МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ»
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ

КОНСПЕКТ ЛЕКЦІЙ

з дисципліни «Мультиагентні інтелектуальні інформаційні системи»
частина 1

для аспірантів спеціальності 122 Комп’ютерні науки
усіх форм навчання

Луцьк, 2023

УДК 004.8
К 65

Конспект лекцій з дисципліни «Мультиагентні інтелектуальні інформаційні системи, ч.1» для аспірантів усіх форм навчання спеціальності 122 Комп'ютерні науки [Електронний ресурс] / укл. Н.О. Маслова. – Луцьк : ДВНЗ ДонНТУ, 2023. – 60 с.

Конспект лекцій «Мультиагентні інтелектуальні інформаційні системи» складено згідно до затверджених чинних вимог державних освітніх стандартів і рекомендацій для аспірантів спеціальності 122 Комп'ютерні науки. Рекомендовано застосування з методичними вказівками до виконання практичних робіт та в процесі виконання самостійної дослідницької роботи аспірантів.

Укладач: Н. О. Маслова, доцент кафедри ПМІ

Рецензент: С. О. Ковальов, доцент, к.т.н., доцент кафедри електронної техніки

Відповідальний за випуск: Н. О. Маслова, в. о. зав. каф. ПМІ

Затверджено навчально-методичним відділом ДВНЗ ДонНТУ,
протокол № 5 від 28.02.2023 р.

Розглянуто на засіданні кафедри прикладної математики та інформатики,
протокол № 2 від 20.02.2023 р.

Технічний редактор та
розробник програмних прикладів
асп. Нікітенко А.О.

ЗМІСТ

ВСТУП

Конспект лекцій з дисципліни «Мультиагентні інтелектуальні інформаційні системи» розроблено та орієнтовано на аспірантів спеціальності 122 Комп’ютерні науки (галузь знань 12) і спрямовано на надання аспірантам стислого теоретичного базису з навчального курсу.

На сучасному етапі розвитку технологій інтелектуальні елементи застосовуються у найрізноманітніших сферах, таких як електронна комерція, логістика, управління ланцюгами поставок, телекомунікації, охорона здоров'я та виробництво. Мультиагентні систем є технологією та інструментом, який допомагає в аналізі та розробці нових моделей і теорій у великомасштабних розподілених системах, системах паралельної обробки даних, або в системах, орієнтованих на застосування у сферах, де пряма участь людини ускладнена.

Конспект лекцій містить матеріал, який включає основні положення та базові терміни дисципліни, опис та приклади застосування новітніх, сучасних стандартів та технологій зі сфери інтелектуальних систем та агентів. Описує архітектуру та принципи проєктування мультиагентних систем, класифікацію та приклади створення агентів різного типу.

Внаслідок вивчення теоретичного матеріалу курсу аспірант вдосконалить здатність застосовувати сучасні методології, методи та інструменти експериментальних і теоретичних досліджень у сфері комп’ютерних наук, виконувати оригінальні дослідження, досягати наукових результатів, які створюють нові знання у комп’ютерних науках та дотичних до них міждисциплінарних напрямах.

ТЕМА 1. АГЕНТИ І МУЛЬТИАГЕНТНІ СИСТЕМИ

1.1 Агент та його характеристики

Існують різні версії для пояснення поняття «Агента», якщо висловитись більш не формально, то агент – це комп’ютерна сутність у вигляді програми або робота. Агента можна вважати автономним, оскільки він здатний адаптуватися до змін свого середовища. Але якщо висловитись більш формально, то агент – це сутність, яка знаходиться в середовищі та відчуває різні параметри, які використовуються для прийняття рішення на основі мети сутності [1]. На основі цього рішення сутність здійснює необхідні дії щодо навколишнього середовища.

Наведене вище визначення містить чотири ключові слова, які можна уточнити. Сутність – відноситься до типу агента. Агентом може бути програмне забезпечення, наприклад апаратний компонент, термостат, робот або їх комбінація. Середовище – відноситься до місця, де знаходиться Агент. Середовищем може бути мережа у випадку Агентів моніторингу трафіку, програмним забезпеченням коли агент стежить за діями програмних компонентів тощо. Агент використовує інформацію, отриману з середовища для прийняття рішень. Параметри – це різни типи даних, які агент може отримувати з середовища. Наприклад, параметрами футбольного робота-агента є позиція та швидкість членів команди та позиція м’яча. Дія – кожен агент може виконувати дію, яка призводить до деяких змін у середовищі. Наприклад, коли футбольний робот б’є м’яч, положення м’яча змінюється. Агент може виконувати набір окремих або постійних дій. У безперервному наборі дій агент може виконувати необмежену кількість дій.

Але варто врахувати, що зараз немає остаточного визначення поняття "агент". Нижче приведені лише основні з них:

«Агент – це апаратна або програмна сутність, здатна діяти в інтересах досягнення цілей, поставлених користувачем».

«Під агентом можна розуміти самостійну програмну систему, що складається з програм–об’єктів, що має можливість приймати вплив з зовнішнього світу, визначати свою реакцію на цей вплив і відповідно до цього формувати відповідну дію. Такі агенти здатні діяти, «міркувати» і обмінюватися даними один з одним в мережі для формування індивідуальних чи колективних рішень».

За визначенням Крістіана Доннегара (директора по технології компанії Living Systems, що займається створенням систем спільної комерції на основі технології Агентів): «Агенти – програмні об’єкти, які виконують певні попереджувальні та коригувальні дії відповідно до завдань, що делеговані їм людиною».

Існує мінімальний набір базових характеристик довільного Агента, який включає в себе:

- активність – здатність до організації та реалізації дій;
- автономність (напівавтономність) - відносна незалежність від навколишнього середовища або наявність певної «свободи волі», пов'язана з зрошеним ресурсним забезпеченням його поведінки;
- комунікативність – випливає з необхідності вирішувати свої завдання спільно з іншими Агентами та забезпечується протоколами комунікації;
- цілеспрямованість – що передбачає наявність власних джерел мотивації, а ширшому плані, спеціальних інтенціональних характеристик;
- мобільність - можливість переміщатися по навколишньому середовищу.

Середовище має кілька особливостей, які впливають на складність системи на основі Агентів:

1) доступність – означає точність, з якою Агенти можуть сприймати дані з середовища. У доступному середовищі Агенти можуть отримувати точні та актуальні дані з середовища. Наприклад, мережевий брандмауер може перехоплювати весь трафік мережі. Навпаки, у недоступному середовищі Агент відчуває шум і/або неповні дані. Фізичний світ, тобто будь-яка подія на землі, є прикладом недоступного середовища, оскільки датчики, які фіксують дані створюють власний шум, який створює спостережуваний сигнал;

2) детермінізм – стосується передбачуваності результатів дії. У детермінованому середовищі результати передбачувані, наприклад в середовищі, яке може бути змодельоване кінцевим автоматом, Агент точно знає наступний стан для кожної дії. У недетермінованих середовищах результат дії не цілком передбачуваний, оскільки на нього можуть впливати інші фактори, наприклад у грі результат залежить від дій усіх учасників;
3) динамічність – відноситься до змін, які відбуваються в середовищі і не залежать від дій Агентів. Середовище, в якому зміни можуть відбуватися лише внаслідок дії Агентів, вважається статичним. У всіх інших випадках середовище вважається динамічним. Пам'ятайте, що процес прийняття рішень агентами спирається на інформацію, яку сприймають із середовища. Коли навколишнє середовище змінюється, інформація, отримана раніше, може бути неточною. Таким чином, у динамічному середовищі агенти повинні виявляти зміни в стані середовища та оновлювати отриману інформацію, що потребує більше витрат порівняно зі статичними середовища, де початкова отримана інформація може бути використана для прийняття рішень протягом життя агента;
4) безперервність – відноситься до безперервності або дискретності середовища агента. За цією ознакою середовище мультиагентних систем класифікується на дві категорії: безперервне та дискретне. Безперервне середовище впливає на стан агента через безперервну функцію, наприклад, агент, який рухається у фізичному середовищі.

Метою кожного агента є розв'язання призначеного йому завдання з деякими додатковими обмеженнями, наприклад кінцевий термін (англ. deadline). Для досягнення цієї мети агент спочатку отримує параметри середовища. Маючи ці дані, агент може накопичувати знання про навколишнє середовище. Агент також може використовувати знання своїх сусідів. Об'єднання агентів в одну систему називають мультиагентною системою.

1.2 Мультиагентні системи

Мультиагентні системи – це напрямок штучного інтелекту, який використовує систему агентів для вирішення надскладних завдань чи глобальних проблем. Агенти в таких системах взаємопов'язані між собою і мають можливість обміну повідомленнями. У кожного агента є своя мета і завдання, яке він вирішує. Зазвичай у мультиагентних системах використовуються програмні агенти. Тим не менш, складовими мультиагентних систем можуть бути роботи, люди або команди людей [2]. Мультиагентні системи можуть бути використані для вирішення таких проблем, які складно чи неможливо вирішити за допомогою одного Агента чи монолітної системи.

Системи з багатьма агентами з'явилися на перетині теорії систем і розподіленого штучного інтелекту. З одного боку, йдеться про відкриті, активні системи, такі системи, що розвиваються, та в яких головна увага приділяється процесам взаємодії агентів як причин виникнення системи з новими якостями. З іншого боку, досить часто МАС будуються як об'єднання окремих інтелектуальних систем, що засновані на знаннях.

МАС зазвичай складається з таких основних компонентів:

• множина організаційних одиниць, у якій виділяються підмножини Агентів, що маніпулюють підмножинами об'єктів;

• множина завдань;

• середовище – тобто деякий простір, в якому існують агенти і об'єкти;

• множина відносин між агентами;

• множина дій Агентів (наприклад, множина операцій над об'єктами).

Основною формою організації взаємодії між агентами, що характеризується об'єднанням їхніх зусиль для досягнення спільної мети при одночасному розподілі між ними функцій, ролей і обов'язків, є кооперація. У загальному випадку це поняття можна визначити формулою:

кооперація = співпраця + координація дій + вирішення конфліктів.

Під координацією зазвичай розуміють управління залежностями між діями. Комунікація між штучними агентами залежить від обраного протоколу, що представляє собою множину правил, що визначають, як синтезувати значущі і правильні повідомлення.

Фундаментальними особливостями групи, складеної з агентів, що співпрацюють для досягнення спільної мети, є соціальна структура і розподіл ролей між Агентами.

Деякі з переваг, доступних завдяки використанню технології МАС у великих системах [3]:

1) збільшення швидкості та ефективності операції за рахунок паралельних обчислень та асинхронної роботи;
2) витончена деградація системи, коли один або більше агентів виходять з ладу. Таким чином підвищується надійність і міцність системи;
3) масштабованість і гнучкість – агенти можна додавати за потреби;
4) зменшена вартість – окремі агенти коштують набагато менше, ніж централізована архітектура;
5) багаторазове використання – агенти мають модульну структуру, і їх можна легко замінити в інших системах або легше оновити, ніж монолітну систему.

На сьогоднішній день мультиагентні системи використовуються для розробки широкого спектра інформаційних і промислових систем. У промисловості МАС найбільш поширені для вирішення завдань автоматизації управління складними системами, для збору і обробки інформації, в іграх. Мультиагентні технології застосовують в управлінні мобільними ресурсами, а також в таких сферах, як проектування об’єктів, промислове виробництво, фінансове планування та аналіз ризиків, розпізнавання образів, вилучення знань з даних, розуміння тексту і для вирішення інших складних проблем. Так, наприклад, IBM використовує агентів для виробництва напівпровідникових мікросхем, датська суднобудівна компанія – для заварювання отворів у кораблях, а в Японії система на базі агентів виконує функції інтерфейсу оператора надшвидкісного потягу.

МАС можуть застосовуватися як для конструювання і моделювання гнучких виробничих систем, так і для управління реальними системами виробництва (логістика), продажу продукції різного призначення (е – комерції), інтеграції і управління знаннями та наукової роботи. Велике значення у мультиагентному підході має соціальний аспект вирішення сучасних завдань як його концептуальна основа. Такі системи повинні постійно «жити» на сервері підприємства і безперервно брати участь у вирішенні завдань, а не запускатися час від часу, а для цього – забезпечувати користувачу можливість введення нових даних і компонентів. Нарешті, такі системи повинні накопичувати інформацію, отримувати від неї нові знання і в залежності від цього змінювати свою поведінку з часом.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Надайте визначення поняттю «Агент»
2. Сформулюйте базові характеристики агента
3. Вкажіть особливості доступного середовища агента
4. Що таке мультиагентна система?
5. Перелічитье основні компоненти мультиагентної системи
6. Вкажіть переваги технологій МАС при їх застосуванні у великих системах

ЛІТЕРАТУРА [1-3]

ТЕМА 2. ІНТЕЛЕКТУАЛЬНІ АГЕНТИ

Під інтелектуальними агентами зазвичай розуміють програми, яким людина делегує свої функції з метою автоматичного чи напівавтоматичного прийняття рішень. Агенти запам’ятовують потрібну для правильного функціонування інформацію, узагальнюють її, навчаються і можуть самостійно генерувати впливи, що управляють, або видавати корисні рекомендації користувачам [4]. Під інтелектуальним агентом також розуміється програмно або апаратно реалізована система, яка має такі властивості:

- автономність - правильне функціонування незалежно від свого власника та здійснення контролю власних дій, моніторингу внутрішнього стану;
- адаптивність – здатність навчатися і пристосовуватися до умов, що змінюються;
- наявність зобов'язань – виникнення додаткових завдань унаслідок прохання інших агентів;
- активність – вміння доцільно підходити до організації та реалізації дій;
- товариськість – взаємодія та спілкування агентів один з одним;
- реактивність – відповідна інтерпретація стану навколишнього середовища та здатність реагувати на його зміну;
- цілеспрямованість – здатність самостійно знаходити джерела мотивації;
- переконання – змінна частина знань агента, що змінюється у часі залежно стану його внутрішніх показників;
- бажання – схильність до досягнення важливого агента стану;
- наміри – плановані агентом дії для здійснення своїх бажань та/або прагнення виконати свої зобов'язання;
- мобільність – здатність передавати код агента між контекстами, що використовуються (серверне середовище).

Таким чином, агенту необхідно приймати рішення та орієнтуватися в середовищі виконання аналогічно до дій, які зазвичай виконує людина у реальному житті.

Зараз інтелектуальні агенти застосовуються в наступних областях бізнесу:

• управління розподіленими або мережевими підприємствами;

• складна і багатофункціональна логістика;

• віртуальні організації та Інтернет – портали з продажу продуктів і послуг;

• управління навчальним процесом в системах дистанційного навчання;

• компанії з розвиненими дистриб'юторськими і транспортними мережами (наприклад, в Procter & Gamble);

• управління каналами розподілу;

• моделювання побажань користувачів.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке Інтелектуальний Агент?
2. Які властивості є характерними для системи з інтелектуальним агентом?
3. Вкажіть області застосування інтелектуальних Агентів

ЛІТЕРАТУРА [1-4]

ТЕМА 3. КЛАСИФІКАЦІЯ АГЕНТІВ

3.1 Загальна класифікація агентів

Загальна класифікація агентів представлена рисунку 3.1.

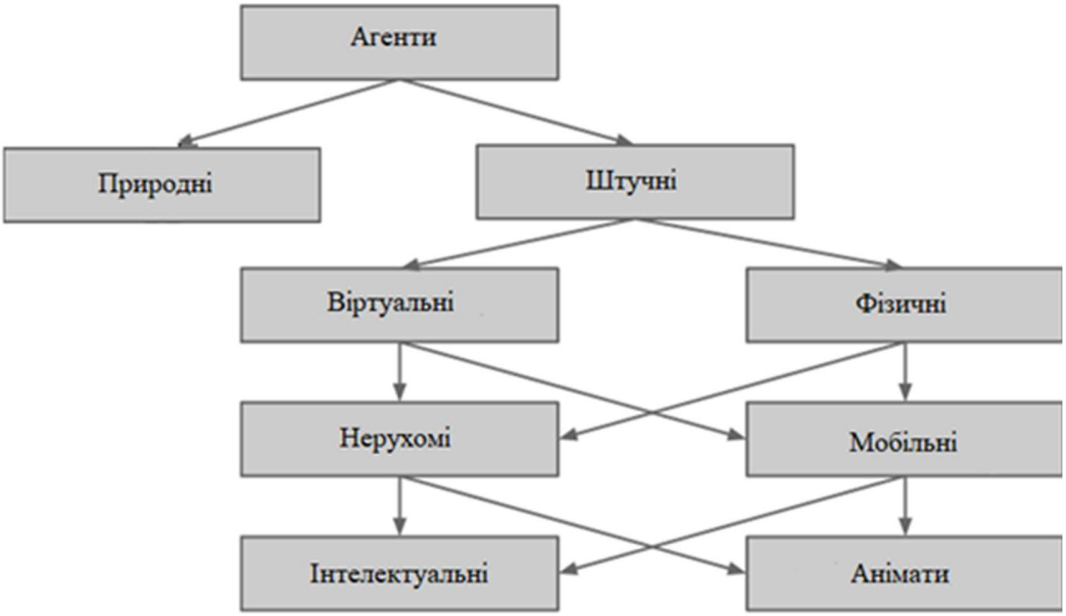

Рисунок 3.1 – Загальна класифікація агентів [5]

3.2 Дихотомії «природне – штучне» та «матеріальне – ідеальне»

Можна запропонувати чимало підстав для побудови класифікацій агентів. Найбільш очевидними є критерії класифікації, пов'язані з полярними шкалами "природне-штучне" та "матеріальне-ідеальне". За першим критерієм виділяються натуральні агенти (тварина, людина, стада тварин, колективи людей) та штучні агенти (роботи, колективи автоматів, складні комп'ютерні програми).

Згідно з другим критерієм, всі штучні агенти поділяються на:

1) матеріальних, фізично існуючих і які працюють у реальному просторі, наприклад, інтегральні роботи, наділені різними засобами “відчуття”, маніпуляторами чи педипуляторами;

2) віртуальних, що існують лише в деякому програмному середовищі (віртуальному просторі), яких нерідко можна уявити як роботів, зайнятих не фізичною, а інформаційною роботою; такі "програмні роботи" (software robots) називають скорочено софтботами (softbots).

3.3 Дихотомії «зосереджене – розподілене» та «нерухоме – рухоме»

Ще одна пара взаємопов'язаних критеріїв класифікації спирається на дихотомію "зосереджене-розподілене" і "нерухоме-рухоме". Прикладом нерухомого агента служить промисловий маніпуляційний робот, а прикладом мобільного – програмний пошуковий агент, який мігрує по комп'ютерній мережі з метою пошуку необхідної інформації.

Важливою основою класифікації є наявність (відсутність) в агентів характеристик навчальності чи адаптивності. У агентів, що навчаються, поведінка заснована на попередньому досвіді.

3.4 Дихотомія «психологічна – біологічна». Поділ за рівнем суб'єктності

Ще одним найважливішим основою класифікації штучних агентів служить прийняття або психологічної, або біологічної метафори під час розгляду природи їх дій (дихотомія “психологічне — біологічне”). В одному випадку, йдеться про трактування агентів як квазісуб'єктів, які самостійно вирішують завдання, що встають перед ними, а в іншому вони уподібнюються найпростішим організмам, що безпосередньо реагують на зміни середовища в інтересах виживання та адаптації. Зокрема, виходячи з біологічної метафори, будувалися М-автомати Н. М. Амосова, а пізніше стали конструюватися "анімати", тобто штучні тварини, які в процесі виживання повинні пристосовуватися до більш складних і ворожих середовищ. [6]. Анімати можуть бути реалізовані як віртуальні агенти (імітація на комп'ютері), і як роботи, що діють у реальному фізичному світі.

У цілому нині, дана типологія агентів тісно пов'язані з класичної проблемою взаємодії “суб'єкт — об'єкт”. Рівень суб'єктності агента безпосередньо залежить від того, чи наділений він символьними уявленнями, потрібними для організації міркувань, або на противагу цьому він працює тільки на рівні образів (субсимвольному), пов'язаних із сенсомоторною регуляцією. Тоді класифікацію агентів (рисунок 3.2) можна побудувати за двома ознаками: а) ступінь розвитку внутрішнього уявлення зовнішнього світу і б) спосіб поведінки.

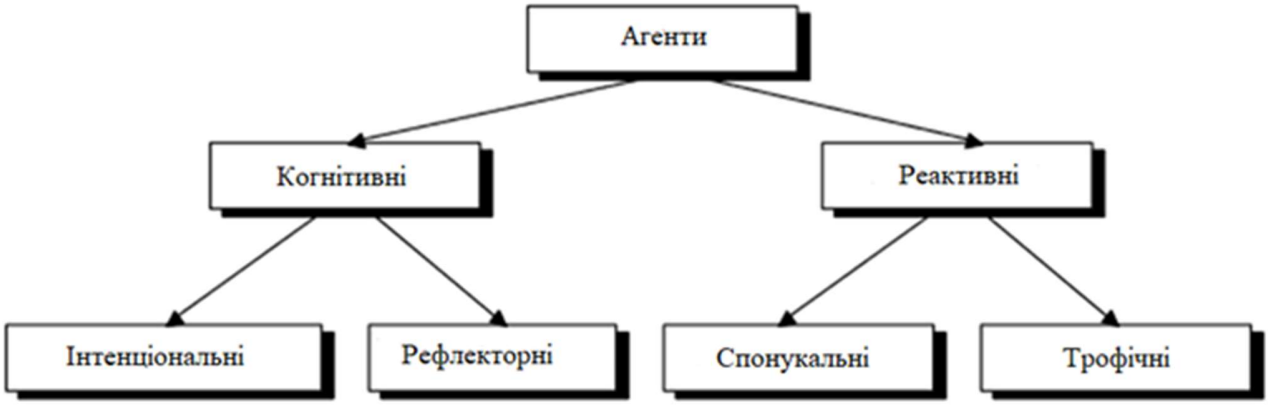

Рисунок 3.2 – Класифікація агентів [5]

За першою ознакою виділяються інтелектуальні (когнітивні, міркуючі, комунікативні, ресурсні) та реактивні агенти. Когнітивні агенти мають добре розвинену і поповнювану символьну модель зовнішнього світу, що досягається завдяки наявності у них бази знань, механізмів вирішення та аналізу дій. Близький термін "розмірковує" зарезервований для позначення агента, який на основі символьної моделі зовнішнього середовища здатний проводити власні міркування, наприклад, використовуючи метод порівняння за зразком, і на їх основі приймати самостійні рішення або виконувати дії, що змінюють середовище.

Невелика відмінність між цими типами інтелектуальних агентів пов'язані з розстановкою акцентів тих чи інших інтелектуальних функцій, отриманні знань про середовище або міркуваннях про можливі дії. У комунікативних агентів внутрішня модель світу перетворюється головним чином на модель спілкування, що

складається з моделей учасників, процесу та бажаного результату спілкування [6]. Нарешті, база знань ресурсного агента містить в основному знання про структуру та стан ресурсів, що визначають різні форми поведінки.

У повноцінного інтелектуального агента обов'язково повинні бути присутніми як мінімум чотири перелічені функції: когнітивна, міркувальна (а, у більш загальному контексті, регулятивна), комунікативна та ресурсна.

У той самий час реактивні агенти не мають ні скільки-небудь розвиненого уявлення довкілля, ні механізму багатокрокових міркувань, ні достатньої кількості власних ресурсів. Звідси випливає ще одна істотна відмінність між інтелектуальними та реактивними агентами, пов'язана з можливостями прогнозування змін довкілля та, як наслідок, свого майбутнього. В силу вищезазначених недоліків реактивні агенти мають дуже обмежений діапазон передбачення. Вони практично не здатні планувати свої дії, оскільки реактивність у чистому вигляді означає таку структуру зворотного зв'язку, яка не містить механізмів прогнозу. Тоді як інтелектуальні агенти, завдяки багатим внутрішнім уявленням зовнішнього середовища та можливостям міркувань, можуть запам'ятовувати та аналізувати різні ситуації, передбачати можливі реакції на свої дії, робити з цього висновки, корисні для подальших дій і, в результаті, планувати свою поведінку. Саме розвинені когнітивні та деліберативні здібності дозволяють таким агентам будувати віртуальні світи, працюючи в яких вони формують плани дій.

Інтелектуальні агенти, будучи значно автономнішими за реактивні, мають куди яскравіше виражену індивідуальність і характеризуються доцільною поведінкою у співтоваристві агентів, а також прагненням використовувати ресурси інших агентів для досягнення власних цілей. У той же час, реактивні агенти, як це видно з їхньої назви, працюють в основному на рівні стимульно-реактивних зв'язків, володіючи дуже бідною індивідуальністю і сильною залежністю від зовнішнього середовища (спільноти агентів). Результати порівняльного аналізу реактивних та когнітивних агентів представлені в таблиці 3.1.

Таблиця 3.1 – Порівняльний аналіз агентів

Характеристики	Когнітивні агенти	Реактивні агенти
Внутрішня модель зовнішнього світу	Розвинута	Примітивна
Міркування	Складні та рефлексивні міркування	Прості одношагові міркування
Мотивація	Розвинута система мотивації, яка включає в собі переконання, бажання, наміри	Найпростіші спонукання, пов'язані з виживанням
Пам’ять	Має	Не має
Реакція	Повільна	Швидка
Адаптивність	Мала	Висока
Модульна архітектура	Має	Не має
Склад МАС	Невелика кількість автономних агентів	Велика кількість агентів, які залежать один від одного

3.5 Способи поведінки

Далі, на кшталт поведінки інтелектуальні агенти діляться на інтенціональних і рефлекторних, а реактивні — на спонукуваних (імпульсивних) і трофічних. Більшість інтелектуальних (когнітивних) агентів можна віднести до інтенціональних. Подібні агенти мають власні механізми мотивації. Це означає, що в них так чи інакше моделюються внутрішні переконання, бажання, наміри та мотиви, що породжують цілі, які визначають їх дії. У свою чергу, модульні або рефлекторні агенти не мають внутрішніх джерел мотивації та власних цілей, а їхня поведінка характеризується найпростішими (однокроковими) висновками чи автоматизмами.

У свою чергу, реактивні агенти містять якби скомпільовані знання про необхідні дії: їм не треба будувати докладне внутрішнє уявлення зовнішнього середовища, оскільки цілком достатніми виявляються реакції на набір ситуацій, що пред'являються, тобто. характер їх реакцій визначається лише поточною інформацією. За складністю цих реакцій та походження джерел мотивації реактивні агенти поділяються на імпульсивних та трофічних агентів. У разі трофічних агентів поведінка визначається найпростішими трофічними зв'язками (типу "хто кого їсть"). Фактично воно зводиться до відповіді стимули, що надходять із довкілля (власних мотивів і цілей немає), т.е. повністю визначається її локальним станом. Тим часом, реактивні агенти можуть мати примітивний механізм мотивації, що штовхає їх на виконання завдання, наприклад, задоволення набору життєвих потреб.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Наведіть загальну класифікацію Агентів та прокоментуйте наведене
2. Поясніть сутність дихотомії «природне-штучне»
3. Назвіть відомі вам критерії класифікації Агентів
4. Які особливості характеризують критерій «матеріальне – ідеальне»?
5. Поясніть сутність дихотомії «природне-штучне»
6. Проведіть класифікацію агентів за ступенем уявлення про зовнішній світ
7. Опишіть особливості класифікації агентів за способом поведінки
8. Що таке реактивний агент та які особливості він має?

ЛІТЕРАТУРА [5,6]

ТЕМА 4. МОБІЛЬНІ АГЕНТИ

Мобільні агенти — це автономні програми, які можуть переміщатися від комп'ютера до комп'ютера в мережі в час і в місця, які вони вибирають. Стан запущеної програми зберігається шляхом передачі до місця призначення. Програма відновлюється на місці призначення, продовжуючи свою обробку зі збереженим станом. Вони можуть забезпечити зручну, ефективну та надійну структуру для впровадження розподілених програм і інтелектуальних середовищ з кількох причин, включаючи покращення затримки та пропускної здатності програм клієнт-сервер і зменшення вразливості до відключення мережі. Насправді мобільні агенти мають ряд переваг у розробці різноманітних послуг у розумних середовищах на додаток до розподілених програм [7].

Зниження витрат на зв'язок. Розподілені обчислення потребують взаємодії між різними комп'ютерами через мережу. Затримка та мережевий трафік взаємодії часто серйозно впливають на якість і координацію двох програм, що працюють на різних комп'ютерах. Як ми можемо бачити на рисунку 4.1, якщо одна з програм є мобільним агентом, вона може перейти на комп'ютер, на якому запущена інша, і спілкуватися з нею локально. Тобто технологія мобільного агента дозволяє віддаленому зв'язку працювати як локальний зв'язок.

Асинхронне виконання. Після переходу на комп'ютер на стороні призначення мобільному агенту не потрібно взаємодіяти з комп'ютером на стороні джерела. Таким чином, навіть якщо джерело можна вимкнути або мережу між одержувачем і джерелом можна від'єднати, агент може продовжувати обробку в одержувачі. Це корисно під час нестабільного зв'язку, включаючи бездротовий зв'язок, у розумних середовищах.

Пряме маніпулювання. Мобільний агент локально виконується на комп'ютері, який він відвідує. Він може мати прямий доступ і керувати обладнанням комп'ютера, якщо комп'ютер це дозволяє. Це корисно для керування

мережею, зокрема для виявлення та усунення збоїв пристрою. Встановлення мобільного агента поблизу системи реального часу може запобігти затримкам, викликаним перевантаженням мережі.

Легка розробка розподілених програм. Більшість розподілених програм складається принаймні з двох програм, тобто програми на стороні клієнта та програми на стороні сервера, і часто містять запасні коди для зв'язку, включаючи виняткову обробку. Однак, оскільки мобільний агент сам може переносити свою інформацію на інший комп'ютер, ми можемо написати лише одну програму для визначення розподілених обчислень. Програма мобільного агента не повинна визначати зв'язок з іншими комп'ютерами. Тому ми можемо легко модифікувати автономні програми як програми мобільних агентів.

Як ми можемо бачити на рисунку 4.1, мобільні агенти можуть зберігати себе за допомогою постійного сховища, дублювати себе та мігрувати на інші комп'ютери під своїм власним контролем, щоб вони могли підтримувати різні типи обробки в розподілених системах.

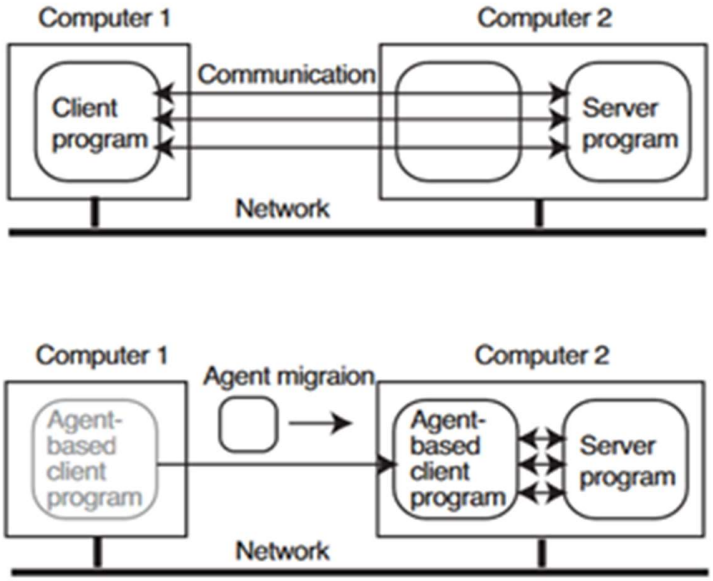

Рисунок 4.1 – Зменшена комунікація [5]

Хоча не всі додатки для розподілених систем потребуватимуть мобільних агентів, існує багато інших додатків, які знайдуть мобільні агенти як

найефективнішу техніку для реалізації всіх або частини їхніх завдань. Технологію мобільного агента можна розглядати як тип технології програмного агента, але вона не завжди вимагає пропонувати інтелектуальні можливості, наприклад, реактивну, проактивну та соціальну поведінку, які є характеристиками існуючих технологій програмного агента. Це пов'язано з тим, що ці можливості, як правило, великі з точки зору масштабу та обробки, і жоден мобільний агент не повинен споживати надмірні обчислювальні ресурси, такі як процесори, пам'ять, файли та мережі, у своїх пунктах призначення. Крім того, технологія є просто підходом до реалізації розподілених систем, а не інтелектуальних систем.

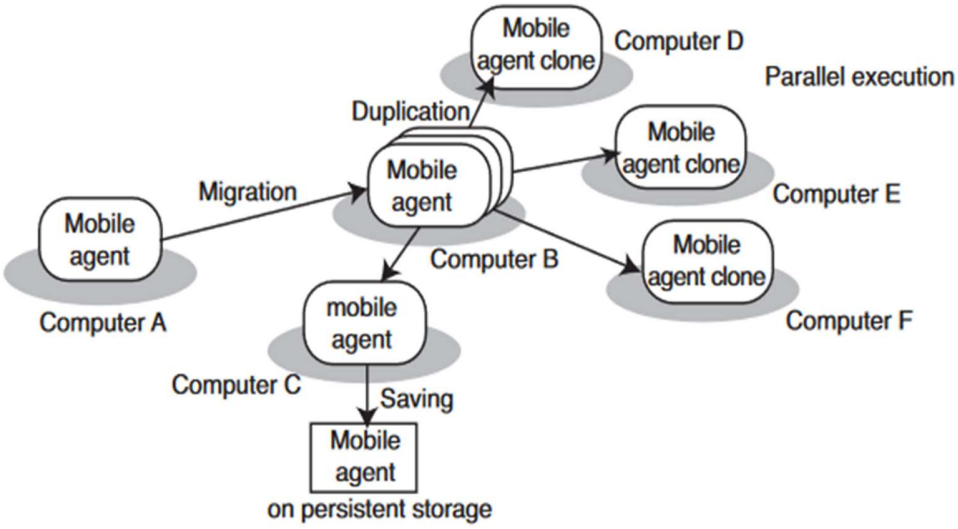

Рисунок 4.2 – Функції мобільних агентів у розподіленій системі [5]

Динамічне розгортання програмного забезпечення. Мобільні агенти корисні як механізм для розгортання програмного забезпечення, оскільки вони можуть вирішувати місця призначення, а їх код і дані можуть динамічно розгортатися там лише тоді, коли вони потрібні. Це корисно в розумних середовищах, оскільки вони складаються з комп'ютерів, обчислювальні ресурси яких обмежені.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що розуміють під поняттям «мобільний агент»?

2. Вкажіть основні переваги застосування мобільних агентів

3. Чи можна стверджувати, що всі додатки для розподілених систем потребуватимуть застосування мобільних агентів?

4. Які функції можуть виконувати агенти у розподіленій системі?

5. Яким чином застосування мобільних агентів може сприяти розробці розподілених програм?

ЛІТЕРАТУРА [7]

ТЕМА 5. АБСТРАКТНІ АРХІТЕКТУРИ АГЕНТІВ

Коли ми говоримо про архітектуру, ми маємо на увазі фреймворк, на основі якого можуть бути побудовані додатки. Архітектури зазвичай визначаються для підтримки певного типу проблем, таких як надійність або робота в реальному часі. Архітектури зазвичай визначаються з точки зору перспективи. Ця перспектива може бути з функціональної точки зору, з точки зору коду або з точки зору користувача.

Архітектура агентів, як і архітектура програмного забезпечення, формально є описом елементів, з яких побудована система, і способу, в який вони взаємодіють. Крім того, ці елементи можуть бути визначені з шаблонів з певними обмеженнями.

Існує ряд загальних архітектур, які мають назви "труба-фільтр" (Pipe-and-Filter) або багаторівнева архітектура. Зверніть увагу, що вони визначають взаємозв'язки між компонентами. Pipe-and-Filter визначає модель, в якій дані переміщуються через набір з одного або декількох об'єктів, які виконують перетворення. Багаторівнева просто означає, що система складається з набору шарів, які забезпечують певний набір логічної функціональності, і що зв'язок зазвичай обмежується шарами, які прилягають один до одного. З точки зору архітектури агентів, можуть існувати шаблони, які підтримують розробку і роботу агентів. Наприклад, можуть існувати компоненти для забезпечення зв'язку між агентами. Інші компоненти можуть підтримувати сприйняття (перегляд оточення агента), а також дії (маніпулювання оточенням). Ці типи компонентів спрощують завдання розробки для розробників агентів, дозволяючи їм сконцентруватися на їх конкретному завданні, а не на загальних проблемах навколишнього середовища. Слід зазначити, що архітектура може бути застосована на декількох рівнях до агентських систем.

Залежно від цілей застосування агента, існують різні архітектури агентів, які можуть допомогти в рішення різноманітних задач. Нижче представлені деякі з основних типів архітектур і додатків, для яких вони можуть бути використані.

5.1 Реактивна архітектура

Реактивна архітектура є найпростішою архітектурою для агентів. У цій архітектурі поведінка агента є просто відображенням між стимулом і реакцією. Агент не має навичок прийняття рішень, тільки реакції на середовище, в якому він існує

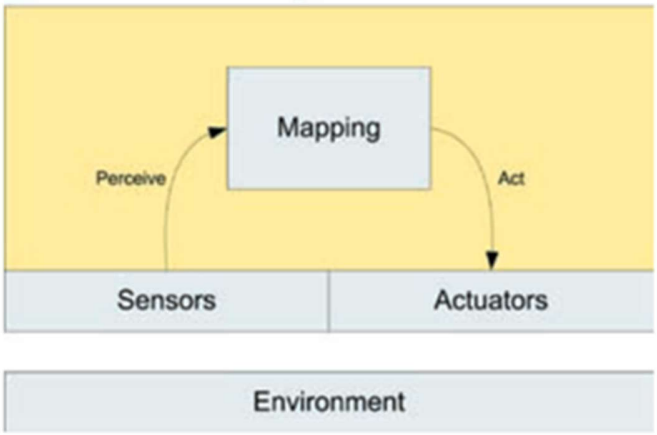

Рисунок 5.1 – Реактивна архітектура визначає простого агента [7]

Як показано на рисунку 5.1, агент просто зчитує навколишнє середовище, а потім зіставляє стан середовища з однією або декількома діями. З огляду на навколишнє середовище, більш ніж одна дія може бути доречною. Перевагою реактивних архітектур є те, що вони надзвичайно швидкі. Така архітектура може бути легко реалізована в апаратному забезпеченні або швидко реалізована в програмному забезпеченні пошуку. Недоліком реактивних архітектур є те, що вони застосовуються лише до простих середовищ. Послідовності дій вимагають наявності стану, який не закодований у функції відображення.

5.2 Деліберативна архітектура

Деліберативна архітектура, як випливає з назви – це архітектура, яка включає в себе певні роздуми над дією, яку потрібно виконати, враховуючи поточний набір вхідних даних. Замість того, щоб зіставляти датчики безпосередньо з виконавчими механізмами, деліберативна архітектура розглядає датчики, стан, попередні результати заданих дій та іншу інформацію для того, щоб вибрати найкращу дію для виконання.

Механізм вибору заходів, як показано на рисунку 5.2, не визначений. Це пов'язано з тим, що це можуть бути різні механізми, включаючи виробничу систему, нейронну мережу або будь-який інший інтелектуальний алгоритм. Перевага деліберативної архітектури полягає в тому, що вона може бути використана для вирішення набагато складніших проблем, ніж реактивна архітектура. Вона може здійснювати планування, виконувати послідовності дій для досягнення мети. Недоліком є те, що вона повільніша за реактивну архітектуру через необхідність обмірковування для вибору дії.

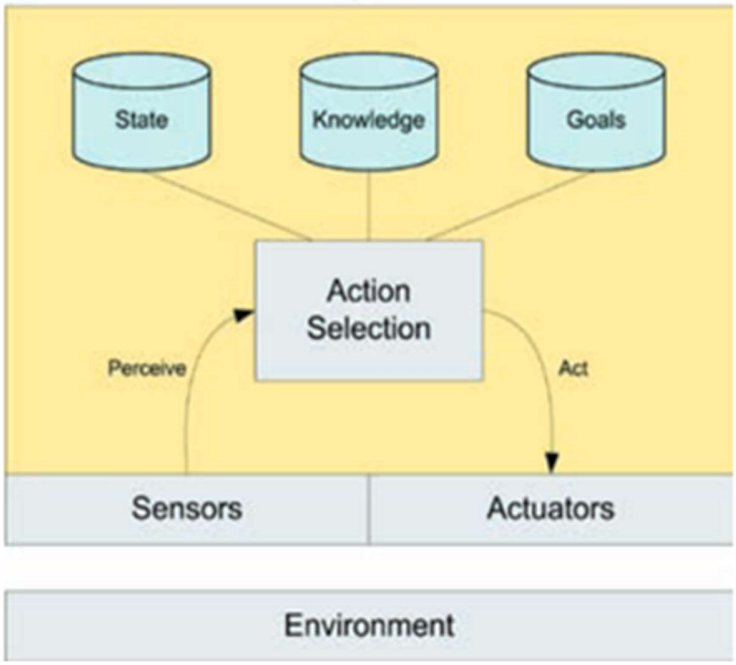

Рисунок 5.2 – Архітектура деліберативного агента обмірковує свої дії [7]

5.3 Архітектура дошок

Архітектура дошки є дуже поширеною архітектурою, яка також є вельми цікавою. Першою архітектурою дошки була HEARSAY-II, яка була системою розуміння мови. Ця архітектура працює навколо глобальної робочої області, яка називається дошка. Дошка є спільною робочою зоною для декількох агентів, які спільно працюють над вирішенням певної задачі. Таким чином, дошка містить інформацію про навколишнє середовище, а також проміжні результати роботи кооперативних агентів.

Приклад, показаний на рисунку 5.3, ілюструє, як архітектура дошки може бути застосована до агентської системи. У цьому прикладі два окремі агенти використовуються для відбору проб середовища через доступні датчики (сенсорний агент), а також через доступні виконавчі механізми (агент дії).

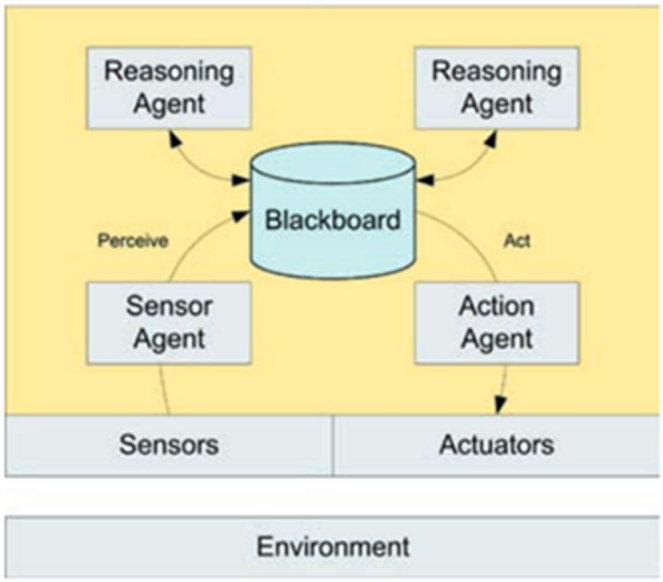

Рисунок 5.3 – Архітектура дошки підтримує мультиагентне вирішення завдань [7]

Дошка містить поточний стан середовища, який постійно оновлюється агентом-сенсором, і коли дія може бути виконана (як зазначено на дошці), агент дії перетворює цю дію в управління виконавчими механізмами. Управління системою агентів забезпечується одним або декількома міркуючими агентами. Ці агенти працюють разом для досягнення цілей, які також будуть міститися на дошці. У

цьому прикладі перший міркуючий агент може реалізовувати поведінку визначення цілей, де другий міркуючий агент може реалізовувати частину планування.

Архітектура дошки, з її глобально доступною робочою зоною, легко реалізується за допомогою багатопотокової системи. Кожен агент стає одним або декількома системними потоками. З цієї точки зору, архітектура електронної дошки є дуже поширеною для агентних та неагентних систем.

5.4 Архітектура Переконання-Бажання-Наміру (Belief-Desire-Intention)

Belief-Desire-Intention (BDI), що розшифровується як "Переконання-Бажання-Намір", – це архітектура, яка слідує теорії людського мислення за визначенням Майкла Братмана. Переконання представляє погляд на світ з боку агента (те, що він вважає станом середовища, в якому він існує). Бажання – це цілі, які визначають мотивацію агента (чого він хоче досягти). Агент може мати численні бажання, які повинні бути узгодженими. Нарешті, наміри вказують на те, що агент використовує переконання і бажання для того, щоб вибрати одну або декілька дій для того, щоб задовольнити бажання.

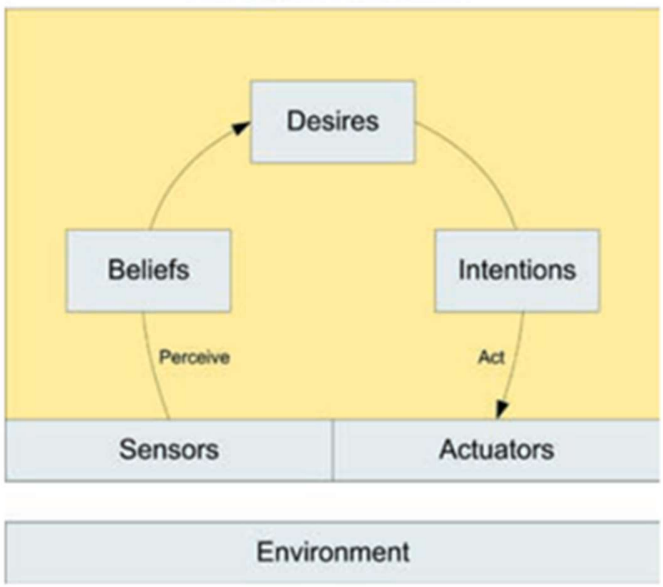

Рисунок 5.4 – Архітектура BDI прагне моделювати ментальні установки [7]

5.5 Гібридна архітектура

Як і у випадку з традиційною архітектурою програмного забезпечення, більшість архітектур є гібридами. Наприклад, архітектура мережевого стеку складається з архітектури "труба-фільтр" та багаторівневої архітектури. Цей же стек також поділяє деякі елементи архітектури дошки, оскільки існують глобальні елементи, які є видимими і використовуються кожним компонентом архітектури. Те ж саме можна сказати і про архітектури агентів. Виходячи з потреб агентської системи, для задоволення цих потреб можуть бути обрані різні архітектурні елементи.

5.6 Мобільна архітектура

Мобільна архітектура надає агентам можливість мігрувати між хостами. Як показано на рисунку 5.5, архітектура агента включає елемент мобільності, який дозволяє агенту мігрувати з одного хоста на інший. Агент може мігрувати на будь-який хост, який реалізує мобільний фреймворк.

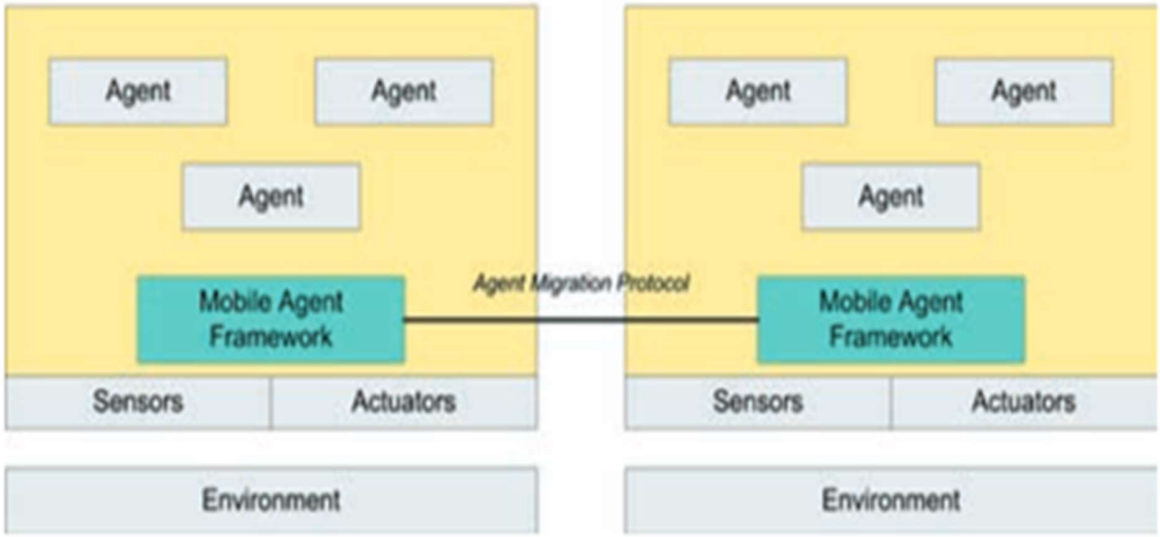

Рисунок 5.5 – Фреймворк для мобільних агентів підтримує мобільність агентів [7]

Фреймворк мобільних агентів забезпечує протокол, який дозволяє зв'язок між хостами для міграції агентів. Ця структура також вимагає певної автентифікації та безпеки, щоб уникнути перетворення структури мобільних агентів на канал розповсюдження вірусів.

Архітектура мобільних агентів має переваги, оскільки підтримує розробку інтелектуальних розподілених систем. Але розподілена система, яка є динамічною, і конфігурація та завантаження якої визначається самими агентами самими агентами.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Поясніть сутність поняття «архітектура агентів»
2. Вкажіть основні типи архітектури агентів
3. Вкажіть особливості реактивної архітектури
4. Що таке деліберативна архітектура?
5. Які особливості має архітектура дошок?
6. Наведіть характерну схему архітектури Переконання-Бажання-Наміру
7. У чому особливості мобільної архітектури?

ЛІТЕРАТУРА [7]

ТЕМА 6. ЗАСОБИ ТА МОВИ КОМУНІКАЦІЇ АГЕНТІВ

6.1 Засоби комунікації агентів

Існують декілька міжнародних підходів до створення мультиагентних систем, найбільш відомі з них:

1) Стандарт OMG MASIF (Object Management Group Mobile Agent System interoperability), в основі якого лежить поняття мобільний агент та який орієнтований на створення умов для міграції мобільних агентів між мультиагентними системами за допомогою стандартизованих інтерфейсів CORBA IDL;

2) Специфікації FIPA (Foundations for Intelligent Physical Agents), що ґрунтуються на припущенні про інтелектуальність агента та орієнтуються на забезпечення можливості взаємодії інтелектуальних агентів через стандартизовану комунікацію агентів і мови контенту;

3) Стандарти DAPRA (Defense Advanced Research Projects Agency), розроблені дослідницьким підрозділом Пентагону, який ініціював роботу з розподілу знань (Knowledge Sharing Effort), в результаті якої мови програмування агентів були поділені на синтакс (syntax), семантику (semantics) та прагматику (pragmatics).

На сьогодні найбільше поширення набула еталонна модель керування агентами, що базується на специфікаціях FIPA. Фонд інтелектуальних фізичних агентів (FIPA) – це міжнародна некомерційна асоціація компаній і організацій, які спільно працюють над створенням специфікацій загальних агентських технологій. FIPA передбачається не просто як технологія для однієї програми, а як загальна технологія для різних областей застосування, і не просто як незалежні технології, а як набір базових технологій, які можуть бути інтегровані розробниками для створення складних систем з високим ступенем сумісності.

FIPA базується на двох основних припущеннях. Перший полягає в тому, що час для досягнення консенсусу та завершення стандарту не повинен бути довгим, і, головним чином, він не повинен виступати гальмом прогресу. По-друге, слід вказувати лише зовнішню поведінку компонентів системи, залишаючи деталі реалізації та внутрішню архітектуру розробникам агентів. Насправді, внутрішня архітектура JADE є запатентованою, навіть якщо вона відповідає інтерфейсам, визначеним FIPA.

Перші вихідні документи FIPA, які називаються специфікаціями FIPA97, визначають нормативні правила, які дозволяють суспільству агентів взаємодіяти, тобто ефективно існувати, працювати та керувати ним. Перш за все, вони описують еталонну модель агентської платформи, як показано на рисунку 6.1.

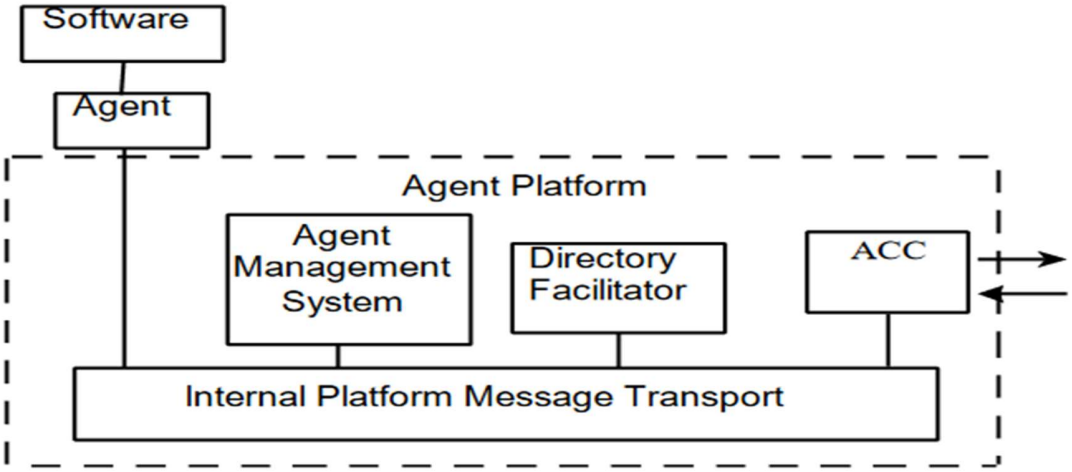

Рисунок 6.1 – Еталонна модель агентської платформи FIPA [8]

По суті, вона визначає ролі деяких ключових агентів, необхідних для управління платформою, і визначає мову та онтологію контенту керування агентом. На платформі агента було визначено три ключові обов’язкові ролі:

- система керування агентами (Agent Management System AMS) — це агент, який здійснює наглядовий контроль за доступом до платформи та її використанням; він відповідає за автентифікацію агентів-резидентів і контроль реєстрацій;

- канал зв’язку агента (Agent Communication Channel (ACC)) — це агент, який забезпечує шлях для базового контакту між агентами всередині та поза платформою; це метод зв’язку за замовчуванням, який пропонує надійну, упорядковану та точну службу регулярних повідомлень; він також повинен підтримувати IIOP для взаємодії між різними агентськими платформами.
- посередник каталогу (Directory Facilitator DF) — це агент, який надає службу жовтої сторінки для платформи агента. Зауважте, що немає жодних обмежень щодо фактичної технології, яка використовується для реалізації платформи: платформа на основі електронної пошти, на основі CORBA, багатопотокові програми Java, усі можуть бути реалізаціями, сумісними з FIPA.

Зазвичай, стандарт визначає також мову комунікації агента ACL (Agent Communication Language). Комунікація агентів базується на передачі повідомлень, коли агенти спілкуються шляхом формулювання та надсилання окремих повідомлень один одному. FIPA ACL визначає стандартну мову повідомлень, встановлюючи кодування, семантику та прагматику повідомлень. Стандарт не встановлює конкретного механізму внутрішнього транспортування повідомлень. Натомість, оскільки різні агенти можуть працювати на різних платформах і використовувати різні мережеві технології, FIPA вказує, що повідомлення, які транспортуються між платформами, повинні бути закодовані в текстовій формі. Передбачається, що агент має певні засоби передачі цієї текстової форми.

Стандарт підтримує загальні форми міжагентних розмов через специфікацію протоколів взаємодії, які є шаблонами повідомлень, якими обмінюються два або більше агентів. Такі протоколи охоплюють діапазон від простих протоколів запитів запитів до добре відомого протоколу мережевих переговорів контракту та англійських і голландських аукціонів.

Інші частини стандарту FIPA визначають інші аспекти, зокрема інтеграцію програмного забезпечення агента, мобільність і безпеку агента, онтологічну службу та зв’язок між людиною та агентом.

6.2 Мови комунікації агентів

Важливим елементом під час створення мультиагентних систем є мова комунікації агентів – ACL (див.табл.6.1), яка визначає типи повідомлень, якими можуть обмінюватись агенти. У рамках парадигми комунікації між агентами кооперація між ними досягається за рахунок ACL, мови контенту та онтології, які визначають набір базових концепцій, що використовуються в повідомленнях кооперації. Онтологія виступає синонімом поняття API (Application Programming Interface), тобто вона визначає конкретний інтерфейс інтелектуальних агентів.

Таблиця 6.1 – Структура повідомлення ACL

Елемент	Опис
Performative	Перформатив
Sender	Відправник повідомлення
Receiver	Одержувач повідомлення
Reply-to	Адреси агентів, яким відправляється відповідь
Content	Вміст повідомлення
Language	Мова кодування повідомлення
Encoding	Кодування вмісту повідомлення
Ontology	Онтологія, використовувана для інтерпретації повідомлення
Protocol	Протокол взаємодії агентів
Conversation-id	Ідентифікатор повідомлень
Reply-with	Рядок, що ідентифікує повідомлення
In-reply-to	Рядок, що ідентифікує повідомлення, яке відповідає параметру reply-with при відповіді на повідомлення
Reply-by	Час, до якого необхідно отримати відповідь

Технічно комунікація між агентами відбувається за рахунок передачі повідомлень використовуючи будь-який транспортний протокол нижнього рівня (SMTP, TCP/IP, HTTP, IIOP).

Альтернативами для використання ACL є ряд інших мов, таких як мови БД (SQL), Distributed object systems (CORBA) та Web languages.

Розробка ACL використовує кілька підходів, один з них називається декларативним підхідом, який включає в себе мову KQML. KQML (Knowledge Query and Manipulation Lanquage) – це мова і набір протоколів, що забезпечують ідентифікацію, зв'язок і обмін інформацією з іншими агентами. Основні риси:

- повідомлення непрозорі щодо вмісту;
- примітиви мови – визначають припустимі операції, які може здійсніти агент для комунікацій з іншими агентами;
- середовище агентів, що використовує KQML, може бути поповнене спеціальними агентами для виконання сервісних функцій.

Синтаксис ACL дуже близький до широко використовуваної мови комунікації KQML. Однак, незважаючи на синтаксичну подібність, існують принципові відмінності між KQML і ACL, найбільш очевидною з яких є існування формальної семантики для ACL, яка має усунути будь-яку двозначність і плутанину під час використання мови [9].

Наразі мови комунікації агентів продовжують еволюціонувати. Оскільки сумісність – визначальна характеристика агентів, під час розробки МАС – дуже важливою є саме стандартизована комунікативність. Основними об'єктами стандартизації є: архітектура агента, мови взаємодії агентів, протоколи взаємодії агентів, знання агентів, мови програмування агентів. В галузі розробки агентів, для подальшої еволюції технологій створення агентів необхідні такі дії:

- розвиток семантики мов комунікації агентів (ACL) (загальних мов контенту та онтології;
- мов для опису дій агентів, намірів та прагнень);

- розвиток онтології агентів (розділені онтології для властивостей агентів та їх поведінки);
- поліпшення використання метаданих (абстрактне та суміжне з багатьма мовами контенту);
- декларативні та ясні протоколи (мови для визначення протоколів високого рівня, що базуються на більш примітивних);
- практичний обмін знаннями між агентами (соціальні механізми для обміну інформацією та знаннями, розгляд обміну знаннями як мобільний код);
- розвиток схем та методів для контролю за системами агентів (штучні ринки, природний відбір і т. п.).

6.3 Платформи для розробки МАС

Найбільш популярні засоби розробки МАС:

− JADE (Java Agent Development Framework) – широко використовуване програмне середовище для створення мультиагентних систем і додатків, що підтримує FIPA – стандарти для інтелектуальних агентів. Включає в себе середовище виконання агентів (агенти реєструються і працюють під управлінням середовища), бібліотеку класів, які використовуються для розробки агентних систем, набір графічних утиліт для адміністрування і спостереження за життєдіяльністю активних агентів. Програмне середовище JADE підключається до будь-якого проекту на мові Java. Агенти JADE можуть бути абсолютно різними – від простих, що тільки реагують, до складних– ментальних;

− JACK Intelligent Agents – Java платформа для створення мультиагентних систем. Так само як і JADE, вона розширює Java своїми класами. JACK одна з небагатьох платформ, де використовується модель логіки агентів, заснована на переконаннях – бажаннях – намірах (Belief – desire – intention software model – BDI), і має вбудовані формально – логічні засоби планування роботи агентів;

− MadKIT – модульна і масштабована мультиагентна платформа, написана на Java. Підтримує агентів на різних мовах: Java, Python, Jess, Scheme, BeanSchell. Красиво візуалізує і дозволяє управляти цими агентами;

− AgentBuilder – великий комерційний продукт, що випускається також і в Academic Edition. Агенти достатньо інтелектуальні, спілкуються на мові KQML (Knowledge Query and Manipulation Language) і мають ментальну модель. Платформа є Java-орієнтованою;

− Cougaar (Cognitive Agent Architecture) – також Java-орієнтована платформа для побудови розподілених мультиагентних систем. Включає не тільки виконуючу систему (run-time engine), але і деякі засоби для візуалізації, управління даними та ін.;

− NetLogo – кроссплатформенне програмоване середовище для програмування мультиагентних систем;

− VisualBots – безкоштовний мультагентний симулятор в Microsoft Excel з Visual Basic синтаксисом;

− MASON – Java бібліотека для моделювання мультиагентних систем;

− REPAST – набір інструментів для створення систем, заснованих на агентах;

− CogniTAO – C++ платформа розробки автономних мультиагентних систем, орієнтована на реальних роботів і віртуальних істот (CGF).

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Назвіть відомі стандарти, які застосовуються для створення МАС.
2. Коротко охарактеризуйте модель керування агентами, що базується на специфікаціях FIPA,
3. Наведіть схему еталонної моделі агентської платформи FIPA.
4. Що таке мова комунікації агентів?
5. Вкажіть найбільш популярні засоби розробки МАС

ЛІТЕРАТУРА [8,9]

ТЕМА 7. РЕАЛІЗАЦІЯ АГЕНТІВ В ПРОГРАМНОМУ СЕРЕДОВИЩІ JADE

7.1 Теоретичні відомості по платформі

JADE (Java Agent Development Framework) – це програмна основа для спрощення розробки агентських програм відповідно до специфікацій FIPA для сумісних інтелектуальних багатоагентних систем. Мета JADE – спростити розробку, одночасно забезпечуючи відповідність стандартам за допомогою повного набору системних служб і агентів.

JADE поставляється в комплекті з деякими інструментами, які спрощують адміністрування платформи та розробку програм. Кожен інструмент міститься в окремому підпакеті jade.tools. На даний момент доступні такі інструменти:

1) Remote Management Agent, скорочено RMA, діє як графічна консоль для управління та контролю платформи. Перший екземпляр RMA можна запустити за допомогою параметра командного рядка ("-gui"), але потім можна активувати більше ніж один GUI. JADE підтримує узгодженість між декількома RMA, просто пересилаючи події всім їм. Крім того, консоль RMA може запускати інші інструменти JADE;

2) Dummy Agent – це інструмент моніторингу та налагодження, що складається з графічного інтерфейсу користувача та основного агента JADE. За допомогою графічного інтерфейсу користувача можна складати повідомлення ACL і надсилати їх іншим агентам; також можна відобразити список усіх надісланих або отриманих повідомлень ACL, доповнений інформацією про позначку часу, щоб дозволити запис розмови агента та репетицію;

3) Sniffer – це агент, який може перехоплювати повідомлення ACL, поки вони знаходяться в польоті, і відображає їх у графічному вигляді, використовуючи нотацію, подібну до діаграм послідовності UML. Це корисно для налагодження ваших агентських товариств, спостерігаючи за тим, як вони обмінюються повідомленнями;

4) Introspector – це агент, який дозволяє відстежувати життєвий цикл агента, повідомлення ACL, якими він обмінюється, і поведінку під час виконання;

5) Графічний інтерфейс користувача DF (менеджер директорій) – це повний графічний інтерфейс користувача, в якому користувач може створити складну мережу доменів і субдоменів жовтих сторінок. Цей графічний інтерфейс дозволяє простим та інтуїтивно зрозумілим способом керувати базою знань DF, об'єднувати DF з іншими DF та віддалено керувати (реєструвати/скасувати реєстрацію/змінювати/шукати) базою знань батьківського DF, а також дочірніх. DF (впровадження мережі доменів і субдоменів);

6) LogManagerAgent – це агент, який дозволяє встановлювати інформацію журналювання під час виконання, наприклад рівень журналу, як для JADE, так і для конкретних класів програми, які використовують журналювання Java;

7) SocketProxyAgent – це простий агент, який діє як двонаправлений шлюз між платформою JADE і звичайним з'єднанням TCP/IP. Повідомлення ACL, що передаються через власну транспортну службу JADE, перетворюються на прості рядки ASCII і надсилаються через сокет-з'єднання. Та навпаки, повідомлення ACL можна тунелювати через це з'єднання TCP/IP до платформи JADE.

Платформа JADE має багато особливостей, нижче наведені лише основні з них:

1) Платформа розподіленого агента. Агентську платформу можна розділити між кількома хостами, на кожному хості виконується лише одна програма Java, а отже, лише одна віртуальна машина Java. Агенти реалізовані як потоки Java і живуть у контейнерах агентів, які забезпечують підтримку виконання агента;

2) Графічний інтерфейс користувача для керування кількома агентами та контейнерами агентів із віддаленого хосту;

3) Інструменти налагодження, які допомагають у розробці багатоагентних програм на основі JADE;

4) Внутрішньоплатформна мобільність агента, включаючи передачу як стану, так і коду (за необхідності) агента;

5) Підтримка виконання декількох, паралельних і одночасних дій агента через модель поведінки. JADE планує поведінку агента без випередження;

6) Сумісна з FIPA Agent Platform, яка включає AMS (Agent Management System) і DF (Directory Facilitator). Ці компоненти автоматично активуються під час запуску агентської платформи;

7) Ефективне транспортування повідомлень ACL всередині однієї платформи агента. Насправді повідомлення передаються закодованими як об'єкти Java, а не рядки, щоб уникнути процедур маршалінгу та демаршалінгу;

8) Сервіс імен, сумісний із FIPA: під час запуску агенти отримують свій GUID (глобальний унікальний ідентифікатор) із платформи.

7.2 Основи роботи з платформою JADE

Розглянемо основні моменти роботи з платформою, опираючись на приклад «Торгівля книгами».

Створення агентів

Агент в платформі JADE створюється за допомогою успадкування від класу *jade.core.Agent* та реалізації методу *setup*():

```
import jade.core.Agent;
public class BookBuyerAgent extends Agent {
      protected void setup() {
      // Виведення в консоль вітання від агенту
      System.out.println("Hello! Buyer-agent "+getAID().getName()+" is ready.");
      }
}
```

Метод *setup*() виконує ініціалізацію агента. Далі агент функціонує в рамках своєї поведінки.

Кожен агент ідентифікується «ідентифікатором агента», представленим у вигляді екземпляра класу *jade.core.AID*. Метод *getAID*() класу Agent дозволяє отримати ідентифікатор агента. Об'єкт AID містить глобально унікальне ім'я та адресу. Ім'я в JADE має форму <нікнейм>@<назва-платформи>, тому агент на ім'я

Peter, який живе на платформі під назвою P1, матиме Peter@P1 як глобально унікальне ім'я. Адреси, включені в AID, є адресами платформи, на якій живе агент. Ці адреси використовуються лише тоді, коли агенту потрібно спілкуватися з іншим агентом, який живе на іншій платформі. Розробники повинні піклуватися про них лише в окремих випадках, які виходять за рамки цього посібника. Знаючи нікнейм агента, його AID можна отримати наступним чином:

```
String nickname = "Peter";
AID id = new AID(nickname, AID.ISLOCALNAME);
```

Передача аргументів агенту

Агенти можуть отримати параметри запуску, указані в командному рядку. Ці аргументи можна отримати як масив *Object* за допомогою методу *getArguments*() класу *Agent*. Ми хочемо, щоб наш *BookBuyerAgent* отримав назву книги, яку потрібно купити, як аргумент командного рядка. Щоб досягти цього, ми змінюємо його наступним чином:

```
import jade.core.Agent;
import jade.core.AID;

public class BookBuyerAgent extends Agent {
      // Назва книги для покупки
      private String targetBookTitle;
      // Список відомих продавців-агентів
      private AID[] sellerAgents = {new AID("seller1", AID.ISLOCALNAME), new
AID("seller2", AID.ISLOCALNAME)};
      // Ініціалізація агента
      protected void setup() {
      // Виведення в консоль вітання від агенту
      System.out.println("Hello! Buyer-agent "+getAID().getName()+" is ready.");
      // Отримання назви книги в якості стартового аргумента
      Object[] args = getArguments();
      if (args != null && args.length > 0) {
      targetBookTitle = (String) args[0];
      System.out.println("Trying to buy "+targetBookTitle);
      }
      else {
      // Змусити агента негайно припинити роботу
      System.out.println("No book title specified");
```

```
        doDelete();
        }
}
        // Розміщення операцій очищення агента
        protected void takeDown() {
        // Роздрукування повідомлення про звільнення
        System.out.println("Buyer-agent "+getAID().getName()+" terminating.");
        }
}
```

Завершення роботи агенту

Навіть якщо агент нічого не робить після виконання усіх дій, він продовжує працювати. Щоб завершити його роботу, необхідно викликати метод *doDelete*(). Безпосередньо перед завершенням роботи агента викликається метод *takeDown*(), який виконує операції з очищення агента.

Клас «поведінка агента»

Активність агента виконується в рамках «поведінки» (режиму роботи). Кожен агент виконується в окремому потоці Java. Логіка агента будується із комбінації режимів.

Поведінка є послідовністю дій, які повинен виконати агент (аналогічно методу Java класу), і реалізується як об'єкт класу, успадкованого від *jade.core.behaviours.Behaviour*. Поведінка агента визначається за допомогою методу *addBehaviour*() класу *Agent*. Поведінка може бути додана будь-якої миті: при запуску агента (у методі *setup*()) або всередині іншого режиму роботи.

Кожен клас, успадкований від класу *Behaviour*, повинен перевизначити метод *action*(), який фактично визначає операції, що задають поведінку агента, а також метод *done*(), який повертає значення бульова, що вказує на завершення тієї чи іншої поведінки, яка повинна бути видалена з пула режимів роботи агента.

Планування та виконання режимів роботи агента

Агент може виконувати кілька дій одночасно. Однак важливо зауважити, що планування поведінки в агенті не є випереджальним (як для потоків Java), а кооперативним. Це означає, що коли поведінку заплановано для виконання, її метод *action*() викликається та виконується до повернення. Таким чином, саме програміст

визначає, коли агент переходить від виконання певної поведінки до виконання наступної. Хоча цей підхід вимагає від програмістів невеликих додаткових зусиль, він має кілька переваг:

1. Дозволяє мати один потік Java для кожного агента (це дуже важливо, особливо в середовищах з обмеженими ресурсами, наприклад мобільних телефонах);

2. Забезпечує кращу продуктивність, оскільки перемикання поведінки надзвичайно швидке, ніж перемикання потоків Java;

3. Усуває всі проблеми синхронізації між одночасними діями, які отримують доступ до тих самих ресурсів (це також прискорює продуктивність), оскільки всі дії виконуються одним потоком Java.

Життєвий цикл агента представлений на рисунку 7.1.

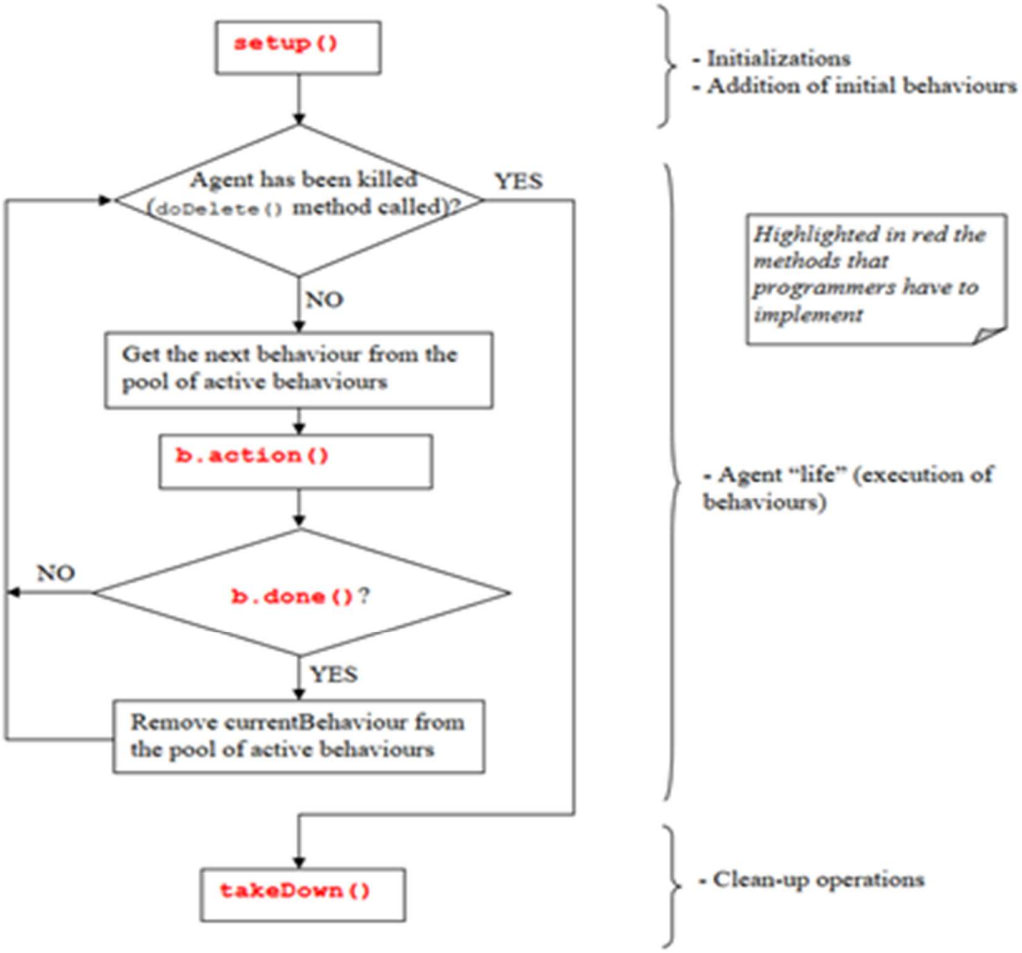

Рисунок 7.1 – Життєвий цикл агента [10]

Під час створення агента викликається метод *setup*(), у якому відбувається ініціалізація агента. Далі в нескінченному циклі вибирається і виконується наступний режим з активного пулу. Потім, залежно від типу режиму, він або видаляється з активного пулу, або залишається в ньому і виконується знову, коли до нього дійде черга. За відсутності активних режимів потік агента «засинає». При видаленні агента викликається метод *takeDown*().

Беручи до уваги описаний механізм планування, важливо підкреслити, що така поведінка, як описана нижче, запобігає виконанню будь-якої іншої поведінки, оскільки її метод *action*() ніколи не повертається.

```
public class OverbearingBehaviour extends Behaviour {
      public void action() {
      while (true) {
      // Виконання дій
      }
    }
    public boolean done() {
      return true;
    }
}
```

Якщо немає доступних для виконання дій, потік агента переходить у сплячий режим, щоб не споживати час ЦП. Він активується, як тільки є поведінка, доступна для виконання.

Типи режимів агента

Розрізняють три типи поведінки агентів:

1) «One-shot» поведінка, яка завершується негайно та чий метод *action*() виконується лише один раз. *jade.core.behaviours.OneShotBehaviour* уже реалізує метод *done*(), повертаючи *true*, і його можна зручно розширити, щоб реалізувати одноразову поведінку.

```
public class MyOneShotBehaviour extends OneShotBehaviour {
      public void action() {
      // Виконати операцію X
      }
}
```

Операція X буде виконана лише один раз.

2) «Циклічна» поведінка, яка ніколи не завершується, і чий метод *action*() виконує ті самі операції кожного разу, коли його викликають. *jade.core.behaviours.CyclicBehaviour* уже реалізує метод *done*(), повертаючи *false*, і може бути зручно розширений для реалізації циклічної поведінки.

```
public class MyCyclicBehaviour extends CyclicBehaviour {
      public void action() {
      // Виконати операцію Y
      }  }
```

Операція Y виконується багаторазово, до завершення роботи агента, що виконує циклічну поведінку.

3) Загальний (generic) режим, який виконує різні операції залежно від певної змінної. Режим залишається активним доти, доки виконується умова, задана у методі *done*().

```
public class MyThreeStepBehaviour extends Behaviour {
      private int step = 0;
      public void action() {
      switch (step) {
      case 0:
      // Виконати операцію X
      step++;
      break;
      case 1:
      // Виконати операцію Y
      step++;
      break;
      case 2:
      // Виконати операцію Z
      step++;
      break;
      }
}
      public boolean done() {
      return step == 3;
      }  }
```

Операції X, Y і Z виконуються одна за одною, та після цього поведінка завершується. JADE передбачає можливість комбінації простих режимів роботи агента для створення більш складних режимів.

Планування операцій у заданих часових точках

JADE надає два готових класи (в пакеті *jade.core.behaviours*), за допомогою яких можна легко реалізувати поведінку, яка виконує певні операції в задані моменти часу.

1) Режим *WakerBehaviour*, чиї методи *action*() і *done*() уже реалізовано таким чином, щоб виконувати абстрактний метод *handleElapsedTimeout*() після закінчення заданого часу очікування (вказаного в конструкторі). Після виконання методу *handleElapsedTimeout*() поведінка завершується.

```
public class MyAgent extends Agent {
      protected void setup() {
      System.out.println("Adding waker behaviour");
      addBehaviour(new WakerBehaviour(this, 10000) {
      protected void handleElapsedTimeout() {
      // Виконати операцію X
      }
      });
  }  }
```

Операція X виконується через 10 секунд після появи повідомлення «Додавання поведінки будильника».

2) TickerBehaviour, чиї методи *action*() і *done*() уже реалізовано таким чином, щоб виконувати абстрактний метод *onTick*() з повторюваним очікуванням заданого періоду (вказаного в конструкторі) після кожного виконання. *TickerBehaviour* ніколи не завершується.

```
public class MyAgent extends Agent {
      protected void setup() {
    addBehaviour(new TickerBehaviour(this, 10000) {
      protected void onTick() {
      // Виконати операцію Y
      }
      });
  }   }
```

Операція Y виконується періодично кожні 10 секунд.

Клас «взаємодія між агентами»

Однією з найважливіших функцій, яку надають агенти JADE, є здатність спілкування між агентами. Прийнятою комунікаційною парадигмою є асинхронна передача повідомлень (рис.6.2). Кожен агент має свого роду поштову скриньку (чергу повідомлень агента), куди середовище виконання JADE надсилає повідомлення, надіслані іншими агентами. Щоразу, коли повідомлення публікується в черзі повідомлень, приймаючий агент отримує сповіщення. Проте, якщо і коли агент дійсно забере повідомлення з черги повідомлень для обробки, це повністю залежить від програміста.

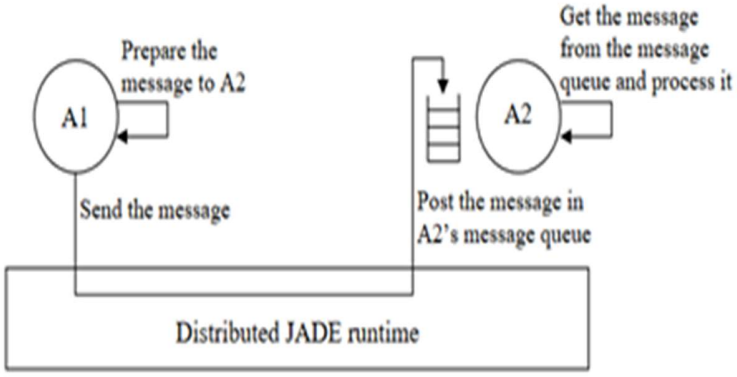

Рисунок 6.2 – Парадигма асинхронної передачі повідомлень JADE [10]

7.3 Мова ACL

Повідомлення, якими обмінюються агенти JADE, мають формат, визначений мовою ACL, визначеною міжнародним стандартом взаємодії агентів FIPA. Цей формат містить низку полів, зокрема:

- відправник повідомлення;
- список приймачів;
- комунікативний дії (також званий «побажання»), які вказують на те, чого відправник має намір досягти, надсилаючи повідомлення. Перформатив може бути REQUEST, якщо відправник хоче, щоб одержувач виконав дію,

INFORM, якщо відправник хоче, щоб одержувач знав про факт, QUERY_IF, якщо відправник хоче знати, чи виконується дана умова, CFP (виклик для пропозиції), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL, якщо відправник і одержувач ведуть переговори тощо;

- вміст, тобто фактична інформація, включена в повідомлення (тобто дія, яка має бути виконана в повідомленні REQUEST, факт, який відправник хоче розкрити в повідомленні INFORM ...);
- мова вмісту, тобто синтаксис, який використовується для вираження вмісту (і відправник, і одержувач повинні мати можливість кодувати розбирати вирази, сумісні з цим синтаксисом, щоб спілкування було ефективним);
- онтологія i.s. словниковий запас символів, які використовуються у вмісті, та їхнє значення (і відправник, і одержувач повинні описати те саме значення символів, щоб спілкування було ефективним);
- деякі поля, які використовуються для керування декількома одночасними бесідами та вказують час очікування для отримання відповіді, як-от ідентифікатор бесіди, відповідь із, відповідь на, відповідь.

Повідомлення в JADE реалізовано як об'єкт класу *jade.lang.acl.ACLMessage*, який надає методи *get* і *set* для обробки всіх полів повідомлення.

Надсилання повідомлень

Надіслати повідомлення іншому агенту так само просто, як заповнити поля об'єкта ACLMessage і потім викликати метод send() класу Agent. Наведений нижче код повідомляє агенту з псевдонімом Peter, що сьогодні йде дощ.

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("Peter", AID.ISLOCALNAME));
msg.setLanguage("English");
msg.setOntology("Weather-forecast-ontology");
msg.setContent("Today it's raining");
send(msg);
```

Надсилання повідомлень у програмі «Торгівля книгами»

У програмі "Торгівля книгами" використовується повідомлення CFP (call for proposal) для сповіщення про те, що агент продавця *Buyer-agent* надіслав агенту

покупця *Seller-agent* пропозицію на придбання книг. Повідомлення PROPOSE містить пропозицію продавця, а повідомлення ACCEPT_PROPOSAL містить замовлення покупця. Повідомлення REFUSE агента продавця повідомляє про те, що книги, яку запросив агент покупця, немає в каталозі. В обох типах повідомлень, що надсилаються агенту покупця, міститься назва книги. Змістом повідомлення PROPOSE має бути ціна на книгу. Нижче наведено приклад, що демонструє, яким чином створюється та оговтується повідомлення CFP.

```
// Повідомлення із запитом пропозиції
ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
for (int i = 0; i < sellerAgents.lenght; ++i) {
cfp.addReceiver(sellerAgents[i]);
}
cfp.setContent(targetBookTitle);
myAgent.send(cfp);
```

Отримання повідомлень

Середовище виконання JADE автоматично поміщає повідомлення, як тільки вони доставлені, у чергу повідомлень одержувача. Агент може отримувати повідомлення з черги повідомлень за допомогою методу *receive*(). Цей метод повертає перше повідомлення, що знаходиться у черзі (і видаляє його звідти), або нічого не повертає, якщо черга повідомлень порожня, після чого відразу ж завершується.

```
ACLMessage msg = receive();
if (msg != null) {
// Обробка повідомлення
}
```

Блокування режиму роботи агента для очікування повідомлення

Дуже часто в поведінці агенту потрібно реалізувати поведінку, яка обробляє повідомлення, отримані іншими агентами. Це стосується поведінки *OfferRequestsServer* і *PurchaseOrdersServer*, де нам потрібно обслуговувати повідомлення від агентів покупців із запитами на пропозиції та замовлення на купівлю. Така поведінка має працювати безперервно (циклічна поведінка) і при кожному виконанні свого методу *action*() повинна перевіряти, чи було отримано

повідомлення, і обробляти його. Дві моделі поведінки дуже схожі. Тут ми представляємо поведінку *OfferRequestsServer* (режим *PurhaseOrdersServer* реалізується аналогічно).

```
/**
Внутрішній клас OfferRequestsServer. Це поведінка, яку використовують агенти-продавці для обслуговування пропозицій від агентів-покупців. Якщо запитана книга є в місцевому каталозі, агент-продавець відповідає з повідомленням PROPOSE із зазначенням ціни. В іншому випадку повідомлення REFUSE відправляється назад.
*/
private class OfferRequestsServer extends CyclicBehaviour {
      public void action() {
      ACLMessage msg = myAgent.receive();
      if (msg != null) {
      // Повідомлення отримано. Обробка
      String title = msg.getContent();
      ACLMessage reply = msg.createReply();
      Integer price = (Integer) catalogue.get(title);
      if (price != null) {
      // Запитана книга доступна для продажу. Відповісти з ціною
      reply.setPerformative(ACLMessage.PROPOSE);
      reply.setContent(String.valueOf(price.intValue()));
      }
      else {
      // Запитана книга НЕ доступна для продажу.
      reply.setPerformative(ACLMessage.REFUSE);
      reply.setContent("not-available");
      }
      myAgent.send(reply);
      }
  }
} // Кінець внутрішнього класу OfferRequestsServer
```

У цьому коді поведінка *OfferRequestsServer* реалізовано як внутрішній клас класу *BookSellerAgent*. Це полегшує реалізацію, т.к. забезпечує прямий доступ до каталогу книг для продажу, але не обов'язковий.

Метод *createReply*() класу *ACLMessage* автоматично створює нове повідомлення *ACLMessage*, правильно встановлюючи отримувачів і всі поля, які

використовуються для керування бесідою (ідентифікатор розмови, відповідь із, відповідь-до), якщо такі є.

Відповідно до рисунка 15.1, при додаванні вищенаведеної поведінки, потік агента починає безперервний цикл, який надзвичайно споживає ЦП. Щоб уникнути цього, слід виконувати метод *action*() поведінки *OfferRequestsServer* лише тоді, коли надходить нове повідомлення. Для цього можна використовувати метод *block*() класу *Behavior*. Цей метод позначає поведінку як «заблоковану», щоб агент більше не планував її виконання. Коли нове повідомлення вставляється в чергу повідомлень агента, усі заблоковані дії знову стають доступними для виконання, щоб вони мали можливість обробити отримане повідомлення. Тому метод *action*() має бути змінено наступним чином.

```
public void action() {
ACLMessage msg = myAgent.receive();
   if (msg != null) { // Повідомлення отримано. Обробка
   ...
   }
   else {
   block();
  }
}
```

Вибір повідомлень із заданими характеристиками з черги повідомлень

Враховуючи, що поведінка *OfferRequestsServer* і *PurchaseOrdersServer* є циклічною, метод *action*() починається з виклику *myAgent.receive*(), що породжує наступну проблему: як ми можемо бути впевнені, що поведінка *OfferRequestsServer* підбирається з черги повідомлень агента лише повідомлення, що містять запити на пропозиції та поведінку *PurchaseOrdersServer* лише повідомлення, що містять замовлення на купівлю? Щоб вирішити цю проблему, потрібно змінити код, вказавши відповідні «шаблони» під час виклику методу *receive*(). Коли вказано шаблон, метод *receive*() повертає перше повідомлення (якщо таке є), яке йому відповідає, а всі невідповідні повідомлення ігнорує. Такі шаблони реалізовані як екземпляри класу *jade.lang.acl.MessageTemplate*, який надає ряд фабричних методів для створення шаблонів дуже простим і гнучким способом.

Для запиту про наявність книги використовується повідомлення CFP, та повідомлення ACCEPT_PROPOSAL – для передачі пропозиції замовлення. Тому необхідно змінити метод *action*() класу *OfferRequestServer* так, щоб виклик *myAgent.receive*() ігнорував усі повідомлення, крім CFP.

```
public void action() {
    MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
    ACLMessage msg = myAgent.receive(mt);
      if (msg != null) {
      // CFP Повідомлення отримано. Обробка...
    }
      else {
    block();
    }
}
```

Складні розмови

Поведінка *RequestPerformer*, являє собою приклад поведінки, яка виконує «складну» розмову. Розмова — це послідовність повідомлень, якими обмінюються два або більше агентів із чітко визначеними причинно-наслідковими та часовими зв’язками. Поведінка *RequestPerformer* має надіслати повідомлення CFP кільком агентам (агентам-продавцям), отримати всі відповіді та, у разі отримання принаймні відповіді PROPOSE, надіслати наступне повідомлення ACCEPT_PROPOSAL (агенту продавця, який зробив пропозицію) і отримати відповідь. Щоразу, коли має бути здійснено розмову, добре вказувати поля керування розмовою в повідомленнях, якими обмінюються під час розмови. Це дозволяє легко та однозначно створювати шаблони, що відповідають можливим відповідям.

```
/**
Внутрішній клас RequestPerformer. Це поведінка, яку використовують агенти-покупці
для запиту цільової книги в агентів-продавців.
*/
private class RequestPerformer extends Behaviour {
      private AID bestSeller; // Агент, який надає найкращу пропозицію
      private int bestPrice; // Найкраща запропонована ціна
      private int repliesCnt = 0; // Лічильник відповідей від продавців-агентів
      private MessageTemplate mt; // Шаблон для отримання відповідей
```

```
    private int step = 0;
    public void action() {
    switch (step) {
    case 0:
    // Надсилання cfp усім продавцям
    ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
    for (int i = 0; i < sellerAgents.length; ++i) {
    cfp.addReceiver(sellerAgents[i]);
    }
    cfp.setContent(targetBookTitle);
    cfp.setConversationId("book-trade");
    cfp.setReplyWith("cfp"+System.currentTimeMillis()); // Унікальне значення
    myAgent.send(cfp);
    // Підготування шаблону для отримання пропозицій
    mt = MessageTemplate.and(MessageTemplate.MatchConversationId("book-trade"),
    MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
    step = 1;
    break;
    case 1:
    // Отримування всіх пропозиції/відмови від агентів продавців
    ACLMessage reply = myAgent.receive(mt);
    if (reply != null) {
    // Отримано відповідь
    if (reply.getPerformative() == ACLMessage.PROPOSE) {
    // Пропозиція
    int price = Integer.parseInt(reply.getContent());
    if (bestSeller == null || price < bestPrice) {
    // Найкраща пропозиція на даний момент
    bestPrice = price;
    bestSeller = reply.getSender();
    }
    }
    repliesCnt++;
    if (repliesCnt >= sellerAgents.length) {
    // Отримання всіх відповідей
    step = 2;
    }
}
else {
block();
}
```

```
        break;
        case 2:
        // Надсилання замовлення продавцю, який надав найкращу пропозицію
        ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
        order.addReceiver(bestSeller);
        order.setContent(targetBookTitle);
        order.setConversationId("book-trade");
        order.setReplyWith("order"+System.currentTimeMillis());
        myAgent.send(order);
        // Підготування шаблону для отримання відповіді на замовлення
        mt = MessageTemplate.and(MessageTemplate.MatchConversationId("book-trade"),
        MessageTemplate.MatchInReplyTo(order.getReplyWith()));
        step = 3;
        break;
        case 3:
        // Отримання відповіді на замовлення
        reply = myAgent.receive(mt);
        if (reply != null) {
        // Отримання відповіді на замовлення
        if (reply.getPerformative() == ACLMessage.INFORM) {
        // Покупка успішна. Припинення роботи
        System.out.println(targetBookTitle+" successfully purchased.");
        System.out.println("Price = "+bestPrice);
        myAgent.doDelete();
        }
        step = 4;
        }
        else {
        block();
        }
        break;
        }
    }
        public boolean done() {
        return ((step == 2 && bestSeller == null) || step == 4);
        }
} // Кінець внутрішнього класу RequestPerformer
```

JADE надає основу для реалізації обміну повідомленнями за певними протоколами в пакеті *jade.protopackage*. Зокрема, обмін повідомленнями,

реалізований у прикладі, підтримується протоколом Contract net і може бути реалізований за допомогою класу *jade.proto.ContractNetInitiator*.

Отримання повідомлень у режимі блокування

Крім методу *receive*(), клас Agent надає також метод *blockingReceive*(), який є блокуючим викликом, т.к. він не повертає керування доти, доки в черзі повідомлень агента не з'явиться повідомлення. Перевантажена версія цього методу отримує як аргумент *MessageTemplate* (метод не повертає управління до тих пір, поки в черзі повідомлень агента не з'явиться повідомлення, що задовольняє шаблон).

Підкреслимо, що метод *blockingReceive*() фактично блокує потік агента. Тому, якщо з деякого режиму викликається метод *blockingReceive*(), виконання інших режимів припиняється до тих пір, поки метод *blockingReceive*() не поверне керування. Хорошим стилем програмування переговорів між агентами є використання *blockingReceive*() у методах setup() та takeDown(); використання всередині поведінки методу *receive*() у поєднанні з *behaviour.block*().

Сервіс «жовтих сторінок»

У наведеному прикладі було зроблено припущення, що існує деяке фіксована множина агентів-продавців (*seller1* і *seller2*) і кожен покупець заздалегідь знає про них. У цьому розділі описано, як використовувати сервіс «жовтих сторінок», що надається платформою JADE, щоб дозволити агентам покупцям динамічно дізнаватися про агентів-продавців, доступних на даний момент часу.

Агент DF

Служба «жовтих сторінок» дозволяє агентам публікувати одну або кілька послуг, які вони надають, щоб інші агенти могли їх знаходити та послідовно використовувати, як описано на рисунку 6.3.

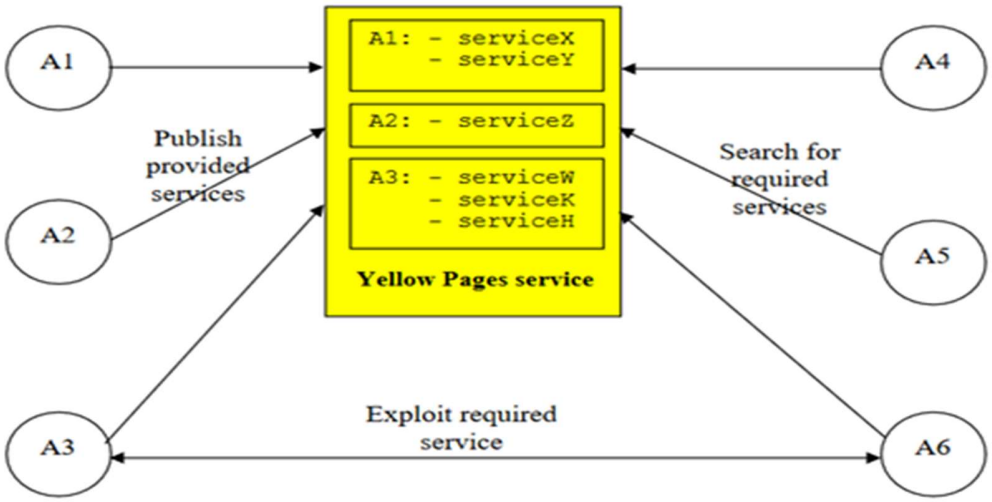

Рисунок 6.3 – Сервіс «жовтих сторінок» [10]

Кожна платформа JADE за замовчуванням підтримує агента DF (його локальне ім'я «*df*»). Можуть бути запущені та об'єднані кілька DF-агентів (включно з тими, що знаходяться в платформі за замовчуванням), для забезпечення єдиного розподіленого каталогу «жовтих сторінок».

Взаємодія з DF

Оскільки DF є агентом, можна взаємодіяти з ним, як зазвичай, обмінюючись повідомленнями ACL, використовуючи належну мову вмісту (мова SLO) і належну онтологію (онтологія керування агентом FIPA) відповідно до специфікації FIPA. Щоб спростити цю взаємодію, JADE надає клас *jade.domain.DFService*, за допомогою якого можна публікувати та шукати служби через виклики методів.

Публікація сервісів

Агент, який бажає опублікувати (зробити доступними) один або більше сервісів, повинен надати DF, що включає AID цього агента, список можливих мов та онтологій, які мають бути відомі іншим агентам для взаємодії з ним, та список сервісів для публікації. Для кожного публікованого сервісу надається опис, що включає тип сервісу, його ім'я, мови та онтології, необхідні для його використання, та ще деякі властивості, специфічні для даного сервісу. Класи *DFAgentDescription*, *ServiceDescription* і *Property*, що входять до складу пакету *jade.domain.FIPAAgentManagement*, відповідно є три згадані абстракції.

Щоб опублікувати сервіс, агент повинен створити опис, представлений екземпляром класу *DFAgentDescription*, та викликати статичний метод *register*() класу *DFService*. Зазвичай реєстрація сервісу виконується методом *setup*(), як показано нижче у прикладі з агентом-продавцем книг.

```
protected void setup() {
      ...
      // Реєстрування послуги книготоргівлі на жовтих сторінках
      DFAgentDescription dfd = new DFAgentDescription();
      dfd.setName(getAID());
      ServiceDescription sd = new ServiceDescription();
      sd.setType("book-selling");
      sd.setName("JADE-book-trading");
      dfd.addServices(sd);
      try {
      DFService.register(this, dfd);
      }
      catch (FIPAException fe) {
      fe.printStackTrace();
      }
      .

}
```

Зауважте, що в цьому простому прикладі ми не вказуємо жодної мови, онтології чи специфічної властивості служби. Коли агент припиняє роботу, рекомендовано скасувати реєстрацію опублікованих служб.

```
protected void takeDown() {
      // Скасувати реєстрацію на жовтих сторінках
      try {
      DFService.deregister(this);
      }
      catch (FIPAException fe) {
      fe.printStackTrace();
      }
      // Закривання GUI
      myGui.dispose();
      // Виведення повідомлення про звільнення
      System.out.println("Seller-agent "+getAID().getName()+" terminating.");
}
```

Пошук сервісів

Агент, якому потрібно знайти деякі послуги, повинен надати агенту DF опис необхідного шаблону. Результатом пошуку є список усіх описів, які відповідають вказаному шаблону. Опис відповідає шаблону, якщо всі поля, вказані в шаблоні, присутні з тими ж значеннями та описом.

Використання статичного методу *search*() класу *DFService* можна проілюструвати прикладом, у якому агент покупець книжок динамічно знаходить всіх агентів, наданих сервісом «book selling» (продаж книжок).

```
public class BookBuyerAgent extends Agent {
      // Назва книги для покупки
      private String targetBookTitle;
      // Список відомих продавців-агентів
      private AID[] sellerAgents;
      // Ініціалізація агента
      protected void setup() {
      // Роздрукування вітального повідомлення в консоль
      System.out.println("Hello! Buyer-agent "+getAID().getName()+" is ready.");
      // Отримання назви книги, як стартовий аргумент
      Object[] args = getArguments();
      if (args != null && args.length > 0) {
      targetBookTitle = (String) args[0];
      System.out.println("Trying to buy "+targetBookTitle);
      // Додавання TickerBehaviour, який щохвилини планує запит до агентів продавців
      addBehaviour(new TickerBehaviour(this, 60000) {
      protected void onTick() {
      // Update the list of seller agents
      DFAgentDescription template = new DFAgentDescription();
      ServiceDescription sd = new ServiceDescription();
      sd.setType("book-selling");
      template.addServices(sd);
      try {
      DFAgentDescription[] result = DFService.search(myAgent, template);
    sellerAgents = new AID[result.length];
      for (int i = 0; i < result.length; ++i) {
      sellerAgents[i] = result[i].getName();
      }
      }
      catch (FIPAException fe) {
```

```
        fe.printStackTrace();
        }
        // Виконання запиту
        myAgent.addBehaviour(new RequestPerformer());
        }
});
```

Зверніть увагу, що оновлення списку відомих агентів-продавців виконується перед кожною спробою купити цільову книгу, оскільки агенти-продавці можуть динамічно з'являтися та зникати в системі. Клас *DFService* також надає підтримку для підписки на DF для отримання сповіщень, щойно агент опублікує певну послугу.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте платформу JADE, вкажіть її основні особливості.
2. Що таке RMA? Назвіть можливості та особливості засобу.
3. Охарактеризуйте інструмент Dummy Agent
4. Назвіть основні етапи життєвого циклу агента, реалізація яких можлива з застосуванням платформи JADE.
5. Вкажіть типи поведінки агентів та які методи платформи JADE застосовуються для їх реалізації?
6. Що таке «Планування операцій у заданих часових точках» ? Які методи платформи JADE при цьому застосовуються?
7. Що ма'ться на увазі під терміном «складна розмова» агентів та що потрібно для її організації?

ЛІТЕРАТУРА [10-12]

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Dorri, A. Multi-agent systems: A survey = Мультиагентні системи: Опитування [Електронний ресурс] / A. Dorri, S.S. Kanhere, R, Jurdak // IEEE Xplore, 2018.– V.6. – P. 28573–28593. – Режим доступу: https://doi.org/10.1109/access.2018.2831228 .
2. Мультиагентні інтелектуальні інформаційні системи : навч. посіб. / Ю.Я. Бобало, І.В. Горбатий, М.Д. Кіселичник та ін. ; ред. Ю.Я. Бобало, І.В. Горбатий. – Львів : вид-во Львівської політех., 2019. – 580 с.
3. Balaji, P.G. An introduction to multi-agent systems = Вступ до багатоагентних систем [Електронний ресурс] / Balaji, P.G. and Srinivasan, D. // Innovations in Multi-Agent Systems and Applications-1. – Springer, 2010. – P. 1–27. – Режим доступу: https://doi.org/10.1007/978-3-642-14435-6_1.
4. Бережний А.О. Методи рішення завдань планування поведінки агентів в інтелектуальних системах підтримки прийняття рішень / А.О. Бережний, М.Ю. Сорока, Н.А. Сало // Збірник наукових праць Харківського національного університету Повітряних Сил. – 2019. – № 4(62). – С. 18-24.
5. Тарасов, В. Б. Агенти, багатоагентні системи, віртуальні спільноти: стратегічний напрям в інформатиці та штучному інтелекті / В. Б. Тарасов // Новини штучного інтелекту. – 1998. – № 2 – С. 5-63.
6. Плескач, В.Л. Агентні технології : монографія / В.Л. Плескач, Ю.В.Рогушина. – К. : Київ. нац. торг.–екон. ун–т, 2005. – 344 с.
7. Satoh, I. Mobile Agent-based Context-aware Services [Електронний ресурс]. / I. Satoh // Journal of Universal Computer Science. – 2010. – V.16, №15. – Режим доступу: https://doi.org/10.1007/978-0-387-93808-0_29 .
8. The foundation for intelligent physical agents = Фонд інтелектуальних фізичних агентів (FIPA) [Електронний ресурс]. – Режим доступу: http://www.fipa.org/

9. Adaptive Computing on the Grid Using AppLeS / F. Berman, R. Wolski, H. Casanova and other // IEEE Transactions on Parallel and Distributed Systems. – 2003. – V 14, N 4. – P. 369-382.
10. Java Agent Development Framework (JADE). [Електронний ресурс]. – Режим доступу: https://jade-project.gitlab.io/
11. Городецкий, В.И. Многоагентные системы (обзор) [Електронний ресурс] / В.И.Городецкий, М.С.Грушинский, А.В.Хабалов. – Режим доступу: http://www.raai.org/library/ainews/1998/2/GGKHMAS.ZIP
12. Рассел, С. Штучний інтелект: сучасний підхід (AIMA-2) = Artificial Intelligence: A Modern Approach / Стюарт Рассел, Пітер Норвіг. – 2-е вид. – М. : Вільямс, 2020. – 1480 с.
13. Будур, І.М. Мультиагентна модель системи підтримки прийняття рішення по управлінню розподіленими об’єктами [Електронний ресурс] / І.М. Будур, С.А. Бойко // Наука і техніка Повітряних Сил Збройних Сил України. – 2020.– №3(63). – С. 54-60. – Режим доступу: https://journal-hnups.com.ua/index.php/soivt/article/view/391/325 .