім того, локальна доставка повідомлень було потрібно близько 10 мс, незалежно від загального навантаження на платформу. Це означає, що кластери агента, розташовані в тому ж контейнері агента можуть обмінюватися багато повідомлень, не вносячи великий вклад в загальну робоче навантаження на підсистеми обміну повідомленнями.

Хоча ці результати є обнадійливими, більш комплексними вимірювання ефективності необхідні обидва JADE один і додатків, заснованих на JADE, для того, щоб чітко оцінити JADE з точки зору продуктивності.

2.6 Платформа на основі Go

Go є відкритим вихідним кодом мови програмування, який дозволяє легко побудувати просту, надійну і ефективну програму. Перейти на відміну

від молодого мови, але він отримав швидкозростаюче наступне. Він був розроблений як мова системи, щоб замінити C. Це мова низького рівня з сучасним синтаксисом високого рівня. Вона фокусується на простих багатослівні рішень.

На відміну від Java, Go компілюється в машинний код і виконується безпосередньо. Тому що це не машина VM дуже сильно відрізняється від Java. Це об'єктно-орієнтовані і в той же час функціонального до певної міри. Потоки і черги вбудовані в мову. Вони називаються goroutines і канали. Ви можете натиснути значення в канал і goroutine може тягнути його. Крім цього, функції, такі як клас, інтерфейс, закриття, і т.д., все екранному. Для виконання паралельних розстрілів роботи, Go використовує goroutines, як і в Java, яка використовує потоки ОС. Вони дуже схожі, коли справа доходить до розпаралелювання, тому що обидві мови виконує свої одиниці роботи на нитках OS. Є, однак, різкі відмінності між їх моделями паралельності і як вони справляються з потоками. Канали розподіляються між одиницями роботи. Канал є по суті (необов'язково буфер) буфер FIFO труби. Одиниця роботи може зчитувати або записувати на канал. Перейти вирішила проблему синхронізації між одиницями роботи шляхом повторного обрамлення проблеми: зв'язок по спільно використовуваних каналах, і синхронізований доступ не є необхідним.

3 ПРЕДСТАВЛЕННЯ ТА ОПИС СЦЕНАРІЇВ РОБОТИ МУЛЬТИАГЕНТНОЇ СИСТЕМИ НА ПРИКЛАДАХ

Оскільки багатоагентні системи можуть підтримувати або інтенсивні обчислення або зв'язку ресурсномістких додатки, ми розглянули, як для цілей характеристики. Мета випробувань, описаних нижче, для вимірювання (характеристики)

Поведінка (продуктивність), які можуть бути отримані з Jason, JADE, Go платформи при виконанні цих типів додатків. Зокрема, ми хочемо, щоб виявити, якою мірою платформи Jason, JADE, Go можуть скористатися архітектури многоядерного, який присутній в більшості сучасних процесорів.

3.1 Тест на продуктивність обчислень

Для того, щоб оцінити продуктивність досягаються обчислення ресурсномістких завдань, ми адаптували простий багато-тест процесора, який спочатку був створений для оцінки кілька основних можливостей загальних Java-додатків.

Тест просто вимірює цикли процесора, споживані кожен потік при виконанні арифметичних операцій над структурою даних 109 розміру. Сума повних циклів процесора, спожитих усіх потоки повинна бути дорівнює загальний час виконання, помножене на кількість ядер, які використовуються. Різниця між цими двома значеннями відбувається через накладні витрати, введених конфігурацією віртуальної машини Java і / або ядра операційної системи.

Ми адаптували цей тест для наших платформ шляхом впровадження простих агентів, які виконують обчислення, відповідні кожну нитку в

розглянутому тесті. Ми запустили цей тест з різною кількістю потоків (агенти) для розмірів популяцій в межах від 1 до 1024 агентів.

Ми розрахували співвідношення між загальним часом ЦП ниток і загальне минулий час виконання. Цей фактор показує прискорення (швидкість-вгору), що надається розглянутої конфігурація многоядерной (кількість ядер). Рис. 5.1 показує залежність швидкості вище виходить три реалізації тесту обчислень.

3.2 Тест зв'язку

Для того, щоб оцінити продуктивність досягнута з Джейсоном для зв'язку ресурсномістких завдань, ми застосували два з критеріїв, запропонованих в (Масштабні мультиагентних тестів платформи. У Мов, методологій і інструментів розвитку для мультиагентних систем (LADS 2007). Праці мульти-Agent Логіка, Мови, і організації - Федеративної майстерні, сторінки 192-204).

Ці тести складаються з безлічі агентів, які обмінюються луна-повідомлення, як за допомогою наступних різних комунікаційних моделей. Крім того, існує також контролер агент, який відповідає за синхронізацію початку і завершення виконання.

3.2.1 Тест зв'язоку 1

Цей тест складається з безлічі агентів, як показано на рисунку 4.1. Всі агенти ведуть себе таким же чином, і вони позначені як Agent1 до AgentN (з N означає число агентів, встановлених користувачем). Agenti починає посилати ехо-повідомлення із запитом на Agenti + 1 і чекає луна-відповіді. Потім, вона протікає по колу з Agenti + 2, + 3 Agenti, і так далі. Кожен агент продовжує

посилати повідомлення таким чином, поки він не посилає число повідомлень, вибраних користувачем.

Тест намагається визначити, як платформа поводиться, коли кількість агентів збільшується. Цей тест включає в себе багато обмінів повідомленнями між кожною парою агентів в мультиагентной системі.

Рисунок 4.1 показує досягнуте прискорення на Y-осі, і він показує кількість модельованих агентів на X-осі. Результати, показані на цьому малюнку показано, що для населення до тридцяти чотирьох агентів реалізація Джейсон забезпечує більш високі показники прискорення, ніж Java і Go реалізації. З цієї точки вгору, швидкість вгору залишається більш-менш постійним в Java і Go реалізації і навіть зменшується для реалізації Джейсоном.

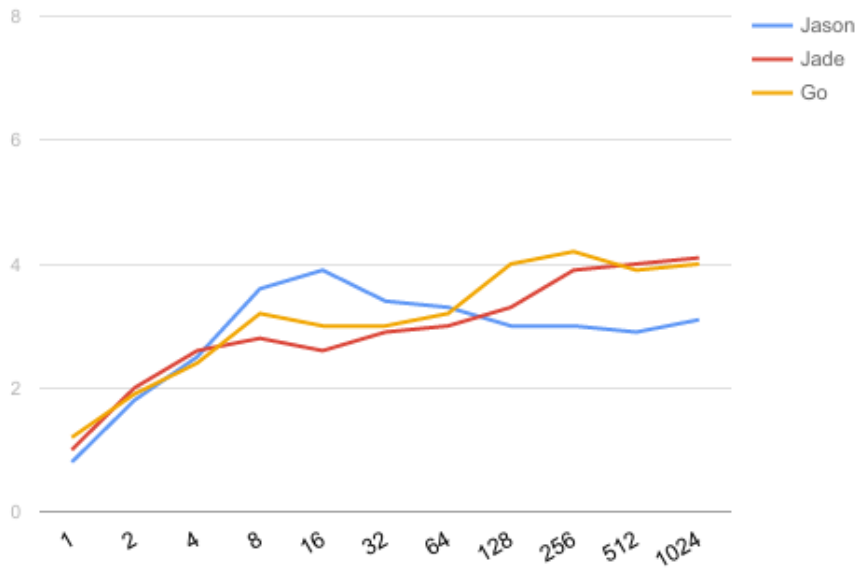

Рисунок 3.1 — Результат для перевірки зв'язку 1

3.2.2 Тест зв'язоку 2

У цьому тесті, існує безліч агентів відправника і набір приймачів агентів (див рис. 3.2). Як число агентів відправника (N) і приймачів агентів (M) є параметри, які будуть встановлені користувачем. Проте, кожен відправник агент посилає повідомлення виключно до відповідного агенту приймача. Навпаки, кожен агент приймача повторює надходять повідомлення до подмножеству відправників, з якого він отримує відлуння-запитів. Таким чином, цей тест може моделювати сценарії, в яких є один або багато моделей трафіку гарячого плями, які генеруються одним або багато чого перевантаженого агента (ів) (наприклад, постачальник послуг (S)).

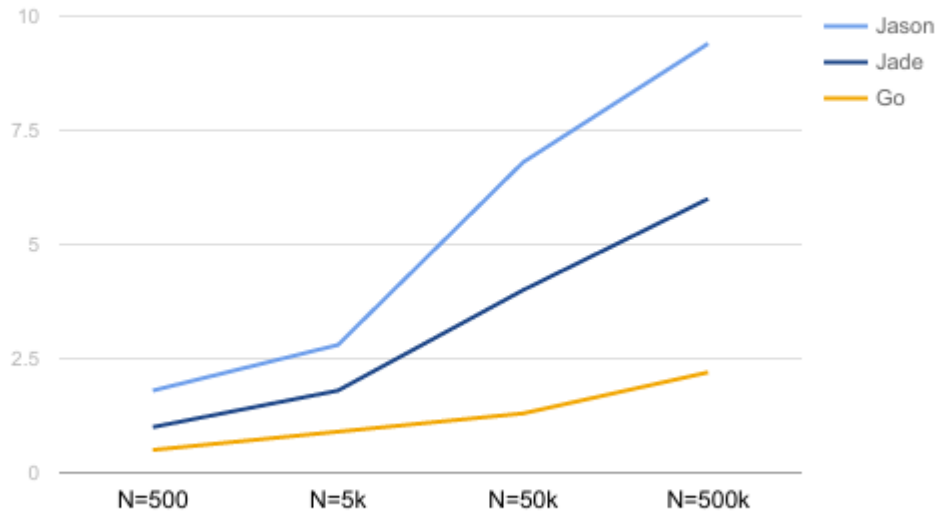

Рисунок 3.2 - Результат тесту для зв'язку 2

При M = 1, мета цього тесту полягає в вимірі платформи пропускної здатності, коли агент має високу швидкість обміну повідомлень. При установці N = M, цей тест перевіряє масштабованість і ефективність,

показавши тим самим, як продуктивність розвивається в великомасштабних системах збалансованих моделей трафіку. Нарешті, для оцінки узгодженості результатів випробувань, при фіксації значення N і значення в діапазоні M від 2 до N результати повинні показати симетричне поведінку з тими, в тесті M = 1.

На рисунку 3.2 показані результати оцінки з точки зору часу відгуку (RTTS) при збільшенні числа агентів при моделюванні. Рисунок 3.2 показує минулий час (в секундах) на Y-осі, і він показує кількість модельованих агентів на X-осі.

4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту для створення мультиагентних систем.

Програмний продукт є тісно пов’язаним з апаратною складовою і спирається на особливості використання конкретних апаратних складових. Так, при зміні однієї з апаратних складових зазвичай необхідно переписувати програмний інтерфейс для її використання.

Нижче наведено аналіз різних варіантів реалізації ПЗ з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Функціонально-вартісний аналіз в соціальному закладі служить інструментом управління процесами, що вимірює вартість надання соціальних послуг. Розрахунок виконується як для функцій, які збільшують цінність послуги, так і для додаткових функцій, які цієї цінності не змінюють. ФВА

досліджує всі можливі виробничі функції (функціональні відносини) з метою найбільш точного розуміння їх питомої ваги (тимчасового, вартісного) в наданні послуг, а також знаходження напрямів модернізації процесів, що виникають в організації, та підвищення їх продуктивності.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

- визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.
- для кожної функції визначаються повні річні витрати й кількість робочих часів.
- для кожної функції на основі оцінок попереднього пункту визначається кількісна характеристика джерел витрат.
- після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки системи аналізу нелінійних нестаціонарних процесів. Оскільки основні проектні рішення стосуються всієї системи, кожна окрема підсистема має їм задовольняти.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

- програмний продукт повинен функціонувати на конкретній апаратній платформі, що проектується і поставляється разом з програмним продуктом;
- необхідно забезпечити мінімальну затримку при передачі даних між наземною станцією і БПЛА;
- необхідно забезпечити шифрування даних для запобігання перехоплення інформації з БПЛА і неможливості підміни інформаційних сигналів;
- необхідно забезпечити можливість протидії РЕБ, та підміні пакетів;
- необхідно швидко реалізувати ПЗ для прототипу для отримання фінансування;
- необхідно передбачати мінімальні витрати на впровадження програмного продукту у рамках проектування програмно-апаратного прототипу.

4.1.1 Обґрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який буде встановлений у наземний і повітряний модулі системи захищеного зв’язку з БПЛА. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір алгоритму шифрування даних;

F_3 – вибір фреймворку для відео стримінгу.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування C;

б) мова програмування C++;

Функція F_2:

а) стандарт AES із 128 бітним ключем;

б) стандарт AES із 256 бітним ключем;

Функція F_3:

а) фреймворк VLC;

б) фреймворк GStreamer.

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

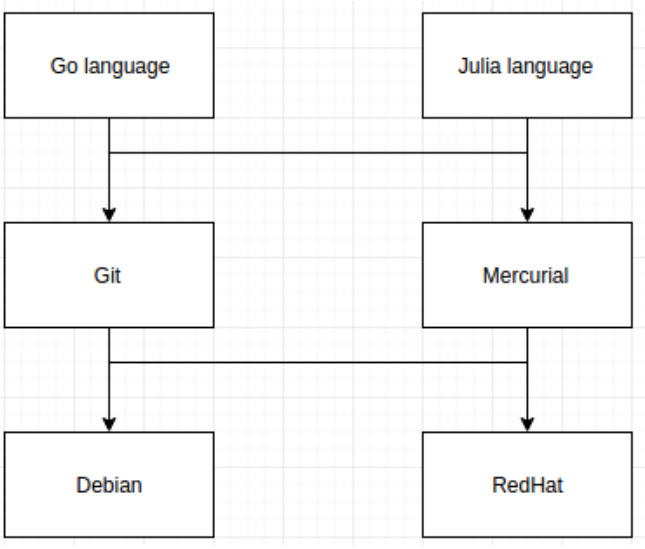

Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті.

Функція *F1:*

Оскільки написання програмної складової на C++ вимагає більше часу на структурування коду і проектування ієрархій, в той час, як C зручно використовувати при програмуванні апаратних драйверів та суміжних з ними інтерфейсів, для швидшого написання програмної частини пріорітетним є вибір мови програмування C.

Функція *F2:*

Надійне шифрування є найбільш пріорітетною складовою захищеного зв’язку. Проте, зважаючи на те, що шифрування AES із 128 бітним ключем на сьогоднішній день є, фактично, незламним, а шифрування 264 бітним ключем займає в середньому на 40% більше часу, оптимальним варіантом шифрування для зменшення затримки при передачі даних є 128 бітний ключ. Якщо з плином часу з’являться можливості теоретичного злому 128 бітного ключа, шифрування буде змінено на 264 бітний ключ.

Функція *F3:*

Обидва фреймворка мають мають досить широкий спектр кодеків та плагінів для реалізації поставленої мети, а також зручні інтерфейси для

програмування нових поагінів на С, тому розглянемо варіанти реалізації програмного продукту із двома можливими фреймворками.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Займає менше часу при написанні коду	Гірша структурованість коду
	Б	Можливість використовувати об’єктно орієнтований підхід і, як наслідок, краща структурованість коду	Займає більше часу при написанні коду і вимагає більшої кількості спеціалістів
F2	А	Більша швидкість шифрування	Злом ключа займає менше часу
	Б	Менша швидкість шифрування	Злом ключа займає більше часу
F3	*А*	Має байндинги під більшіть мов програмування	Має невелику кількість плагінів
	Б	Дуже велика кількість плагінів	Має стабільні байндинги тільки під C++ та Python

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а
2. F1а – F2а – F3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обґрунтування системи параметрів ПП

4.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

- *X1* – час проектування та написання коду;
- *X2* – затримка при передачі даних;
- *X3* – завантаженість процессора мікрокомп’ютера;
- *X4* – час повного відновлення лінії зв’язку після атаки.

X1: Відображає час, за який буде спроектовано і реалізовано ПП що відповідає поставленим вимогам у розрахунку на одного розробника.

X2: Відображає затримку при передачі даних між наземним і повітряним модулем зв’язку з БПЛА.

X3: Відображає завантаженість процессора у режимі функціонування всіх вузлів зв’язку. Еталонним процессором для порівнянь є процессор Raspberry Pi 3 model B.

X4: Відображає час відновлення всіх компонент зв’язку після РЕБ.

4.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	**Умовні позначення**	**Одиниці виміру**	**Значення параметра**		
			гірші	**середні**	**кращі**
Час проектування та написання коду	X1	міс	12	9	5
Затримка при передачі даних	X2	мс	1000	500	150
Завантаженість процесора мікрокомп’ютера	X3	%	95	70	50
Час повного відновлення лінії зв’язку після атаки	X4	с	20	10	5

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

4.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту.

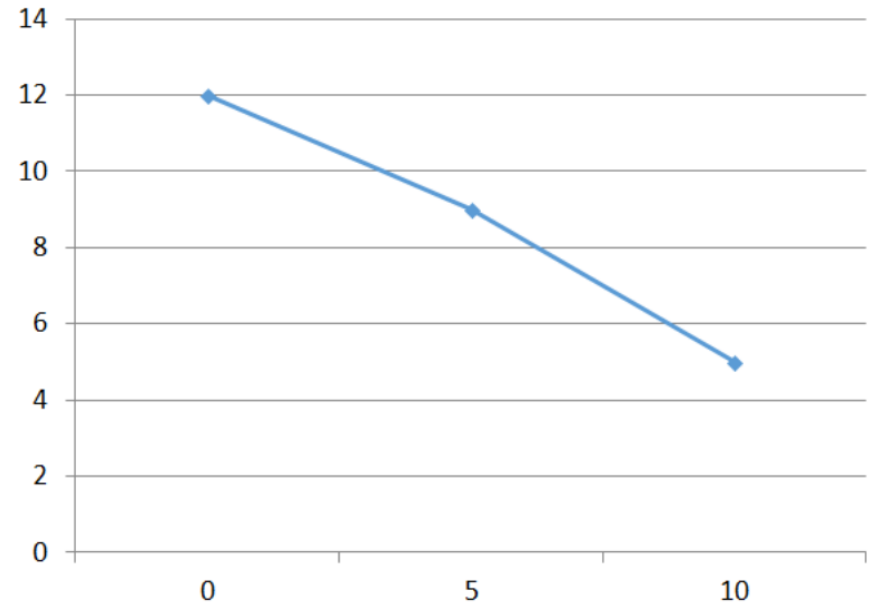

Рисунок 4.2 – X1: Час проектування та написання коду

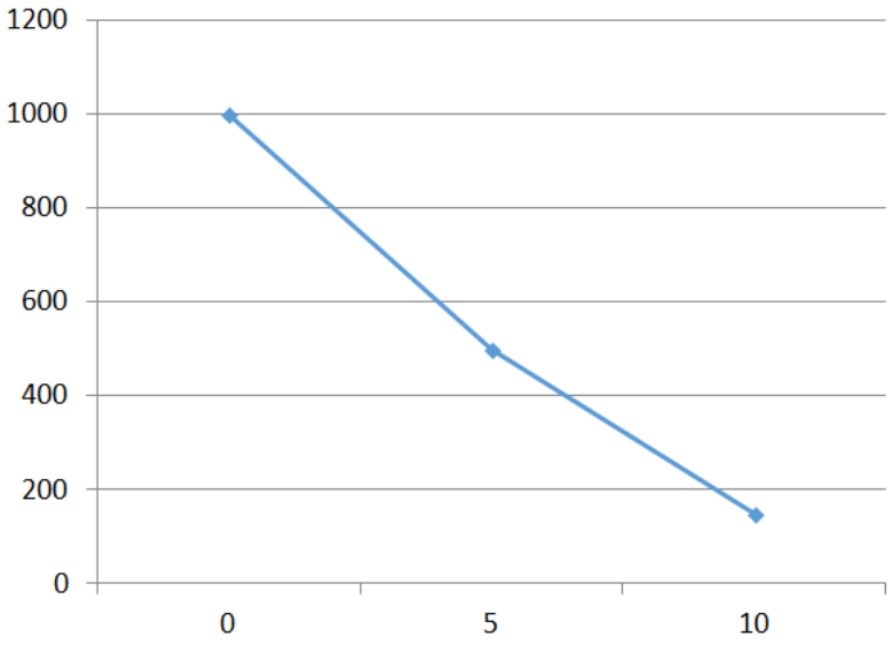

Рисунок 4.3 – X2: Затримка при передачі даних

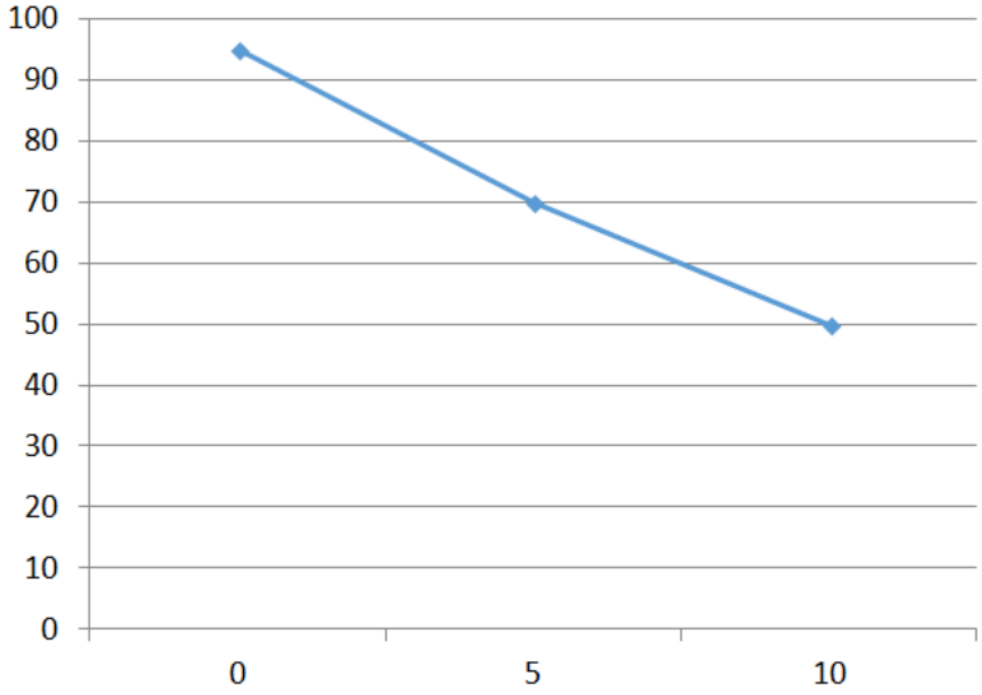

Рисунок 4.4 – X3: Завантаженість процесора мікрокомп’ютера

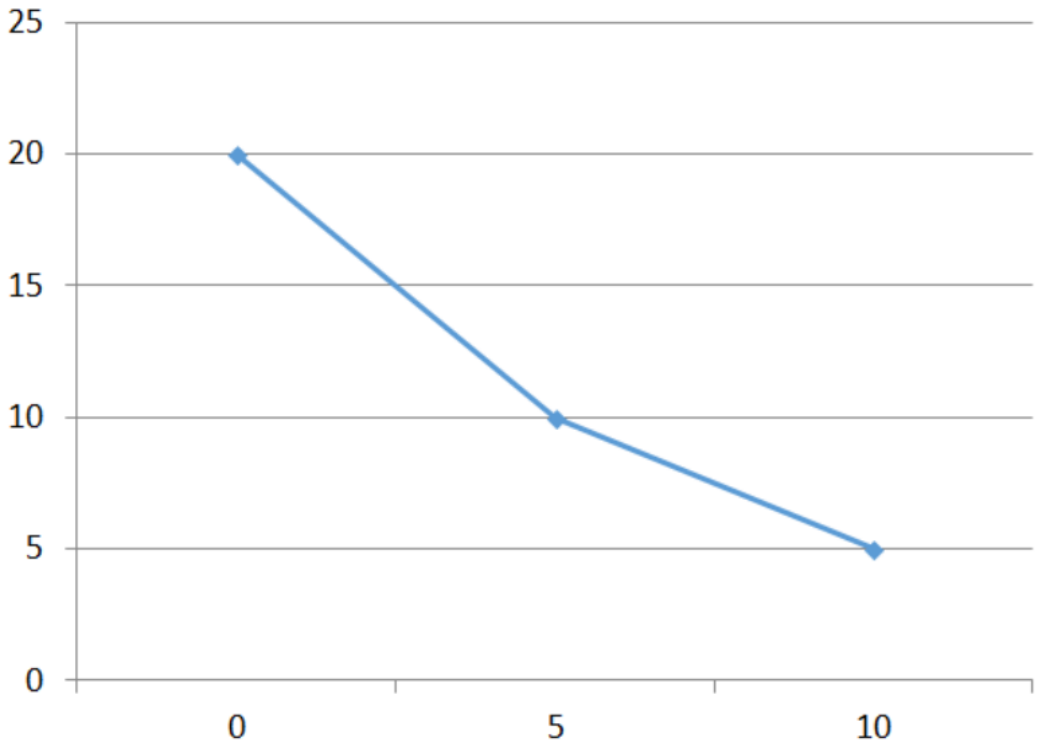

Рисунок 4.5 – X4: Час повного відновлення лінії зв’язку після атаки

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення коефіцієнтів значимості передбачає:

- визначення рівня значимості параметра шляхом присвоєння різних рангів;
- перевірку придатності експертних оцінок для подальшого використання;
- визначення оцінки попарного пріоритету параметрів;
- обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 70,$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 17{,}5.$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S=\sum_{i=1}^{N}\Delta_i^2=201.$$

Порахуємо коефіцієнт узгодженості:

$$W=\frac{12S}{N^2\left(n^3-n\right)}=\frac{12\cdot 201}{7^2\left(4^3-4\right)}=0{,}82>W_k=0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4.

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$aij=\begin{cases}1.5\,при\ Xi>Xj\\1.0\,при\ Xi=Xj\\0.5\,при\ Xi<Xj\end{cases}$$

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
Час проектування та написання	X1	міс	1	2	2	2	1	2	2	12	-5,5	30,25

коду											
Затримка при передачі даних	X2	мс	2	1	1	1	2	1	1	9	-8,5	72,25
Завантаженість процесора мікрокомп’ютера	X3	%	3	3	4	3	4	4	3	24	6,5	42,25
Час повного відновлення лінії зв’язку після атаки	X4	с	4	4	3	4	3	3	4	25	7,5	56,25
	Разом		10	10	10	10	10	10	10	70	0	201

З отриманих числових оцінок переваги складемо матрицю $A=\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{ві}$ за наступними формулами:

$$K_{ві}=\frac{b_i}{\sum_{i=1}^{n} b_i} \quad \text{,де} \quad b_i=\sum_{i=1}^{N} a_{ij} \quad .$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{ві}=\frac{b_i^{'}}{\sum_{i=1}^{n} b_i^{'}}, \quad \text{де} \quad b_i^{'}=\sum_{i=1}^{N} a_{ij} b_j \quad .$$

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	>	<	<	<	>	<	<	<	0,5
X1 і X3	>	>	>	>	>	>	>	>	1,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	>	>	>	>	>	>	>	>	1,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	>	<	>	<	<	>	>	1,5

Як видно з таблиці 4.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X1(Час на вивчення API та написання коду) та X4 (Потенційний об’єм програмного коду) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X2(Об’єм пам’яті для збереження даних) буде найкращим у випадку обрання у F3 варіанта Б і становитиме 10, для варіанту А це значення буде 28.

Абсолютне значення параметра X3(Час отримання данних з хмари) буде найкраще при обрані варіанту А 180, а при обрані варіанту Б воно буде середнім 900.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j)=\sum_{i=1}^{n} K_{\text{в}i,j} B_{i,j},$$

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{ві}$	b_i^1	$K_{ві}^1$	b_i^2	$K_{ві}^2$
X1	1,0	0,5	1,5	1,5	4,5	0,281	16,25	0,275	59,125	0,274
X2	1,5	1,0	1,5	1,5	5,5	0,344	21,25	0,360	77,875	0,361
X3	0,5	0,5	1,0	1,5	3,5	0,219	12,25	0,208	44,875	0,207
X4	0,5	0,5	0,5	1,0	2,5	0,156	9,25	0,157	34,125	0,158
Всього:					16	1	59	1	216	1

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Параметри	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	А	X1	6	9	0,361	3,249
F2	А	X4	10	5	0,207	1,035
F3	А	X2	88	2	0,158	0,316
		X3	900	2	0,274	0,548
	Б	X2	55	9	0,158	1,422
		X3	210	9	0,274	2,466

де n – кількість параметрів; $K_{вi}$ – коефіцієнт вагомості i–го параметра; B_i – оцінка i–го параметра в балах.

За даними з таблиці 4.6 за формулою

$$K_K = K_{ТУ}[F_{1k}] + K_{ТУ}[F_{2k}] + ... + K_{ТУ}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 3,249 + 1,035 + 1,422 + 2,466 = 8,172

K_{K2} = 3,249 + 1,035 + 0,316+ 0,548= 5,148

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки ПП

Програмний продукт, що буде встановлюватись на сервер кафедри вже розроблено, проте необхідно увесь рік тримати під контролем сервер, на якому буде встановлено програмне забезпечення. Отже

$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4$ людино-днів.

$T_1 = 122.4 \cdot 8 = 979.2$ людино-годин;

В розробці беруть участь C/C++ програміст з окладом 12000 грн та системний аналітик з окладом 8000 грн. Визначимо зарплату за годину за формулою:

$$СЧ = \frac{М}{T_m \cdot t} грн.,$$

де М – місячний оклад працівників; T_m – кількість робочих днів тиждень; t – кількість робочих годин в день.

$$СЧ = \frac{11000 + 9000}{2 \cdot 21 \cdot 8} = 59{,}52\, грн.$$

Тоді, розрахуємо заробітну плату за формулою

$$СЗП = C_ч \cdot T_i \cdot КД \ ,$$

де $С_ч$– величина погодинної оплати праці програміста; T_i – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$С_{ЗП} =$ 59,52 * 979.2 · 1.2 = 69900.6 грн на рік.

Відрахування на єдиний соціальний внесок незалежно від групи професійного ризику становить 22%:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 69900.6 * 0.22 = 15378.13$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Працюватиме одна електрона обчислювальна машина цілодобово:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 12000 \cdot 0,2 = 28800$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 28800 \cdot (1 + 0.2) = 34560$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.2 = 34560 * 0,2 = 12669$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 25000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 25000 = 7187,5$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$ – річна норма амортизації; $Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 25000 \cdot 0.05 = 1437,5$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ}$ =($Д_К$ – $Д_В$ – $Д_С$ – $Д_Р$) · $t_З$· $К_В$ = (365 – 104 – 8 – 16) · 8 · 0.9 = 1706.4 годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни. Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ}$ = $Т_{ЕФ}$· $N_С$· $Ц_{ЕН}$ =1706.4 · 0.6 · 1.93819 = 1984.39 грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н$ = $Ц_{ПР}$·0.67 = 25000 · 0,67 = 16750 грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС}$ = 69900.6 + 15378.13 + 7187.5 + 1437.5 + 1984.39 + 16750 = 112638.12 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М-Г}$ = $С_{ЕКС}$/ $Т_{ЕФ}$ = 112638.12 / 1706,4 = 66 грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М-Г} \cdot Т$$

$C_м = 66 * 979.2 = 64627.2$

Накладні витрати складають 67% від заробітної плати:

$$C_н = C_{зп} \cdot 0,67$$

$C_н = 69900.6 * 0.67 = 46833.40$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$$C_{пп} = C_{зп} + C_{від} + C_м + C_н$$

$C_{пп} = 69900.6 + 15378.13 + 63775.29 + 46833.40 = 196739.33$ грн.;

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$K_{ТЕРj} = K_{Кj} / C_{Фj},$$

$K_{ТЕР1} = 8,172 / 196739.33 = 4,15 \cdot 10^{-5}$;

$K_{ТЕР2} = 5,148 / 196739.33 = 2,61 \cdot 10^{-5}$;

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $K_{ТЕР1} = 4,15 \cdot 10^{-5}$.

ВИСНОВОКИ

У цій статті ми представили аналіз результатів трьох різних мов агент програмує, використовуючи набір еталонних тестів, які були недавно запропонованими. Java робить справедливість свого імені і зробив краще майже у всіх випадках, коли розглядалися масштабні коефіцієнти, в той час як Go стояв далеко, як один з істотно більш високої продуктивності, ніж на двох інших мовах в обох минулих час обчислення і сценаріїв. Джейсон, як представник від «важкої» парадигми, не розчарував, уважно стежить на обох аспектах, масштабованості і продуктивності, і навіть перевершуючи в деяких аспектах у порівнянні з цими двома мовами агент програмує, доводячи, що агент-орієнтовані мови програмування можуть виконувати дивно близько до своїх попередників, наскільки зв'язку стурбований.

При розробці агента, який призначений бути введений в промисловість, інженерний підхід у розвитку розглядаються в якості основного фактора, необхідного. Тому мови займуть значну роль, особливо коли число агентів в системі зростає. Таким чином, при розробці системи на базі агентів, методи управління, тестування і повторного використання повинні бути застосовані. У даній статті розглянуті деякі мови / рамки розвитку для розробки систем на основі агентів. Мови в країнах, що розвиваються агентів можуть бути обрані на основі вимог до системи, типу агентів і навколишнього середовища, в якій використовують агенти.

Запропонована архітектура мультиагентної системи для досягнення таких аспектів поведінки, як підтримання дистанції між агентами, узгодження швидкостей і, відповідно, обхід перешкод. Поведінка групи агентів, що задається за допомогою нечіткої логіки вищого типу більш інформативні по відношенню до критеріїв узгодження дистанції і швидкості ніж поведінка, породжувана на основі звичайний нечіткі правила.

Модель МАС є зараз єдиною моделлю, яка пропонує ефективне рішення при розробці додатків, які з значної частини автономних, зокрема, мобільних сутностей, що мають власні цілі, що володіють обмеженими знаннями про інших сутності, структурі системи в цілому і зовнішньому середовищі.

В даний час є сотні розробок - прототипів МАС, ефективно вирішують дуже складні завдання. Вони показують, що модель МАС є перспективною моделлю, яка вже зараз пропонує дуже ефективне напрямки розвитку інформаційних технологій, спеціально орієнтоване на реалізацію

ПЕРЕЛІК ПОСИЛАНЬ

1. Macal C. Agent-based modeling and simulation for exascale computing. / Macal C., North M. // SciDAC Review - Summer 2008. – P. 34–41.
2. Zadeh L.A. Fuzzy Logic Computing With Words / Zadeh L.A. // IEEE Transactions on Fuzzy Systems. – 1996. – Vol. 4. – P. 103–111.
3. Zadeh L.A. The concept of a linguistic variable and its application to approximate reasoning / Zadeh L.A. // Information Sciences. – 1975. – Vol.8, N 8. – P. 199–249, P. 301–357.
4. Zadeh L. A. The concept of a linguistic variable and its application to approximate reasoning / Zadeh L.A. // Information Sciences. – 1975. – Vol.8, N 8. – P. 199–249, P. 301–357.
5. Mendel J.M. Type-2 sets made simple / Mendel J.M., John R.I. // IEEE Trans. on Fuzzy Systems. – 2002. – Vol.10, N 2. – P. 117–127.
6. Karnik N.N. Type-2 fuzzy logic system / Karnik N.N., Mendel J.M., Liang Q. // IEEE Trans. on Fuzzy Systems. – 1999. – Vol.7, N 6. – P. 643–658.
7. Парасюк И.Н. Нечеткие модели мультиагентных систем в распределенной среде / Парасюк И.Н., Ершов С.В. // Проблеми програмування. – 2010. – № 2–3. – С. 330–339.
8. Парасюк И.Н. Моделе-ориентированная архитектура нечетких мультиагентных систем / Парасюк И.Н., Ершов С.В. // Компьютерная математика. – 2010. – № 2. – С.139–149.
9. Mendel J.M. Type-2 fuzzy sets and systems: an overview / Mendel J.M. // IEEE Computational Intelligence Magazine. – 2007. – Vol. 2. – P. 20–29.
10. Wang D. A survey of hierarchical fuzzy systems / Wang D., Zeng X., Keane J. // International Journal of Computational Cognition. – 2006. – Vol. 4, N 1. – P. 18–29.

11. Bellifemine F. Developing Multi-Agent Systems with JADE. / Bellifemine F., Caire G., Greenwood D. // IEEE Computational Intelligence Magazine. – 303 p.
12. Wooldridge M.J. An Introduction to Multi-agent Systems. / Wooldridge M.J. // Cambridge: MIT Press, 2002. – 366 p.
13. Reynolds C.W. Flocks, herds and schools: a distributed behavioural model / Reynolds C.W. // Comput. Graph. – 1987. – Vol.21, N 4. – P. 25–34.