МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М.Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Пояснювальна записка до дипломного проекту

магістра
(освітній ступінь)

на тему «Формалізація взаємодії інтелектуальних агентів в галузі Інтернету речей за допомогою онтологічного інжинірингу»

ХАІ.603.667П1.121.156340.20О

Виконав: студент 6 курсу групи № 667П1
Спеціальність 121 – Інженерія програмного забезпечення
(код та найменування)
Освітня програма Хмарні обчислення та Інтернет речей
(найменування)
Левашов Д.А.
(прізвище й ініціали студента)
Керівник: Волобуєва Л.О.
(прізвище й ініціали)
Рецензент: Ляшенко О.С.
(прізвище й ініціали)

Харків – 2020

Міністерство світи і науки України
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу
(повне найменування)
Кафедра інженерії програмного забезпечення
(повне найменування)
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – інженерія програмного забезпечення
(код та найменування)
Освітня програма хмарні обчислення та Інтернет речей
(найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (ініціали та прізвище)
“_____” _____ 2020 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЕКТ СТУДЕНТУ

Левашову Денису Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема дипломного проекту «Формалізація взаємодії інтелектуальних агентів в галузі Інтернету речей за допомогою онтологічного інжинірингу»
керівник дипломного проекту Волобуєва Ліна Олексіївна к.т.н, доцент каф. 603
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом Університету № _____ від “____” _____ 2020 року
2. Термін подання студентом проекту _____
3. Вихідні дані до проекту формалізувати взаємодію інтелектуальних агентів в галузі Інтернету речей за допомогою онтологічного інжинірингу.
_____
_____
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) вивчити способи застосування інтелектуальних агентів в галузі Інтернету речей, формалізувати взаємодію таких інтелектуальних агентів за допомогою онтологічного інжинірингу, оцінити адекватність побудованої онтології і розробити практичні рекомендації щодо її застосування.
_____
_____
_____
5. Перелік графічного матеріалу (усього 108 сторінок), 24 рисунків, 6 таблиць, 30 джерел, додаток А, додаток Б, додаток В

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Волобуєва Л.О., доцент каф. 603		
2	Волобуєва Л.О., доцент каф. 603		
3	Волобуєва Л.О., доцент каф. 603		

Нормоконтроль _____ _____Постернакова В.А._ «___»_____20__р.
(підпис) (ініціали та прізвище)

7. Дата видачі завдання «___»_____20_____ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	Аналіз концепції Інтернету речей	01.10.19 - 01.12.19	
2	Вивчення теорії інтелектуальних агентів	01.01.20 - 01.03.20	
3	Вивчення теорії формалізації знань та поняття онтології	01.04.20 - 01.06.20	
4	Аналіз використання інтелектуальних агентів у галузі Інтернету речей	01.06.20 - 01.07.20	
5	Вивчення еталонної моделі архітектури речі в концепції Інтернету речей	01.09.20 - 01.10.20	
6	Розробка онтології для формалізації взаємодії інтелектуальних агентів у галузі Інтернету речей	01.10.20 - 21.10.20	
7	Вивчення архітектури та принципів роботи програмної платформи «JADE»	01.11.20 - 15.11.20	
8	Розробка прототипів агентів для тестування розробленої онтології	21.11.20 - 24.11.20	
9	Написання програм агентів у програмній платформі «JADE» та тестування онтології	25.11.20 - 27.11.20	
10	Оформлювання пояснювальної записки до дипломного проекту	01.12.20 - 15.12.20	
11	Оформлення презентації та доповіді до дипломного проекту	15.12.20 - 15.12.20	

Студент _____ Левашов Д.А.
(підпис) (прізвище та ініціали)

Керівник проекту ____ Волобуєва Л.О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломний проект на тему «Формалізація взаємодії інтелектуальних агентів в галузі Інтернету речей за допомогою онтологічного інжинірингу»: 108 сторінок, 24 рисунків, 30 джерел.

Об'єкт дослідження – процес взаємодії інтелектуальних агентів в системах Інтернету речей .

Предмет дослідження – формалізація взаємодії інтелектуальних агентів в системах Інтернету речей засобами онтологічного інжинірингу.

Мета дослідження – формалізація взаємодії інтелектуальних агентів в галузі Інтернету речей, що спростить їх розробку і надасть нові можливості для використання.

Мета роботи полягає в використанні методів: онтологічного інжинірингу та теорії інтелектуальних агентів.

В першому розділі були розглянуті основні принципи і архітектура Інтернету речей, розвиток і перспективи, а також головні проблеми; розглянуті поняття інтелектуальних агентів, класифікація, переваги та використання їх в області Інтернету речей; доведена актуальність формалізації взаємодії інтелектуальних агентів в області Інтернету речей.

У другому розділі була розглянута теорія роботи інтелектуальних агентів; детально розглянуте поняття онтології, і теорія побудови онтологій; був описаний процес розробки онтології для формалізації взаємодії інтелектуальних агентів в області Інтернету речей.

У третьому розділі були розроблені прототипи агентів, на прикладі взаємодії яких було доведено можливість застосування розробленої онтології для проектування реальних систем інтелектуальних агентів в галузі Інтернету речей.

Актуальність даної роботи полягає в тому, що на сьогоднішній день Інтернет речей – це галузь, що бурхливо розвивається, що має важливе технічне, соціальне і економічне значення. У ній повсюдно використовуються інтелектуальні агенти, породжуючи концепцію Інтернету агентів як нової версії технології многоагентних систем. У такій концепції у кожної «речі» повинні бути не тільки набір датчиків і пристроїв впливу на навколишній світ, але свій агент, який міг би представляти його інтереси в віртуальному середовищі інших речей, що дозволило виконувати більш складні завдання, роблячи річ по-справжньому «розумною», а не просто автоматизованої. Тому актуальність завдання формалізації взаємодії агентів в контексті їх використання в Інтернеті речей на сьогоднішній висока.

КОНЦЕПЦІЯ ІНТЕРНЕТУ РЕЧЕЙ, IOT, ІНТЕЛЕКТУЛЬНІ АГЕНТИ, ІНТЕРНЕТ АГЕНТІВ, ФОРМАЛІЗАЦІЯ ВЗАЄМОДІЇ ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ, ОНТОЛОГІЧНИЙ ІНЖИНІРІНГ

РЕФЕРАТ

Дипломный проект на тему «Формализация взаимодействия интеллектуальных агентов в области Интернета вещей с помощью онтологического инжиниринга»: 108 страниц, 24 рисунков, 30 источников.

Объект исследования – интеллектуальные агенты в области Интернета вещей и их взаимодействие.

Предмет исследования – формализация взаимодействия интеллектуальных агентов в области Интернета вещей.

Цель исследования – формализация взаимодействия интеллектуальных агентов в области Интернета вещей, что упростит их разработку и предоставит новые возможности для использования.

Цель работы заключается в использовании методов: онтологического инжиниринга и теории интеллектуальных агентов.

В первом разделе были рассмотрены основные принципы и архитектура Интернета вещей, развитие и перспективы, а также главные проблемы; рассмотрены понятие интеллектуальных агентов, классификация, преимущества и использование их в области Интернета вещей; доказана актуальность формализации взаимодействия интеллектуальных агентов в области Интернета вещей.

Во втором разделе было рассмотрена теория работы интеллектуальных агентов; подробно рассмотрено понятие онтологии, и теория построения онтологий; был описан процесс разработки онтологии для формализации взаимодействия интеллектуальных агентов в области Интернета вещей.

В третьем разделе были разработаны прототипы агентов, на примере взаимодействия которых было доказана возможность применения разработанной онтологии для проектирования реальных систем интеллектуальных агентов в области Интернета вещей.

Актуальность данной работы заключается в том, что на сегодняшний день Интернет вещей – это отрасль бурно развивается, что имеет важное техническое, социальное и экономическое значение. В ней повсеместно используются интеллектуальные агенты, порождая концепцию Интернета агентов как новой версии технологии многоагентных систем. В такой концепции в каждой «вещи» должны быть не только набор датчиков и устройств воздействия на окружающий мир, но свой агент, который мог бы представлять его интересы в виртуальной среде других вещей, что позволило выполнять более сложные задачи, делая вещь по-настоящему «умной», а не просто автоматизированной. Поэтому актуальность задачи формализации взаимодействия агентов в контексте их использования в Интернете вещей на сегодняшний высока.

КОНЦЕПЦИЯ ИНТЕРНЕТ ВЕЩЕЙ, IOT, ИНТЕЛЕКТУЛЬНОМУ АГЕНТЫ, ИНТЕРНЕТ АГЕНТОВ, ФОРМАЛИЗАЦИЯ ВЗАИМОДЕЙСТВИЯ ИНТЕЛЛЕКТУАЛЬНЫХ АГЕНТОВ, ОНТОЛОГИЧЕСКИЙ ИНЖИНИРИНГ

ABSTRACT

Diploma project on the topic "Formalizing the Interaction of Intelligent Agents in the Field of the Internet of Things via Ontological Engineering": 108 pages, 24 figures, 30 sources.

The object of study – intellectual agents in the field of Internet of things and their interaction.

The subject of the study is the formalization of the interaction of intellectual agents in the field of the Internet of Things.

The purpose of the study is to formalize the interaction of intelligent agents typical of the Internet, which will simplify their development and provide new opportunities for use.

The purpose of the work is to use methods: ontological engineering and theory of intellectual agents.

The first section examined the basic principles and architecture of the Internet of Things, development and prospects, as well as the main problems; considered the concept of intelligent agents, classification, advantages and their use in the field of the Internet of Things; the urgency of formalizing the interaction of intelligent agents in the field of the Internet of Things has been proved.

In the second section, the theory of the work of intelligent agents was considered; the concept of ontology and the theory of building ontologies are considered in detail; the process of developing an ontology for formalizing the interaction of intelligent agents in the field of the Internet of Things was described.

In the third section, prototypes of agents were developed, on the example of the interaction of which the possibility of using the developed ontology for the design of real systems of intelligent agents in the field of the Internet of Things was proved.

The relevance of this work lies in the fact that today the Internet of Things is an emerging industry that has important technical, social and economic importance. It is widely used by intellectual agents, giving rise to the concept of the Internet of Agents as a new version of multi-agent system technology. In such a concept, each "thing" should have not only a set of sensors and impact devices on the outside world, but an agent who could represent his interests in the virtual environment of other things, which made it possible to perform more complex tasks by doing the thing " smart, not just automated. Therefore, the relevance of the task of formalizing the interaction of agents in the context of their use on the Internet of Things is high today.

CONCEPT INTERNET OF THINGS, IOT, INTELLECTUAL AGENTS, AGENT INTERNET, FORMALIZATION OF INTERACTION OF INTELLECTUAL AGENTS, ONTOLOGY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ ВИМІРЮВАНЬ ФІЗИЧНИХ ВЕЛИЧИН, СКОРОЧЕНЬ І ТЕРМІНІВ

IoT – Інтернет речей [1].

MAC – багатоагентні системи, мультиагентні системи [2].

ІА – інтелектуальні агенти[3].

ПЗ – програмне забезпечення.

ОІ – онтологічний інжиніринг [4].

ШІ – штучний інтелект.

БЗ ─ база знань.

Агент ─ у ШІ це комп'ютерна система, яка знаходиться в певному середовищі і здатна до самостійних дій у цьому середовищі для досягнення своїх цілей проектування [17].

Онтологія ─ описання та представлення знань деякої предметної області, домену [23].

IoT Reference Model ─ еталонна модель Інтернету речей, розроблена у рамках проекту IoT-A [28].

IoT Domain Model ─ модель домену Інтернету речей, є частиною еталонної моделі Інтернету речей [28].

OWL ─ мова веб-онтології ─ це сімейство мов подання знань для створення онтологій [23].

ВСТУП

Тема дипломного проекту: формалізація взаємодії інтелектуальних агентів в галузі Інтернету речей за допомогою онтологічного інжинірингу.

Актуальність роботи. На сьогоднішній день Інтернет речей ─ це галузь, яка бурхливо розвивається, що має важливе технічне, соціальне і економічне значення. У ній повсюдно використовуються інтелектуальні агенти, породжуючи концепцію Інтернету агентів як нової версії технології многоагентних систем. У такій концепції у кожної «речі» повинні бути не тільки набір датчиків та пристроїв впливу на навколишній світ, але свій агент, який міг би представляти її інтереси в віртуальному середовищі інших речей, що дозволило би виконувати більш складні завдання, роблячи річ по-справжньому «розумною», а не просто автоматизованою.

Для того, щоб інтелектуальний агент міг демонструвати «розумну» поведінку, міг навчатись на своєму досвіді, міг спілкуватись з іншими агентами, кооперуватись із ними для виконання складних задач, міг вивчати навколишній світ, та в цілому «розумів», що він робить, а не просто діяв лінійно по заданим алгоритмам для його проектування використовують бази знань. В базах знань проектувальник описує факти про світ та логіку поведінки агента у виді предикатів, щоб потім на основі їх агент міг приймати рішення. Для створення баз знань треба спочатку установити єдиний набір понять, якими база знань буде оперувати, онтологію. Вона відповідає на питання з чого складається світ, з яких концептів, та як ці концепти відносяться один до одного. Онтології завжди відносяться до якогось домену, предметної області, поняття якої вони намагаються охопити. В умовах мультиагентних систем, інтелектуальним агентам необхідно спілкуватись і для того, щоб вони могли розуміти один одного вони повинні розмовляти на єдиній мові, тобто розділяти єдину онтологію.

Проблема галузі Інтернету речей наразі у тому, що вона є дуже фрагментарною, тобто в ній не існує загальних стандартів для побудови рішень. Ті рішення, які існують, вони поодинокі і зовсім несумісні один з одним, так як створені з урахування конкретних проблем. Таким чином, поняття Інтернету речей насправді означає купу окремих, ізольованих один від одного Інтернетів речей.

Таким чином, з урахуванням наведених проблем можна сказати, що питання побудови онтології для формалізації взаємодії інтелектуальних агентів у галузі Інтернету речей, яка буде охоплювати специфічні поняття домену Інтернету речей та буде передбачати поняття для полегшення взаємодії агентів досить важливе.

Мета дипломного проекту - формалізація взаємодії інтелектуальних агентів в галузі Інтернету речей, що спростить їх розробку та взаємодію.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1 Вивчити способи застосування інтелектуальних агентів в Інтернеті речей.

2 Виявити типові завдання інтелектуальних агентів, які вони вирішують в контексті їх використання в Інтернеті речей, і спосіб їх взаємодії один з одним.

3 Формалізувати взаємодію таких інтелектуальних агентів за допомогою онтологічного інжинірингу.

4 Оцінити адекватність побудованої онтології і розробити практичні рекомендації щодо їх застосування.

Об'єкт дослідження – процес взаємодії інтелектуальних агентів в системах Інтернету речей .

Предмет дослідження – формалізація взаємодії інтелектуальних агентів в системах Інтернету речей засобами онтологічного інжинірингу.

Методи дослідження: онтологічний інжиніринг, теорія інтелектуальних агентів.

Наукова новизна дослідження: отримала подальший розвиток методологія проектування систем Інтернету речей за рахунок формалізації взаємодії інтелектуальних агентів засобами онтологічного інжинірингу.

Практичне значення результатів складається в застосуванні у процесі розробці систем інтелектуальних агентів в контексті Інтернету речей онтології, отриманої в результаті формалізації.

1 ОГЛЯД СУЧАСНОГО СТАНУ ТЕХНОЛОГІЙ

1.1 Огляд концепції Інтернету речей

На даний момент не існує єдиного визначення поняття Інтернету речей, згідно з найбільш поширеним формулюванням, Інтернет речей (англ. Internet of Things, IoT) – це концепція обчислювальної мережі фізичних предметів («речей»)[1], оснащених вбудованими технологіями для взаємодії один з одним або з зовнішнім середовищем, яка розглядає організацію таких мереж як явище, здатне перебудувати економічні та суспільні процеси, що виключає необхідність дії і операції за участі людини.

Концепція Інтернету речей сформульована в 1999 році як осмислення перспектив широкого застосування засобів радіочастотної ідентифікації для взаємодії фізичних предметів між собою і з зовнішнім оточенням. Наповнення концепції «Інтернету речей» різноманітним технологічним змістом і впровадження практичних рішень для її реалізації починаючи з 2010-х років вважається стійкою тенденцією в інформаційних технологіях, перш за все, завдяки повсюдного поширення бездротових мереж, появи хмарних обчислень, розвитку технологій міжмашинної взаємодії (Machine to machine (M2M)), початку активного переходу на IPv6 і освоєння програмних мереж.

«Речами» в Інтернеті речей є, головним чином, вбудовані пристрої з такими відмітними особливостями, як вузька смуга пропускання, збір даних з низькою повторюваністю і малий обсяг використовуваних даних. Ці пристрої обмінюються даними один з одним і надають дані через інтерфейси. Деякі вбудовані пристрої IoT, такі як охоронні відеокамери високого дозволу, відеотелефони VoIP і деякі інші, вимагають для роботи широкосмугового стрімінга. Але незліченна кількість інших продуктів вимагає передачі пакетів даних всього лише час від часу.

На даний момент IoT відкриває безпрецедентні можливості користувачам, виробникам пристроїв і постачальникам послуг в самих різних секторах (рисунок 1.1). У числі напрямків, яким підуть на користь можливості збору даних, автоматизації та аналізу, що надаються IoT, – охорона здоров'я і фітнес індустрія, моніторинг та автоматизація житлових будинків, енергозбереження та «інтелектуальна електромережа», сільське господарство, транспорт, екологічний моніторинг, інвентаризація та управління продукцією, безпека, відеоспостереження, освіту і багато інших [6].

Для практичної реалізації IoT всі навколишні предмети і пристрої (домашні прилади і посуд, одяг, продукти, автомобілі, промислове обладнання та ін.) повинні бути забезпечені мініатюрними ідентифікаційними і сенсорними (чутливими) пристроями [1]. Тоді при наявності необхідних каналів зв'язку з ними можна не тільки відслідковувати ці об'єкти і їх параметри в просторі і в часі, але і керувати ними. У загальному вигляді з інформаційно-комунікаційної точки зору Інтернет речей можна поділити на такі складові:

−сенсори (датчики);

–дані;

–мережі;

–послуги;

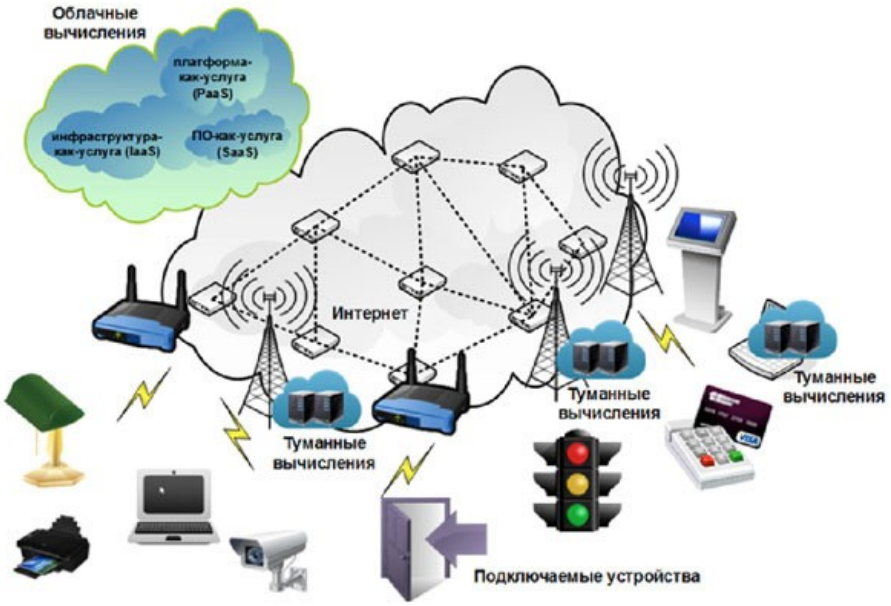

Рисунок 1.1 - Архітектура інтернету речей

Таким чином, Інтернет речей ─ це глобальна мережа комп'ютерів, датчиків (сенсорів) і виконавчих пристроїв (актуаторів), що зв'язуються між собою з використанням інтернет протоколу IP (Internet Protocol).

Нижче наведені фундаментальні характеристиками Інтернету речей.

Взаємопов’язаність. Всі пристрої взаємодіють через глобальну або локальну інфраструктуру інформаційного обміну.

Сервіси, орієнтовані на пристрої. Інтернет речей здатний забезпечити семантичну узгодженість між фізичними об’єктами реального світу і їх інформаційним поданням у віртуальному просторі і об’єднати фізичні пристрої з урахуванням правил і обмежень.

Гетерогенність. Пристрої в IoT неоднорідні за визначенням і можуть належати різним мережам і апаратних платформ, що не є перешкодою до взаємодії.

Динамічність. Стан пристроїв змінюється постійно: включення і виключення, контекстна і технологічна інформація, включаючи місце розташування і швидкість. Кількість підключених пристроїв також може динамічно змінюватися.

Масштабність. Кількість пристроїв, які будуть «спілкуватися» і отримувати керуючий вплив в десятки разів перевищить кількість вузлів в поточній мережі Інтернет. Очевидно, що кількість комунікацій, які можуть бути ініційовані пристроями, радикально перевищить можливе число з’єднань, ініціаторами яких виступають люди. Тому на перший план виходять питання інтерпретації даних, з метою їх подальшого застосування.

1.2 Огляд технології інтелектуальних агентів

1.2.1 Призначення і принцип роботи інтелектуальних агентів

Проблематика інтелектуальних агентів і мультиагентних систем (МАС) має майже 40-річну історію і сформувалася на основі результатів, отриманих в рамках робіт з роздільного ШІ (DAI), розподіленого вирішення завдань (DPS) і паралельного ШІ (PAI). Дана тематика інтегрує досягнення в області комп'ютерних мереж і відкритих систем, ШІ та інформаційних технологій [7].

Агент – це апаратна або програмна сутність, здатна діяти в інтересах досягнення цілей, поставлених перед ним власником або користувачем.

Таким чином, в рамках МАС-парадигми програмні агенти розглядаються як автономні компоненти, що діють від імені користувача.

Інтелектуальні агенти повинні мати наступні властивості[17]:

– автономність – здатність функціонувати без втручання з боку свого власника і здійснювати контроль внутрішнього стану і своїх дій;

– соціальну поведінку – можливість взаємодії і комунікації з іншими агентами;

– реактивність – адекватне сприйняття середовища і відповідні реакції на його зміни;

– активність – здатність генерувати цілі і діяти раціонально для їх досягнення;

- базові знання – знання агента про себе, навколишнє середовищу, включаючи інших агентів, які не змінюються в рамках ЖЦ агента;

– переконання – змінна частина базових знань, які можуть змінюватися в часі, хоча агент може про це не знати і продовжувати їх використовувати для своїх цілей;

- цілі – сукупність станів, на досягнення яких спрямована поточна поведінка агента;

– бажання – стану або ситуації, досягнення яких для агента важливо;

– зобов'язання – завдання, які бере на себе агент на прохання інших агентів;

– наміри – то, що агент повинен робити в силу своїх зобов'язань.

Залежно від концепцій, обраних для організації МАС, зазвичай виділяються три базових класу архітектур:

– архітектури, які базуються на принципах і методах роботи зі знаннями;

– архітектури, засновані на поведінкових моделях типу «стимул-реакція»;

– гібридні архітектури.

Агентом є все, що може сприймати своє середовище за допомогою датчиків і впливати на неї за допомогою виконавчих механізмів (людина, робот, програма) (рисунок 1.2). Кожен агент може сприймати власні дії[18].

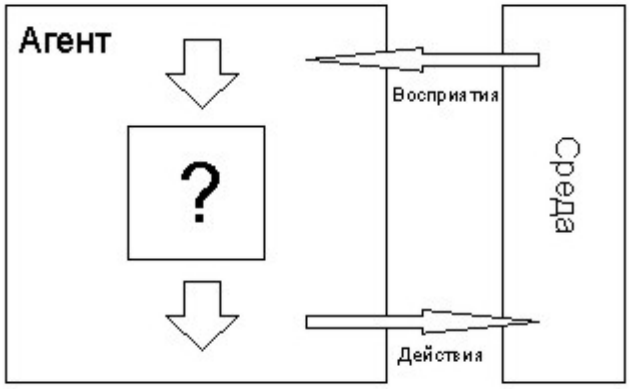

Рисунок 1.2 – Модель роботи агента

Вибір агентом дії в будь-який конкретний момент часу може залежати від всієї послідовності актів сприйняття, що спостерігалися до цього моменту часу. Поведінка агента може бути описано і за допомогою функції агента, яка відображає будь-яку конкретну послідовність актів сприйняття на деяку дію. Зовнішнім описом агента може служити таблиця. Внутрішній опис складається у визначенні того, яка функція агента реалізується за допомогою програми агента, тобто конкретна реалізація, діюча в рамках архітектури агента [7].

Всіх агентів можна розділити на п'ять груп по типу обробки сприймають інформації:

− агенти з простою поведінкою;

− агенти з поведінкою, заснованою на моделі;

− цілеспрямовані агенти;

− практичні агенти;

− агенти, що навчаються.

Агенти з простою поведінкою діють тільки на основі поточних знань. Їх агентська функція заснована на схемі умова-дія[17].

Така функція може бути успішною лише тоді, коли навколишнє середовище повністю піддається спостереженню. Деякі агенти також можуть мати інформацію про їх поточний стан, що дозволяє їм не звертати уваги на умови, передумови яких вже виконані.

Агенти з поведінкою, заснованим на моделі, можуть оперувати з середовищем, що лише частково піддається спостереженню. Усередині агента зберігається уявлення про ту частину, що знаходиться поза межами огляду. Щоб мати таке уявлення, агенту необхідно знати, як виглядає навколишній світ, як він улаштований. Ця додаткова інформація доповнює «картину світу».

Цілеспрямовані агенти схожі з попереднім типом, проте вони, крім іншого, зберігають інформацію про ті ситуації, які для них бажані. Це дає агенту спосіб вибрати серед багатьох шляхів той, що призведе до потрібної мети[18].

Цілеспрямовані агенти розрізняють тільки стан, коли мета досягнута, і коли не досягнута. Практичні агенти, крім цього, здатні розрізняти, наскільки бажано для них поточний стан. Така оцінка може бути отримана за допомогою «функції корисності», яка проектує безліч станів на безліч заходів корисності станів.

В деякій літературі агенти, що навчаються також називаються автономними інтелектуальними агентами (intelligent agents), що означає їх незалежність і здатність до навчання і пристосування до обставин, що змінюються. Система повинна проявляти такі здібності:

– навчатися і розвиватися в процесі взаємодії з навколишнім середовищем;
– пристосовуватися в режимі реального часу;
– швидко навчатися на основі великого обсягу даних;
– покроково пристосовувати нові способи вирішенням проблем;
– володіти базою прикладів з можливістю її поповнення;
– мати параметри для моделювання швидкої і довгої пам'яті, віку і т. д.
– аналізувати себе в термінах поведінки, помилки і успіху;

У будь-який конкретний момент часу оцінка раціональності дій агента залежить:

– від показників продуктивності;
– знань агента про середовище, отриманих раніше;
– дій, які можуть бути виконані агентом;
– послідовності минулих актів сприйняття агента[8].

1.2.2 Застосування технології інтелектуальних агентів в області Інтернету речей

Програмні агенти (програмні роботи, софтботи) існують в складних необмежених проблемних галузях. Наприклад, софтбот для управління тренажером, що імітує пасажирський літак, морське судно або софтбот, призначений для перегляду новин в Інтернеті та показу клієнтам повідомлень, які їх цікавлять. Інтернет є середовищем, яка за своєю складністю змагається з фізичним світом, а в число мешканців цієї мережі входить безліч штучних агентів [7].

Таким чином, очевидно чому технологія інтелектуальних агентів отримала широке застосування в області Інтернету речей. За допомогою агентів, які представляють інтереси кожної окремої речі її поведінка може стати набагато складнішою і послідовною, роблячи її по-справжньому розумною. На даний момент часто можна зустріти, що під Інтернетом речей розуміють просто віддалене управління, збір інформації та в кращому випадку автоматизацію. Ясно, що не варто очікувати від таких систем виконання складних, комплексних завдань, щоб участь людини обмежувалося тільки постановкою мети, не замислюючись над способом виконання (в таких системах людина занадто залучена в ланцюжок виконання завдання). Тому цілком очікувано що сьогодні йде повсюдне впровадження технології саме інтелектуальних агентів в Інтернет речей.

Нові покоління таких систем будуються на основі ідеї, що кожна «річ» рано чи пізно повинна стати «розумної», а користувач – отримати свого асистента, який допомагає вирішувати проблеми реального життя в реальному часі. Будь-яка річ у цьому підході рано чи пізно повинна буде володіти не тільки датчиками і пристроями впливу на об'єкти зовнішнього світу і комунікації, а й приймати рішення, причому по можливості узгоджено з іншими речами. Ці вимоги, в свою чергу, породжують концепцію Інтернету агентів як нової версії технології многоагентних систем, здатної реалізувати на практиці принципи самоорганізації і кооперації всіх компонент складної.

1.3 Онтологічний інжиніринг в задачах формалізації взаємодії агентів в межах концепції Інтернету речей

1.3.1 Призначення формального опису знань

Вперше, завдання побудови формального опису моделі виникла в програмних системах управління базами знань. Бази знань-це систематизоване зібрання фактів, яким керує розроблена спеціально для цих цілей програма. База реалізується як експертна система, тобто програма, яка виступає в якості спеціаліста-експерта в деякій тематиці. Наприклад, зараз дуже популярні експертні системи в медицині. Така база знань містить таку велику кількість медичних фактів, що одна людина просто не в змозі запам'ятати. Експертній системі можуть бути подані на вхід результати аналізів крові і програма, застосувавши факти своєї бази знань, може видати за результатами цих аналізів діагноз хворого. Експертну систему можна донавчати, додаючи в неї нові факти. Тепер, припустимо, що існують дві такі експертні системи (рисунок 1.3), зроблені різними виробниками і розташовані в різних медичних установах, і міністерство охорони здоров'я вирішило створити нову експертну систему на основі двох існуючих. Якби експерти були людьми, а не машинами, то все було б просто: вони б зустрілися і розповіли один одному все, що знають. З машинами ситуація інша ─ вони один одному просто так розповісти нічого не можуть. Тому, необхідно формально описати вміст бази знань кожної експертної системи з тим, щоб цей опис було зрозуміло програмі, яка реалізує їх об'єднання. Тут виникають такі труднощі:

1 Треба розробити мову, яка була б зрозуміла кожній з трьох програм, що беруть участь. Ця мова має бути досить формальною для того, щоб опис баз знань (онтології) однозначно інтерпретувалися машинами.

2 Побудувати опис вмісту бази знань на цій мові. Зрозуміло, що людина таке опис зробити не в змозі, тому що вона не в змозі тримати в умі вміст всієї бази знань експертної системи. Тому опис повинна робити програма.

3 За описом бази знань, яке зроблене розробленою мовою, додати нові факти в базу знань третьої експертної системи. Це також має зробити програма.

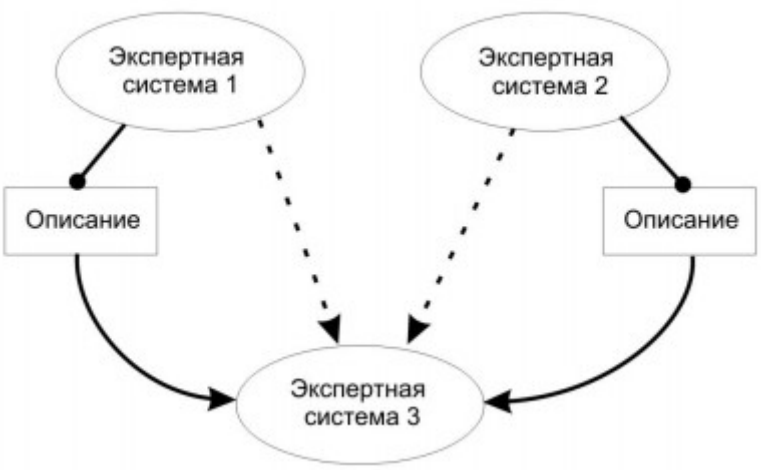

Рисунок 1.3 - Обмін знаннями між експертними системами

Тоді виникла думка стандартизувати мову опису знань, щоб більше не витрачати зусиль на її розробку. Тоді, другий і третій пункти зі списку вище звелися б просто до написання перекладачів з внутрішнього мови бази знань на цю мову, і з мови опису на мову подання знань в експертній системі[11].

Зараз відбувається процес впровадження формального опису знань в різні галузі інженерії подібний до того, який відбувався в 70-80 роках минулого століття з базами даних. У 60-х роках минулого століття була усвідомлена необхідність відділення опису даних від самих даних. Опис даних-це так звана схема бази даних (концептуальна схема). Була створена формальна теорія таких описів і, потім, були створені додатки, які могли працювати з такими описаннями – так звані сервери баз даних. Подібні процеси, тільки щодо опису знань, спостерігаються і в наш час. Подібно до того, як введення баз даних дозволило замінити машинами людини (і, тим самим, істотно збільшити обсяги оброблюваної інформації) в різних областях, пов'язаних з обробкою даних[11].

1.3.2 Застосування онтологій для формального опису знань

Онтологія ─ це артефакт, структура, що описує значення елементів деякої системи.

В останні роки розробка онтологій – явний формальний опис термінів предметної області та відносин між ними – переходить зі світу лабораторій зі штучного інтелекту на робочі столи експертів по предметним областям. У всесвітній павутині WWW онтології стали звичайним явищем. Онтології в мережі варіюються від великих таксономії, що категоризують веб-сайти, до категоризації товарів, що продаються і їх характеристик (як на сайті Amazon.com). У багатьох дисциплінах зараз розробляються стандартні онтології, які можуть використовуватися експертами по предметним областям для спільного використання і анотування інформації в своїй області.

Наприклад, в області медицини створені великі стандартні, структуровані словники, такі як SNOMED і семантична мережа Системи Уніфікованого Медичного Язика (Unified Medical Language System, UMLS). Також з'являються великі загальні онтології. Наприклад, Програма ООН з Розвитку (the United Nations Development Program) і компанія Dun & Bradstreet об'єднали зусилля для розробки онтології UNSPSC, яка надає термінологію товарів і послуг.

Онтологія визначає загальний словник для вчених, яким потрібно спільно використовувати інформацію предметної області. Вона включає машинно-інтерпретовані формулювання основних понять предметної області і відносини між ними.

Чому виникає потреба в розробці онтології? Ось деякі причини, які нижче будуть розглянуті докладніше:

− спільне розуміння знань домену (в межах спільноти людей, маючих єдину мету);

− обмін знаннями про домен;

− повторне використання знань домену;

− аналіз знань домену;

− відокремлення доменних знань від оперативних;

− виразні припущення щодо домену;

Забезпечення можливості використання знань предметної області стало однією з рушійних сил недавнього сплеску в вивченні онтологій. Наприклад, для моделей багатьох різних предметних областей необхідно сформулювати поняття часу. Це уявлення включає поняття тимчасових інтервалів, моментів часу, відносних заходів часу і т.д. Якщо одна група вчених детально розробить таку онтологію, то інші можуть просто повторно використовувати її в своїх предметних областях. Крім того, якщо потрібно створити велику онтологію, ми можемо інтегрувати кілька існуючих онтологій, що описують частини базової предметної області. Ми також можемо повторно використовувати основну

онтологію, таку як UNSPSC, і розширити її для опису цікавить нас предметної області.

Створення явних припущень в предметної області, що лежать в основі реалізації, дає можливість легко змінити ці припущення при зміні наших знань про предметну область. Жорстке кодування припущень про світ на мові програмування призводить до того, що ці припущення не тільки складно знайти і зрозуміти, але і також складно змінити, особливо не програмістам. Крім того, явні специфікації знань в предметній області корисні для нових користувачів, які повинні дізнатися значення термінів предметної області.

Відділення знань предметної області від оперативних знань - це ще один варіант загального застосування онтологій. Ми можемо описати завдання конфігурації продукту з його компонентів відповідно до необхідної специфікацією і впровадити програму, яка робить цю конфігурацію незалежною від продукту і самих компонентів. Після цього ми можемо розробити онтологію компонентів і характеристик ЕОМ і застосувати цей алгоритм для конфігурування нестандартних ЕОМ. Ми також можемо використовувати той же алгоритм для конфігурування ліфтів, якщо ми надамо йому онтологію компонентів ліфта.

Аналіз знань в предметній області можливий, коли є декларативна специфікація термінів. Формальний аналіз термінів надзвичайно цінний як при спробі повторного використання існуючих онтологій, так і при їх розширенні [11].

1.4 Висновки до розділу 1

У цьому розділі ставилась задача розглянути сучасний стан технологій для формалізації взаємодії інтелектуальних агентів у галузі Інтернету речей за допомогою онтологічного інжинірінгу.

Для цього були розглянуті основні принципи і архітектура Інтернету речей, розвиток і перспективи, а також головні проблеми; розглянуті поняття інтелектуальних агентів, класифікація, переваги та використання їх в області Інтернету речей; доведена актуальність формалізації взаємодії інтелектуальних агентів в області Інтернету речей.

2 МОДЕЛІ І МЕТОДИ ДЛЯ ФОРМАЛІЗАЦІЇ ВЗАЄМОДІЇ ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ В ОБЛАСТІ ІНТЕРНЕТУ РЕЧЕЙ

2.1 Теорія інтелектуальних агентів

2.1.1 Визначення поняття агенту та проблемного середовища

В області штучного інтелекту агентом може називатися все що може сприймати свою середу за допомогою датчиків і впливати на цю середу за допомогою виконавчих механізмів**.** Наприклад, людина, що розглядається в ролі агенту, має очі, вуха і інші органи чуття, а виконавчими механізмами для нього служать руки, ноги, рот і інші частини тіла[12]. Ця ідея проілюстрована на рисунку 2.1.

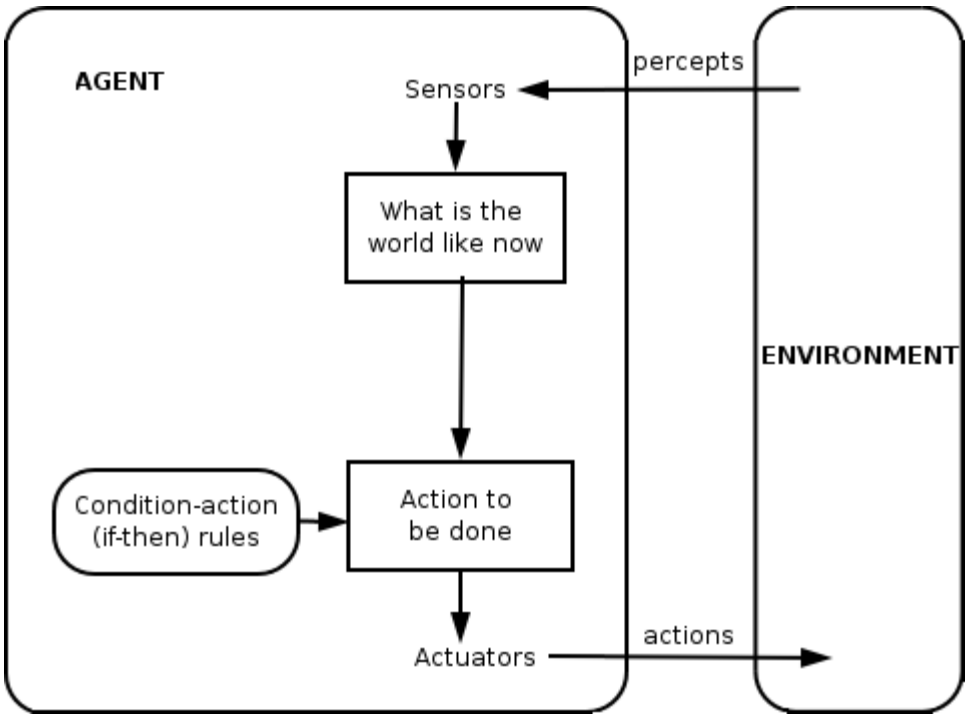

Рисунок 2.1 – Схема роботи агенту

Далі приведені основні поняття пов’язані з агентом.

Сприйняття – дані, які отримані агентом в будь-який конкретний момент часу за допомогою датчиків. Послідовність актів сприйняття агентом – це повна історія всього, що було сприйняте агентом. Вибір агентом дії в будь-який конкретний момент часу може залежати від всієї послідовності актів сприйняття, що спостерігалися до цього моменту часу[12].

Функція агенту – відображення послідовності актів сприйняття на деяку дію. Функція являє собою абстрактне математичне опис, в той час коли програма агенту – це конкретна його реалізація. Архітектура агенту – це

обчислювальний пристрій, на якому працює програма агенту. Внутрішню структуру агенту можна описати за допомогою наступної формули:

Агент = Архітектура + Програма

Показники продуктивності втілюють в собі критерії оцінки успішної поведінки агенту[17]. Після занурення в середу агент виробляє послідовність дій, що відповідають отриманому їм сприйняттю. Ця послідовність дій змушує середу пройти через послідовність станів. Якщо така послідовність відповідає бажаному, то агент функціонує добре.

Проблемне середовище агенту вміщує у собі опис чотирьох компонентів агенту:

- продуктивність;
- оточення;
- виконавчі механізми;
- датчики;

Далі представлений перелік властивостей середи.

За повнотою спостерігання середовище буває повністю спостережним або частково спостережним . Якщо датчики агента надають йому доступ до повної інформації про стан середовища в кожен момент часу, то таке проблемне середовище називається повністю спостережним[18].

Середовище може бути детермінованим або стохастичним. Якщо кожний наступний стан середовища повністю визначається поточним станом і дією, виконаною агентом, то таке середовище називається детермінованим, в іншому випадку воно – стохастичне[2.14].

Середовище може бути епізодичне або послідовне. Якщо кожна наступна дія агента залежить тільки від поточного акту сприйняття, то таке середовище називається епізодичним, якщо для виконання дії необхідно застосувати дані попередніх актів сприйняття – то послідовним[12].

Середовище може бути статичним або динамічним. Якщо середовище може змінитися у ході того, як агент вибирає чергову дію, то таке середовище називається динамічним, в іншому випадку вона є статичним.

Середовище може бути одноагентним чи багатоагентним[12].

На основі складності середовищ та задачі виділяють такі основні типи програм агентів:

- прості рефлексні агенти;
- рефлексні агенти, засновані на моделі;
- агенти, що діють на основі мети;
- агенти, що діють на основі корисності.

Прості рефлексивні агенти – це такий тип агентів, які вибирають дії на основі поточного акту сприйняття, ігноруючи решту історію актів сприйняття.

Рефлексивні агенти, засновані на моделі – агент, який зберігає історію актів сприйняття, що може використовуватись коли середовище є частково спостережним.

Агенти, що діють на основі корисності – це такі агенти які використовують функцію корисності щоб оцінити поточний стан середовища у порівнянні із бажаним.

2.1.2 Інтелектуальний агент

Раціональним агентом є такий агент, який виконує правильні дії. Висловлюючись більш формально, таким є агент, в якому кожен запис в таблиці для функції агенту заповнена правильно.

У будь-який конкретний момент часу оцінка раціональності дій агенту залежить від чотирьох перерахованих нижче факторів:

− показники продуктивності, які визначають критерії успіху.

− знання агенту про середовище, придбані раніше.

− дії, які можуть бути виконані агентом.

− послідовність актів сприйняття агенту, які відбулися до теперішнього часу.

З урахуванням цих чинників можна сформулювати наступне визначення раціонального агента.

Для кожної можливої послідовності актів сприйняття раціональний агент повинен вибрати дію, яка, як очікується, максимізує його показники продуктивності, з урахуванням фактів, наданих даної послідовністю актів сприйняття і всіх вбудованих знань, якими володіє агент.

Для того щоб агент міг діяти раціонально в середовищах, які наближені своєю складністю до реальності він повинен бути інтелектуальним. На ділі інтелектуальність означає, що агент здатний навчатись та має достатню автономність від початкової конфігурації. Початкова конфігурація агента може відображати деякі попередні знання про середовище, але в міру набуття агентом досвіду ці знання можуть модифікуватися і поповнюватися. Існують крайні випадки, в яких середовище повністю відоме заздалегідь. У подібних випадках агенту не потрібно сприймати інформацію або навчатися; він просто відразу діє правильно. Такі агенти є дуже уразливими. Якщо ступінь, в якій агент покладається на апріорні знання свого проектувальника, а не на своє сприйняття, занадто висока, то такий агент розглядається як володіє недостатньою автономністю.

2.1.3 Інтелектуальні агенти, засновані на знаннях

Для агентів, які призначені для взаємодії із простими середовищами, питання їх проектування досить просто вирішується за допомогою побудови відповідних алгоритмів. У випадку, коли агент має існувати у середовищі реальних життєвих задач застосовують агентів заснованих на знаннях.

Агенти, засновані на знаннях, здатні використовувати знання, виражені в дуже загальних формах, комбінуя та рекомбінируя інформацію відповідно до безлічі зовнішніх умов. Часто цей процес може бути дуже далеким від потреб поточного моменту; це можна порівняти з тим, як математик доводить абстрактну теорему або астроном обчислює очікувану тривалість існування Землі[17].

Крім того, знання і міркування відіграють вирішальну роль, коли доводиться діяти в частково спостережуваних варіантах середовища. Агент, заснований на знаннях, здатний поєднувати загальні знання з результатами поточних даних, щоб мати можливість виявити приховані аспекти поточного стану, перш ніж вибирати дії[12].

Такі агенти характеризуються значною гнучкістю. Вони здатні приймати до виконання нові завдання, виражені в формі явно поставлених цілей, вони можуть швидко досягати компетентності, отримуючи інструкції або засвоюючи нові знання, отримані з-поміж себе, крім того, вони здатні пристосовуватися до змін в своєму середовищі, оновлюючи відповідні знання.

Центральним компонентом будь-якого агента, заснованого на знаннях, є його база знань. Неформально базу знань можна визначити як безліч висловлювань. Кожне висловлювання виражено мовою, яка називається мовою подання знань, і представляє деяке твердження про світ[12].

Висловлювання записуються за допомогою логіки.

2.2 Огляд платформи для розробки інтелектуальних агентів JADE

JADE (Java Agent Development Framework) – це програмне забезпечення, платформа з відкритим кодом повністю реалізована мовою Java, для програм, заснованих на агентах. Це спрощує впровадження мультиагентних систем за допомогою проміжного програмного забезпечення, яке відповідає специфікаціям FIPA, та за допомогою набору графічних інструментів, що підтримують етапи налагодження та розгортання. Системи, засновані на JADE, можна розподілити між машинами (які навіть не потребують спільного використання однієї і тієї ж ОС), а конфігурацією можна керувати за допомогою віддаленого графічного інтерфейсу. Конфігурацію можна змінити навіть під час виконання, переміщуючи агентів з однієї машини на іншу, коли потрібно. JADE повністю реалізований на мові Java, мінімальною системною вимогою є Java 5[19].

Платформа JADE заснована на стандартах FIPA. The Foundation for Intelligent Physical Agents (FIPA) – це орган, який розробляє та встановлює стандарти комп'ютерного програмного забезпечення для різнорідних та взаємодіючих агентів та систем, заснованих на агентах. FIPA була заснована як швейцарська неприбуткова організація в 1996 році з амбіційною метою визначити повний набір стандартів для систем, що будуть використовувати агентів у своїй роботі[20].

Принципи дизайну агенту в платформі JADE:

− агент є автономним та ініціативним: агент не може надавати посилання на власний об’єкт іншим агентам для того, щоб зменшити будь-який шанс іншим організаціям впливати на його поведінку. Агент повинен мати власний потік виконання, використовуючи його для контролю свого життєвого циклу та самостійного прийняття рішень.

- агенти можуть сказати «ні», вони слабко пов’язані: асинхронне спілкування на основі повідомлень є основною формою спілкування між агентами в JADE; агент, який бажає спілкуватися, повинен надіслати повідомлення до визначеного пункту призначення (або набору пунктів призначення)[19].

- Peer-to-Peer система: кожен агент ідентифікується глобальним унікальним іменем (AgentIdentifier, або AID, як визначено FIPA). Він може в будь-який час приєднатися і вийти до хост-платформи, може виявити інших агентів за допомогою служб, що надаються в JADE AMS та агентами DF, як це визначено також специфікаціями FIPA. Агент може ініціювати спілкування з будь-яким іншим агентом у будь-який час, коли він цього забажає, і, однаково, може бути об’єктом вхідного спілкування в будь-який час[19].

Основні функціональні можливості JADE[21]:

- повністю розподілена система агентів, де кожен агент працює як окремий програмний потік, потенційно на різних машинах і здатний прозоро

обмінюватися даними, тобто платформа забезпечує унікальний API, незалежний від місцезнаходження, який абстрагує базову комунікаційну інфраструктуру.

− повна відповідність вимогам FIPA. Платформа успішно брала участь у всіх заходах оперативної сумісності FIPA і використовувалася як проміжне програмне забезпечення для багатьох платформ у мережі Agentcities (Agentcities).

− ефективне транспортування асинхронних повідомлень через прозорий API. Платформа вибирає найкращі доступні засоби зв'язку та, коли це можливо, уникає маршалювання та демаршалювання об'єктів Java. При перетині меж платформи повідомлення автоматично трансформуються із власного внутрішнього представлення JADE у власні FIPA-сумісні синтаксиси, кодування та транспортні протоколи.

− просте, але ефективне управління життєвим циклом агента. Коли агенти створюються, їм автоматично присвоюється унікальний глобальний ідентифікатор та транспортні адреса, які використовуються для реєстрації. Простий API та графічні інструменти надаються для локального та віддаленого управління життєвими циклами агента, тобто створення, призупинення, відновлення, заморожування, міграція, клонування та видалення.

− підтримка мобільності агента. Код агента та, за певних обмежень, стан агента можуть мігрувати між процесами та машинами. Міграція агентів робиться прозорою для комунікаційних агентів, які можуть продовжувати взаємодіяти навіть під час процесу міграції.

− механізм підписки для агентів і навіть зовнішніх додатків, які хочуть підписатися на платформу, щоб отримувати повідомлення про всі події платформи, включаючи події, пов’язані з життєвим циклом та події обміну повідомленнями.

− наявність графічних інструментів для підтримки програмістів при налагодженні та моніторингу. Є можливість емулювати різні ситуації, а виконанням агента можна керувати віддалено та інтроспекційно.

− підтримка онтологій. Перевірка онтологій та кодування вмісту виконується платформою автоматично. Програмістами можуть вибрати бажані мови онтології (наприклад, на основі XML та RDF). Програмісти також можуть впроваджувати нові мови онтологій, щоб задовольнити конкретні вимоги до програми.

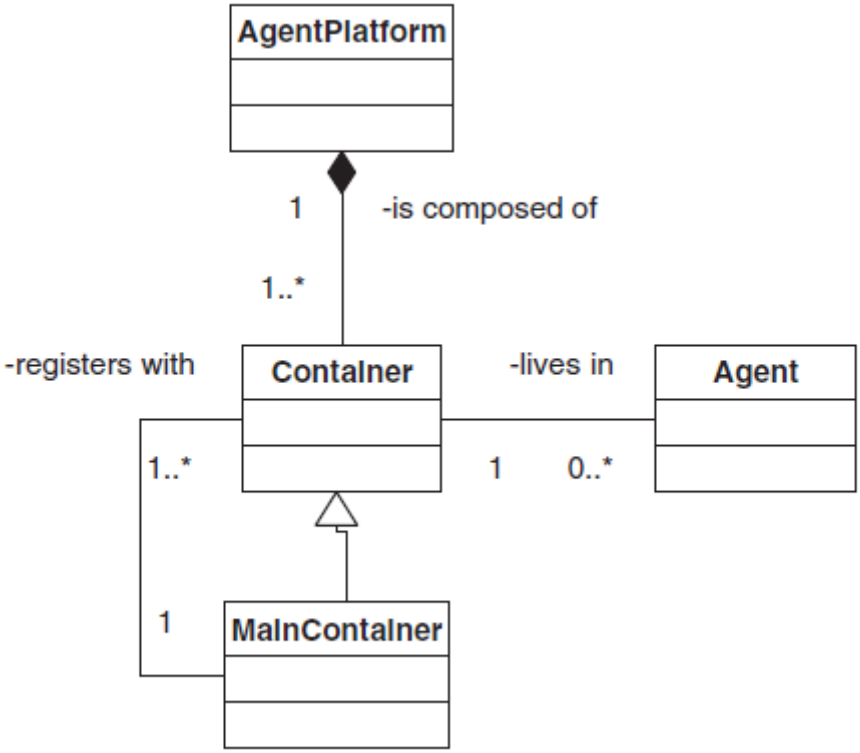

Рисунок 2.2 – Основні компоненти архітектури JADE

Платформа JADE складається з контейнерів агентів, які можуть бути розподілені по мережі. Агенти існують у контейнерах, які є процесом Java, що забезпечує роботу JADE та все, що необхідно для розміщення та виконання агентів. Також є спеціальний контейнер, який називається основний контейнер, який представляє стартову точку завантаження платформи: це перший контейнер, який запускається, а всі інші контейнери повинні приєднатися до основного контейнера, зареєструвавшись у ньому. Діаграма UML на малюнку 2.2 схематизує взаємозв'язки між основними архітектурними елементами JADE.

2.3 Теоретичні відомості про поняття онтології

2.3.1 Визначення поняття онтології

Онтології використовуються для опису та представлення знань деякої предметної області, домену. Онтологія необхідна для спільного розуміння знань домену та складається з понять, що використовуються для опису знань домену та взаємозв'язків між цими поняттями. Кожна галузь науки має свою онтологію[23].

Неформально онтологія представляє собою деякий опис погляду на світ стосовно конкретної області інтересів. Це опис складається з термінів і правил використання цих термінів, що обмежують їх значення в рамках конкретної області.

На формальному рівні онтологія – це система, що складається з набору понять і набору тверджень про ці поняття, на основі яких можна описувати класи, відносини, функції та індивіди[27].

Інше, більш точне визначення, виглядає наступним чином: онтологія – це точна специфікація концептуалізації.

Концептуалізація – це структура реальності, що розглядається незалежно від словника предметної області і конкретної ситуації[24].

Наприклад, якщо ми розглядаємо просту предметну область, що описує кубики на столі, то концептуалізації є набір можливих положень кубиків, а не конкретне їх розташування в поточний момент часу.

Можна виділити наступні причини побудови онтології[27]:

− спільне розуміння знань домену (в межах спільноти людей, маючих єдину мету);

− обмін знаннями про домен;

− повторне використання знань домену;

− аналіз знань домену;

− відокремлення доменних знань від оперативних;

− виразні припущення щодо домену;

Для ефективного спілкування користувачі, програми та агенти повинні мати спільне розуміння термінів, що використовуються у спілкуванні. Терміни повинні мати однакове значення для всіх сторін, що беруть участь у спілкуванні. Для цього потрібно забезпечити, щоб усі сторони, що беруть участь у спілкуванні, мали однакову онтологію.

Інтелектуальні агенти з різною внутрішньою архітектурою можуть спілкуватися між собою за допомогою загальної онтології. Онтологія є машиночитаною і підтримує комунікацію агента шляхом визначення та надання спільного словника, який буде використовуватися в процесі спілкування. Онтології не лише дають визначення термінів, які можна використовувати в спілкуванні; онтології також дають визначення світу, в

якому агент обґрунтовує свої дії. Різні агенти системи можуть досягти спільного розуміння, дотримуючись однієї і тієї ж онтології.

Нижче наведені основні властивості онтологій[26]:

− онтології описують окремо взятий домен;

− користувачі онтології погоджуються використовувати терміни онтології послідовно;

− поняття та відносини онтології однозначно визначаються формальною мовою за допомогою аксіом та визначеннь.

- взаємозв'язки між поняттями онтології визначають структуру онтології, наприклад ієрархічні або неієрархічні. Відносини генералізації та спеціалізації між двома поняттями є прикладом ієрархічності взаємозв'язок між поняттями.

− онтології повинні бути зрозумілі комп’ютером.

2.3.2 Характеристика онтологічної моделі

Основними компонентами онтології можуть бути:

- класи (або поняття),
- відносини (або властивості, атрибути),
- функції,
- аксіоми,
- екземпляри.

Класи або поняття використовуються в широкому сенсі. Поняттям може бути будь-яка сутність, про яку може бути дана будь-яка інформація. Класи – це абстрактні групи, колекції або набори об'єктів. Вони можуть включати в себе екземпляри, інші класи, або ж поєднання і того, і іншого. Класи в онтологіях зазвичай організовані в таксономії – ієрархічну класифікацію понять по відношенню включення. Наприклад, класи «Чоловік» і «Жінка» є підкласами класу «Людина», яка в свою чергу входить в клас «Ссавці»[23].

Відносини представляють тип взаємодії між поняттями предметної області. Формально n-арні відносини визначаються як підмножина множин: $R \subseteq C1 \times C2 \times ... \times Cn$. Приклад бінарного відносини - відношення «частина-ціле». Відносини теж можуть бути організовані в таксономії по включенню; наприклад, відносини «бути татом для» і «бути мамою для» на безлічі людей містяться в відношенні «буть батьком для», яке в свою чергу міститься в відношенні «буть предком для»[23].

Функції – це спеціальний випадок відносин, в яких n -й елемент відносини однозначно визначається n-1 попередніми елементами. Формально функції визначаються наступним чином: $F: C1 \times C2 \times ... \times Cn-1 \rightarrow Cn$. Прикладами функціональних відносин є відносини «бути мамою для» на безлічі людей, або «ціна автомобіля», яка обчислюється залежно від моделі автомобіля, дати виготовлення і пробігу[25].

Аксіоми використовуються, щоб записати висловлювання, які завжди правдиві. Вони можуть бути включені в онтологію для різних цілей, наприклад,

для визначення комплексних обмежень на значення атрибутів, аргументи відносин, для перевірки коректності інформації, описаної в онтології, або для виведення нової інформації [11].

Представлення знань змінюється з часом. Рівень деталізації онтологій пов'язаний із рівнем деталізації інформації. Часто доводиться фільтрувати та опускати деталі інформації, щоб зробити ситуацію зрозумілішою, представляючи в контексті лише інформацію, що стосується концепції.

Для представлення онтологій переважно використовуються ієрархії (рисунок 2.3). Існує дві основні причини для цього:

– ієрархії подібні тому, як ми упорядковуємо моделі світу у своєму розумі;

– ієрархії забезпечують механізм генералізації та спеціалізації в обробці інформації.

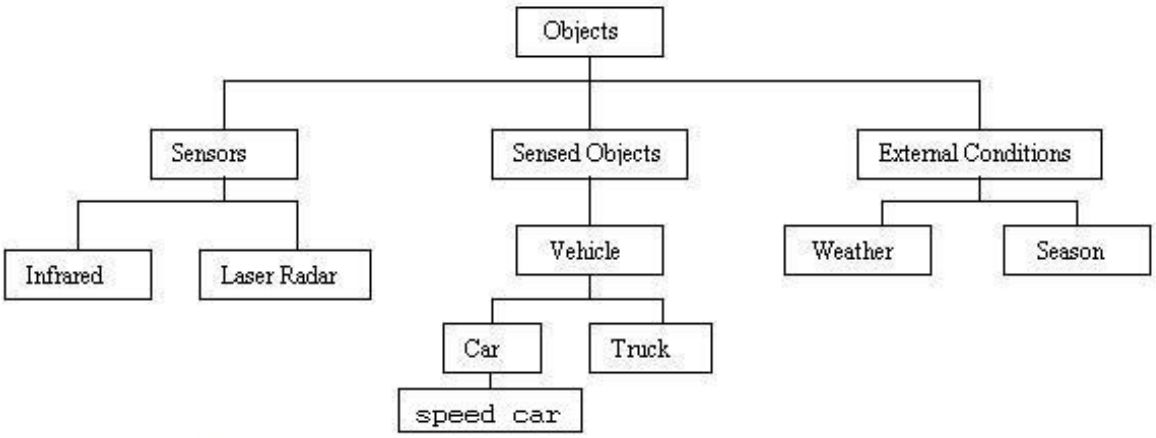

Рисунок 2.3 – Приклад таксономії

Хоча онтології часто набувають форми таксономічної ієрархії класів, вони жодним чином не обмежуються ієрархіями. Насправді онтології можуть набувати набагато більш загальних і складних структур. Поняття онтології можуть бути згруповані разом, щоб сформувати шари відповідних дерев ієрархії, мереж, кубічних структур або будь-яких інших форм. Спосіб групування концепцій онтології залежить від мети проекту онтології та складності знань, які вона представляє.

2.3.3 Генералізація та спеціалізація онтологій

Загальні онтології містять менш детальну інформацію, ніж спеціалізовані онтології. Зазвичай більш детальні спеціалізовані онтології виводяться із загальних онтологій (рис. 2.4). Ці нові спеціалізовані онтології є більш детальними, оскільки вони вдосконалюють і розширюють загальні описи, присутні в загальних онтологіях[23].

Загальна онтологія, швидше за все, буде доступною для великої кількості спільнот. Зазвичай спеціалізовані онтології призначені для меншої спільноти всередині більшої спільноти. Загальна онтологія складається з мінімальної

кількості аксіом, тоді як спеціалізована онтологія має велику кількість аксіом і потребує дуже виразної мови. Можна поступово переходити від загальної до спеціалізованої онтології шляхом поступового збільшення кількості аксіом.

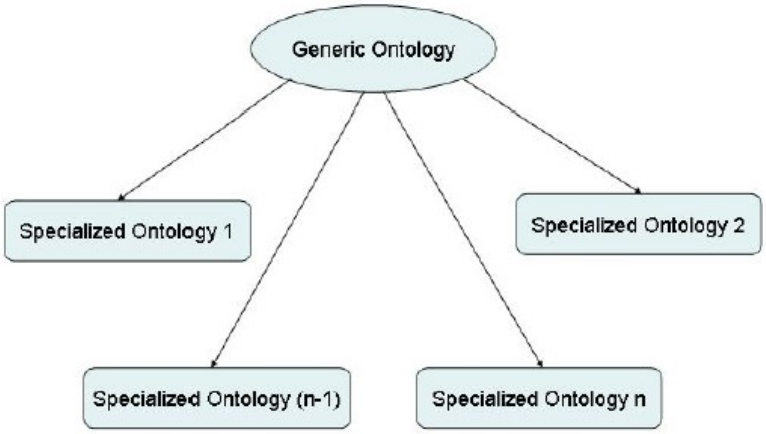

Рисунок 2.4 – Приклад спеціалізації онтології

Також можливо побудувати загальну онтологію, яка базується на ряді спеціалізованих онтологій. Такий процес називається «генералізація онтології» і показаний на рисунку 2.5. Цей процес спрямований на вирішення конфліктів між визначеннями термінів в онтологіях. Результатом такого процесу є онтологія, яку можуть прийняти більші спільноти.

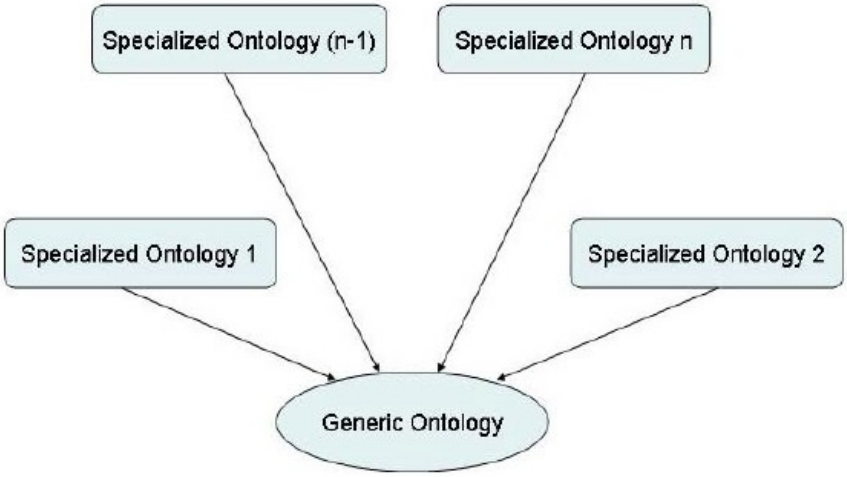

Рисунок 2.5 – Приклад генералізації онтології

2.3.4 Мови онтологій

Мова онтології – це формальна мова, яка використовується для кодування онтологій так, щоб вони могли бути зрозумілі комп’ютером. В останні роки було створено низку дослідницьких мов, які можуть використовуватися для кодування онтологій, таких як загальна Алгебраїчна мова специфікацій, загальна логіка, CycL, DOGMA, Gellish, IDEF5, KIF, RIF та OWL.

Мови онтогій можуть буть класифіковані наступним чином:

1 Логічні мови:

– логіка предикатів першого порядку,

– логіка на основі правил,

– логіка опису.

2 Мови на основі фреймів:

– подібно до реляційних баз даних.

3 Мови на основі графіків:

– семантична мережа (аналогія з Інтернетом є основним обґрунтуванням для семантичної мережі).

Далі навединий огляд деяких мов онтологій.

2.3.4.1 Knowledge Interchange Format (KIF)

KIF – це комп'ютерна мова, розроблена для того, щоб дати змогу системам обмінюватися інформацією із системами, заснованими на знаннях, та повторно використовувати їх. KIF подібний до мов фреймів, таких як KL-One та LOOM, але на відміну від них, основна роль KIF не призначена для вираження чи використання знань, а скоріше для обміну знаннями між системами. Дизайнери KIF порівняли це з PostScript. PostScript не був розроблений насамперед як мова для зберігання та обробки документів, а скоріше як формат обміну для систем та пристроїв для спільного використання документів. Таким же чином KIF призначений для сприяння обміну знаннями між різними системами, що використовують різні мови, формалізми, платформи тощо.

KIF має декларативну семантику. Він призначений для опису фактів про світ, а не процесів чи процедур. Знання можна описати як об'єкти, функції, відношення та правила. Це формальна мова, тобто вона може виражати довільні твердження в логіці першого порядку і може підтримувати аргументи, які можуть довести узгодженість набору заяв KIF. KIF також підтримує немонотонні міркування. KIF був створений Майклом Дженезеретом, Річардом Фікесом[24].

2.3.4.2 DOGMA

DOGMA – це скорочення від «Developing Ontology-Grounded Methods and Applications», - це назва дослідницького проекту, що виконується в лабораторії досліджень STARLab Vrije Universiteit у Брюсселі. Це внутрішньо фінансуваний проект, що стосується більш загальних аспектів пошуку, зберігання, представлення та перегляду інформації. DOGMA, як діалект підходу, що базується на фактах, має коріння в семантиці баз даних і теорії моделей. DOGMA дотримується методології управління інформацією, що базується на фактах.

Методологічні принципи DOGMA включають:

– незалежність даних: значення даних слід відокремлювати від самих даних.

– незалежність інтерпретації: унарні або бінарні типи фактів (тобто лексони) повинні дотримуватися формальної інтерпретації з метою збереження семантики; самі лексони не мають семантики;

– кілька видів та використання збереженої концептуалізації. Онтологія повинна бути масштабованою та розширюваною.

– онтологія повинна бути мовно нейтральна, повинна відповідати багатомовним потребам.

– незалежність представлень: онтологія в DOGMA повинна відповідати будь-яким потребам користувачів у представленні.

– поняття повинні перевірятися зацікавленими сторонами.

– неформальні текстові визначення понятть тощо повинні надаватися у випадку, якщо джерело онтології відсутнє або неповне.

2.3.4.3 Web Ontology Language (OWL)

Мова веб-онтології (OWL) - це сімейство мов подання знань для створення онтологій. Мови OWL характеризуються формальною семантикою. Вони побудовані на основі стандарту XML для об’єктів, що називаються Resourse Description Framework (RDF). OWL та RDF привернули значний науковий, медичний та комерційний інтерес. Специфікація OWL включає визначення трьох варіантів OWL з різним рівнем виразності. Це OWL Lite, OWL DL і OWL Full.

Дані, описані онтологією OWL, інтерпретуються як сукупність "індивідів" та набір "тверджень про власність", які пов'язують цих індивідів між собою. Онтологія OWL складається з набору аксіом, які накладають обмеження на набори індивідів (які називаються «класами») та типи взаємозв'язків, дозволених між ними. Ці аксіоми забезпечують семантику, дозволяючи системам робити висновки про додаткову інформацію на основі явно наданих даних. OWL є одночасно синтаксисом для опису та обміну онтологіями та має формально визначену семантику, яка надає їм значення.

2.4 Еталонна модель IoT

У цьому розділі описана загальна архітектура речі, загальні компоненти та абстракції з точки зору IoT ARM (Architectural Reference Model)[28], які будуть використовуватись у розробці онтології для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей.

2.4.1 Концепція та задачі еталонної моделі IoT

Сьогодні поняття Інтернету речей використовується як загальний вираз у багатьох джерелах. Воно охоплює дуже багато рішень, так чи інакше пов'язаних зі світом взаємозв'язку розумних об'єктів. Ці рішення мають незначні можливості взаємної сумісності або взагалі не мають їх, оскільки зазвичай вони розробляються з урахуванням конкретних проблем, дотримуючись конкретних вимог. Більше того, оскільки концепція IoT охоплює абсолютно різні галузі застосування, виявляється, що цикли розробки та використання технології можуть надзвичайно різнитися. Як наслідок, з’являються винятково вертикальні та поодинокі рішення, які не мають спільного технічного обґрунтування та загальних архітектурних принципів, що в кінцевому підсумку необхідно для повноцінного Інтернету речей[28].

І насправді саме в цьому є справжня проблема: розумність додатків у галузі Інтернету речей може бути досягнута лише тоді, коли буде досягнута повна співпраця між ними.

Також необхідно взяти до уваги той факт, що технології, пов’язані з IoT, мають високий рівень неоднорідності, вони засновані на окремих протоколах, розроблених з урахуванням конкретних застосувань, це призводить до того, що в наш час галузь IoT виглядає дуже фрагментованою. На сьогоднішній день існує багато рішень в таких областях областях як транспорт, енергетика, навколишнє середовище та за більшою частиною з них заздалегідь закріплена приставка "розумний", щоб підкреслити той факт, що вони мають своєрідний інтелект. Однак вони в цілому вони сьогодні утворюють лише те, що можна було б назвати Інтернетами речей, а не Інтернетом речей.

Сумісність рішень на комунікаційному рівні, а також на рівні сервісу, повинна бути забезпечена на різних платформах. Як і в галузі мереж, де на початковому етапі було кілька рішень, які потім залишили місце загальній моделі, набору протоколів TCP/IP, поява загальної еталонної моделі для домену IoT та вибір еталонних архітектур можуть призвести до швидшого, та більш цілеспрямованого розвитку та експоненціального збільшення рішень, пов’язаних із IoT[28].

З урахуванням вище описаних проблем була створена IoT RM (IoT Reference model), або еталонна модель IoT, яка пропонує концепції та поняття, на основі яких можуть бути побудовані рішення в галузі IoT.

IoT RM спрямована на встановлення спільного обґрунтування, спільної мови для розробки архітектур IoT та систем IoT. Вона складається з інших підмоделей, показаних на рисунку 3.1. Основною з них є доменна модель (IoT

Domain model), яка описує усі концепції, що грають ключову роль у галузі Інтернеті речей. Усі інші підмоделі базуються на концепціях, представлених у доменній моделі. Хоча певні моделі, такі як модель зв'язку (IoT Communication Model) та модель безпеки та конфіденційності (IoT Trust, Security Model) можуть бути менш критичними у певних сценаріях застосування, сама модель домену є обов’язковою у IoT ARM.

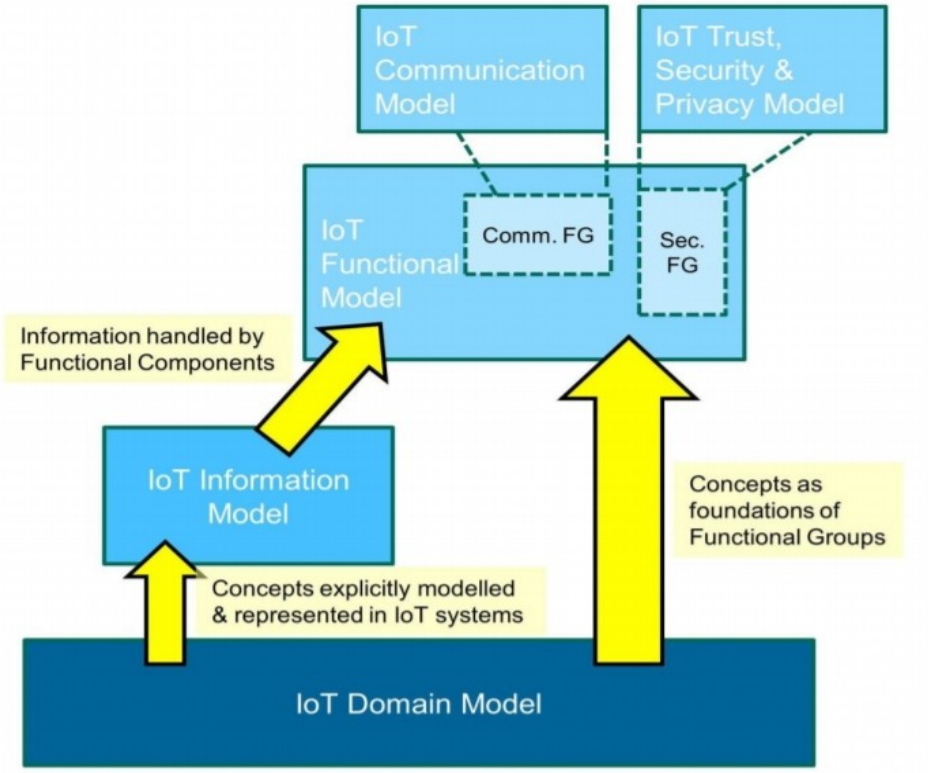

Рисунок 2.6 - Взаємодія підмоделей еталонної моделі IoT

Нижче приведено опис підмоделей еталонної IoT.

Основою еталонної моделі IoT є модель домену IoT, яка представляє основні поняття Інтернету речей, такі як пристрої, сервіси IoT та віртуальні сутності речей, а також встановлює взаємозв'язок між цими поняттями. Рівень абстракції доменної моделі IoT був обраний таким чином, щоб її концепції не залежали від конкретних технологій та випадків використання. Ідея полягає в тому, що протягом наступних десятиліть чи довше ці концепції не будуть сильно змінюватися[28].

На основі моделі домену IoT була розроблена інформаційна модель IoT (IoT Informational model). Вона визначає структуру інформації (наприклад, відносини та атрибути), пов’язаної з Інтернетом речей, у системі Інтернету речей на концептуальному рівні, не обговорюючи, як вона буде представлена. Інформація, що стосується концепцій представлених у моделі домену IoT, моделюється, збирається, зберігається та обробляється в системі IoT, наприклад інформація про пристрої, послуги IoT та віртуальні сутності.

Функціональна модель IoT визначає групи функціональних можливостей, більшість з яких ґрунтуються на ключових концепціях моделі домену IoT. Ряд цих функціональних груп (FG) базуються одна на одній, дотримуючись

відносин, визначених у моделі домену IoT. Групи функціональності надають функціональні можливості для взаємодії з екземплярами цих концепцій або управління інформацією, що стосується цих концепцій, наприклад інформація про віртуальні сутності або описи служб Інтернету речей. Функціональні можливості, які управляють інформацією, використовують Інформаційну модель IoT як основу для структурування своєї інформації.

Ключовою функцією будь-якої розподіленої комп'ютерної системи є зв'язок між різними компонентами. Однією з характеристик систем IoT часто є неоднорідність використовуваних комунікаційних технологій, що часто є прямим відображенням складних потреб, які такі системи повинні задовольняти. Комунікаційна модель IoT вводить концепції управління складністю спілкування в неоднорідних середовищах IoT. Зв'язок також становить одну з функціональних груп у функціональній моделі IoT.

Довіра, безпека та конфіденційність надзвичайно важливі у типових сценаріях використання IoT. Тому відповідні функціональні можливості та їх взаємозалежність та взаємодія представлені в моделі TSP(Trust, Security and Privacy). Як і у випадку з комунікацією, безпека становить ще одну функціональну групу у функціональній моделі28].

Для розробки онтології для формалізації інтелектуальних агентів в галузі Інтернету речей у наступних розділах буде розглянуті доменна та інформаційна моделі IoT RM.

2.4.2 Модель домену IoT

Поняття моделі домену можна визначити як опис концепцій, що належать до певної сфери інтересів. Модель домену також визначає основні атрибути цієї сфери, такі як назва та ідентифікатор. Крім того, модель домену визначає взаємозв'язки між поняттями, наприклад "сервіси представляють ресурси". Моделі доменів також допомагають полегшити обмін даними між доменами[29].

Основна мета моделі домену полягає у формуванні загального розуміння цільового домену, про який йде мова. Таке загальне розуміння важливо не лише для внутрішнього проекту, але й для наукового дискурсу в цілому. Тільки при спільному розумінні основних концепцій стає можливим розмова щодо архітектурних рішень та їх оцінка. Як вже зазначалося, домен IoT вже страждає від непослідовного використання та розуміння значення багатьох центральних термінів.

Доменна модель є важливою частиною будь-якої еталонної моделі, оскільки вона включає визначення основних абстрактних понять (абстракцій), їх обов’язків та взаємозв’язків. Що стосується рівня деталізації, модель домену повинна відокремлювати те, що не сильно відрізняється від того, що робить. Наприклад, у домені IoT концепція пристрою, ймовірно, залишатиметься актуальною в майбутньому, навіть якщо типи використовуваних пристроїв змінюватимуться з часом або змінюватимуться залежно від контексту програми. Наприклад, існує безліч технологій ідентифікації об’єктів: RFID, штрих-коди, розпізнавання зображень тощо. Але яка з них все ще буде використовуватися через 20 років? І яка технологія найкраще підходить для конкретного застосування? Оскільки ніхто не має відповідей на подібні та супутні питання, модель домену IoT не включає певні технології, а скоріше їх абстракції[29].

2.4.2.1 Основні поняття моделі домену IoT

На рисунку 2.7 зображена UML діаграма моделі домену IoT, де

– жовтим кольором позначено людей або тварин;

– синім -- апаратне забезпечення;

– зеленим -- програмне забезпечення;

– золотим -- комбінацію декількох категорій;

В наступних підрозділах наведено опис основних понять доменної модель IoT.

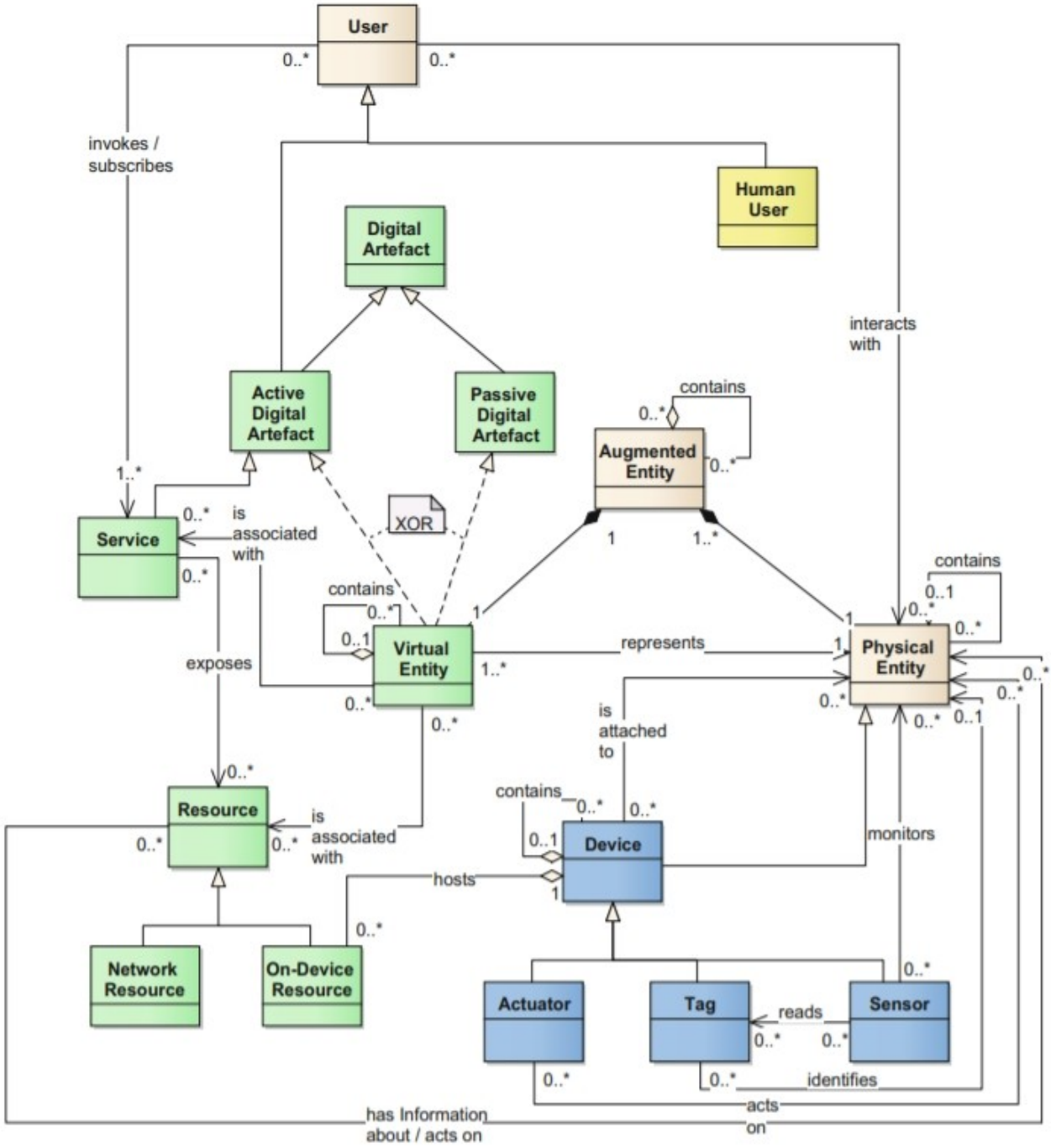

Рисунок 2.7 ─ UML діаграма моделі домену IoT

2.4.2.2 Поняття користувача, віртуальної та фізичної сутності у моделі домену IoT

Найбільш загальний сценарій IoT, зображений на малюнку 2.8, можна визначити як сценарій, коли деякому користувачу (User, див рис. 2.8) потрібно взаємодіяти з (можливо віддаленим) фізичним суб'єктом (Physical Entity, див рис. 2.8), річчю, у фізичному світі. Таким чином можна одразу ввести дві ключові сутності IoT. Користувач це людська особа або якийсь цифровий артефакт (наприклад, сервіс, додаток або агент програмного забезпечення), який повинен взаємодіяти з фізичною об’єктом або річчю[30].

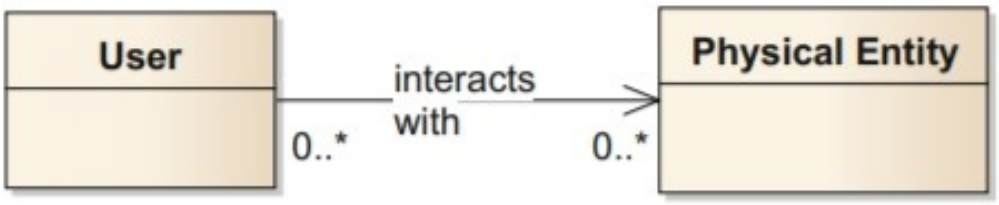

Рисунок 2.8 — UML діаграма загального сценарію IoT

У фізичному середовищі взаємодія може відбуватися безпосередньо (наприклад, переміщенням об’єкту з місця X до Y вручну). Однак у IoT ми хочемо мати можливість взаємодіяти опосередковано або за посередництвом, тобто звернувшись до відповідного сервісу, який надаватиме інформацію про фізичний об’єкт та діятиме безпосередньо на нього. Коли людина-користувач отримує доступ до сервісу, вона робить це через клієнта сервісу, через програмне забезпечення з доступним інтерфейсом користувача. Для сфери дії доменної моделі IoT взаємодія, як правило, характеризується ціллю, яку встановлює користувач.

Фізична об’єкт (річ), можна визначити, як ідентифіковану частину фізичного середовища, яка представляє інтерес для користувача для досягнення його мети. Фізичними об’єктами може бути майже будь-яка річ або середовище; від людей чи тварин до автомобілів; від предметів магазину або логістичної мережі до комп’ютерів; від електронних приладів до ювелірних виробів чи одягу.

В цифровому світі фізичні сутності представлені віртуальними сутностями (Virtual Entity, див. рис. 2.8). Існує багато видів цифрових відображень фізичних сутностей: 3D-моделі, аватари, записи у базі даних, об’єкти (або екземпляри класу об’єктно-орієнтованою мовою програмування), і навіть обліковий запис соціальної мережі можна розглядати як таке відображення, тому що він в цифровій формі представляє певні аспекти людини, наприклад, фотографію або список його інтересів. Однак у контексті IoT віртуальні сутності мають дві основні властивості[30]:

– цифрові артефакти. Віртуальні сутності, як правило, асоційовані з однією фізичною сутністю, хоча для кожного віртуального об'єкта може існувати лише

один фізичний об'єкт, можливо, один і той же фізичний об'єкт може бути пов'язаний з кількома віртуальними об'єктами, наприклад, різними репрезентаціями на кожний домен програми. Кожна віртуальна сутність повинна мати один і лише один ідентифікатор, який однозначно ідентифікує її. Віртуальні об'єкти - це цифрові артефакти, які можна класифікувати як активні, так і пасивні. Активні цифрові артефакти (ADA) - це програмні додатки, агенти або сервіси, які можуть отримати доступ до інших сервісів або ресурсів. Пасивні цифрові артефакти (PDA) - це пасивні елементи програмного забезпечення, такі як записи в базі даних, які можуть бути цифровими зображенням фізичної особи. Зверніть увагу, що всі цифрові артефакти можна класифікувати як активні, так і пасивні цифрові артефакти;

– синхронізація. В ідеалі, віртуальні сутності - це синхронізовані відображення заданого набору характеристик (або властивостей) фізичної сутності. Це означає, що відповідні цифрові параметри, що представляють характеристики фізичного об’єкту, оновлюються при будь-яких змінах перших. Таким же чином зміни, що впливають на віртуальну сутність, можуть проявлятися у фізичній сутності. Наприклад, блокування дверей вручну може призвести до зміни стану дверей в програмному забезпеченні для домашньої автоматизації, і, відповідно, встановлення дверей на «заблоковані» в програмному забезпеченні може призвести до спрацьовування електричного замку у фізичному світі.

Слід зазначити, що, хоча на перший погляд модель домену IoT передбачає взаємодію лише людини-користувача з деякими фізичними сутностями, вона також охоплює взаємодію між двома машинами: у цьому випадку управляюче програмне забезпечення першої машини є активним цифровим артефактом і, отже, користувачем, а другий апарат - або пристрій в умовах моделі домену IoT - може бути змодельована як фізична сутність. Таким чином можна ввести поняття поняття розширеної сутності (Augmented Entity, див рис. 2.8) як композицію віртуальної сутності та фізичної сутності, з якою вона пов’язана, щоб підкреслити той факт, що ці дві концепції належать один до одного. Розширена сутність - це те, що насправді дозволяє повсякденним об’єктам стати частиною цифрових процесів, отже, розширена сутність може розглядатися як складова «річ» в Інтернеті речей.

2.4.2.3 Поняття пристрою, ресурсу та сервісу у моделі домену IoT

Зв'язок між віртуальною сутністю та фізичною сутністю зазвичай досягається шляхом вбудовування, приєднання або просто розміщення в безпосередній близькості від фізичного об’єкта одного або декількох пристроїв (Device), що забезпечують технологічний інтерфейс для взаємодії або отримання інформації про фізичну сутність. Завдяки цьому пристрій фактично розширює фізичну сутність і дозволяє останній бути частиною цифрового світу. Таким чином, пристрій опосередковує взаємодію між фізичними сутностями (які не мають проекцій у цифровому світі) та віртуальними сутностями (які не мають проекцій у фізичному світі), утворюючи пару, яку можна розглядати як

продовження будь-якої з них, тобто розширена сутність. Таким чином, пристрої є технічними артефактами для поєднання реального світу фізичних сутностей із цифровим світом Інтернету. Це робиться шляхом забезпечення можливостей контролю, зондування, спрацьовування, обчислення, зберігання та обробки.

З точки зору IoT можна виділити такі три основні типи пристроїв:

- Датчики надають інформацію, знання або дані про фізичну особу, яку вони контролюють. У цьому контексті це може варіюватися від ідентифікації фізичної сутності до виміру її фізичного стану. Як і інші пристрої, вони можуть бути прикріплені або іншим чином вбудовані у фізичну структуру фізичної сутності, або розміщені в навколишньому середовищі та опосередковано контролювати фізичні сутності. Прикладом є камера для розпізнавання облич[28].
- Теги використовуються для ідентифікації фізичних осіб, до яких вони зазвичай фізично прикріплені. Процес ідентифікації називається "зчитуванням", і він здійснюється за допомогою певних сенсорних пристроїв, які зазвичай називають зчитувачами. Основна мета тегів - полегшити та підвищити точність процесу ідентифікації. Цей процес може бути оптичним, як у випадку зі штрих-кодами та QR-кодами, або може бути на основі RF, як у випадку з мікрохвильовими системами розпізнавання автомобільних табличок та RFID.
- Актуатори можуть модифікувати фізичний стан фізичної сутності, наприклад, змінювати стан (переміщувати, обертати, перемішувати, надувати) простих фізичних сутностей або вмикати та вимикати функціональні можливості більш складних.

Ресурси - це програмні компоненти, програмне забезпечення які надають дані або використовуються для активації фізичних сутностей. Ресурси, як правило, мають власні інтерфейси. Існує різниця між вбудованими ресурсами та мережевими ресурсами. Як випливає з назви, вбудовані ресурси розміщуються на пристроях, тобто це програмне забезпечення, яке розгортається локально на пристрої, пов’язаному з фізичною сутністю. Вони включають виконуваний код для доступу, обробки та зберігання інформації про датчики, а також код для управління виконавчими механізмами. З іншого боку, мережеві ресурси - це ресурси, доступні десь у мережі, наприклад, у хмарі. Віртуальна сутність також може бути пов’язана з ресурсами, що забезпечують взаємодію з фізичною сутністю, яку представляє віртуальна сутність.

На відміну від різнорідних ресурсів, реалізація яких може сильно залежати від базового обладнання пристрою, сервіс надає відкритий та стандартизований інтерфейс, пропонуючи всі необхідні функціональні можливості для взаємодії з ресурсами та пристроями, пов’язаними з фізичними сутностями. Взаємодія із сервісом здійснюється через мережу.

Оскільки саме сервіс робить ресурс доступним, відносини між ресурсами та віртуальними сутностями моделюються як асоціації між віртуальними сутностями та сервісами. Для кожної віртуальної сутності можуть існувати асоціації з різними сервісами, які можуть надавати різні функціональні можливості, такі як отримання інформації або забезпечення виконання завдань активації. Також один сервіс може надаватися різними екземплярами пристроїв.

У цьому випадку для однієї і тієї ж віртуальної сутності може бути кілька асоціацій одного типу.

2.4.3 Інформаційна модель IoT

Інформаційна модель IoT визначає структуру (наприклад, відносини, атрибути, сервіси) всієї інформації для віртуальних сутностей на концептуальному рівні. Термін інформація використовується поряд із визначеннями даних, де дані визначаються як чисті значення без відповідного корисного контексту. Інформація ж у свою чергу додає правильний контекст до даних і пропонує відповіді на типові запитання, наприклад, хто, що, де і коли.

Опис представлення інформації (наприклад, XML, RDF тощо) та конкретні реалізації не є частиною інформаційної моделі IoT.

На рисунку 2.9 наведена UML діаграма інформаційної моделі IoT RM.

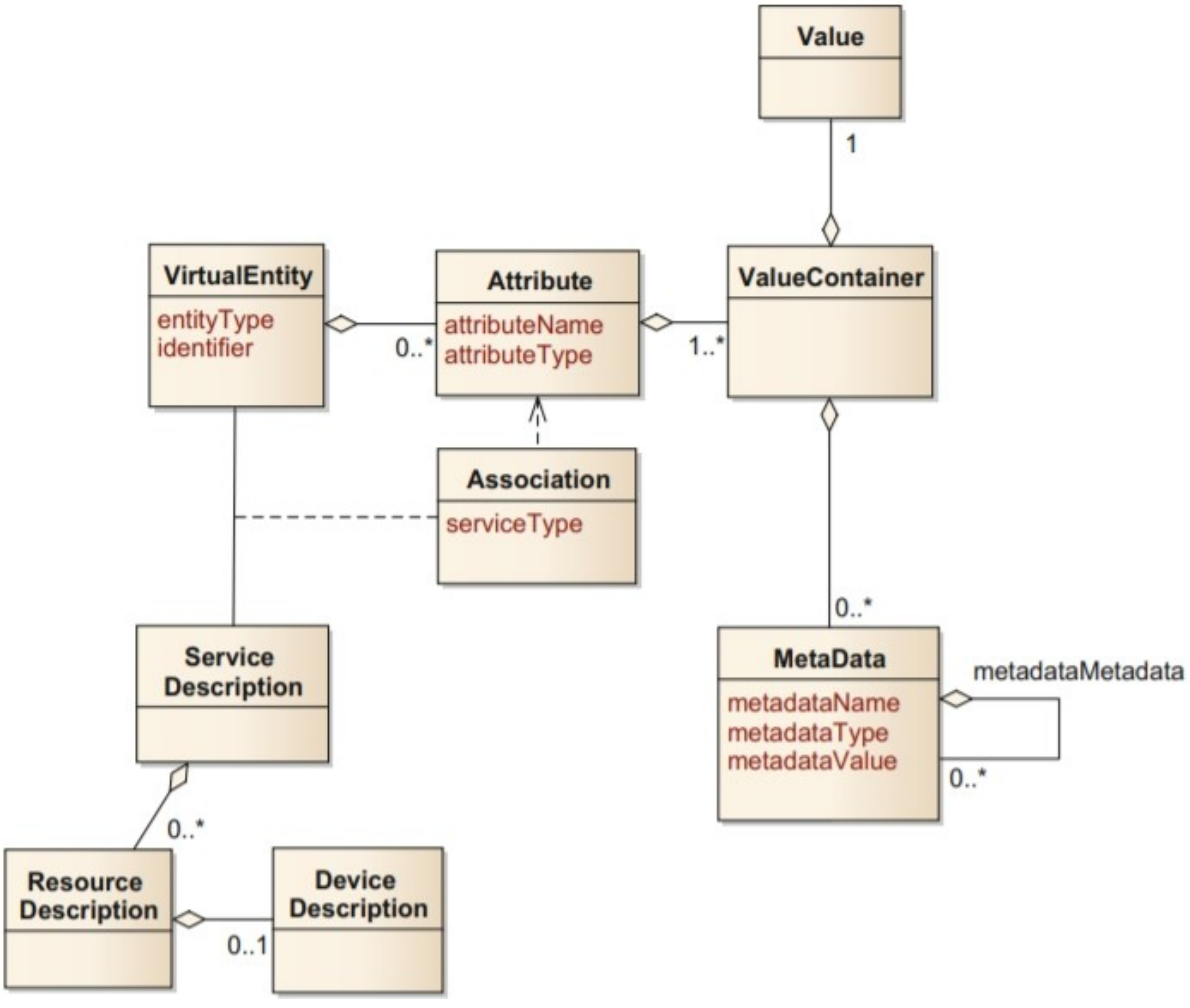

Рисунок 2.9 — UML діаграма інформаційної моделі IoT RM

Інформаційна модель IoT детально описує моделювання віртуальної сутності. Віртуальна сутність (VirtualEntity, див розділ 2.4.2.2) має атрибути з іменем (відповідно Attribute та AttributeName, див рис. 2.9) і типом (AttributeType, див рис. 2.9) та одним або кількома значеннями (Value, див рис.

2.9), до яких може бути прив’язана метаінформація (MetaData, див рис. 2.9). Важливою метаінформацією є, наприклад, в який час було виміряно значення (тобто позначка часу), місце, де відбулося вимірювання, або якість вимірювання. Метадані можуть самі містити додаткові метадані, наприклад одиниця вимірювання метаданих. Асоціація (Association, див рис. 2.9) між віртуальною сутністю та сервісом встановлена у тому сенсі, що він надає доступ до певного атрибуту віртуальної сутності. Тип сервісу (serviceType, див рис. 2.9) може бути встановлений як “INFORMATION”, якщо сервіс надає значення атрибута для зчитування, або “ACTUATION”, якщо сервіс дозволяє встановлювати значення атрибуту, як результат відповідної зміни у фізичному світі[28].

Діаграма на рисунку 2.9 показує структуру інформації, яка обробляється в системі IoT. Основні аспекти представлені елементами VirtualEntity, ServiceDescription та Association. Віртуальна сутність моделює фізичну сутність, а опис сервісу описує службу, яка обслуговує інформацію про саму фізичну сутність або навколишнє середовище. Через асоціацію моделюється зв’язок між атрибутом віртуальної сутності та описом сервісу, наприклад сервіс діє як функція «отримати» для значення певного значення атрибуту фізичної сутності.

Кожна віртуальна сутність повинна мати унікальний ідентифікатор (Identifier) тип сутності (entityType), що визначає тип представлення віртуальної сутності, наприклад людина, машина або датчик температури. Крім того, віртуальна сутність може мати нуль для багатьох різних атрибутів. EntityType класу VirtualEntity може посилатися на поняття в онтології, яка визначає, які атрибути має віртуальна сутність цього типу. Кожен атрибут має ім'я (attributeName), тип (attributeType) і одне до багатьох значень (ValueContainer). AttributeType визначає семантичний тип атрибута, наприклад, що значення представляє температуру. Він може посилатися на концепції онтології. Таким чином, можна, наприклад, змоделювати атрибут, наприклад список значень, який сам має кілька значень. Кожен ValueContainer групує одне значення або жодного до багатьох (0..1-*) одиниць інформації про метадані, що належать до даного значення. Наприклад, метадані можна використовувати для збереження значення часу або інших параметрів, таких як точність або одиниця виміру. Віртуальна сутність (VirtualEntity) також підключена до ServiceDescription за допомогою асоціації ServiceDescription-VirtualEntity. Опис сервісу описує відповідні аспекти служби, включаючи її інтерфейс. Крім того, він може містити один (або більше) описів ResourceDescription, що описують ресурс, функціональність якого розкрита службою. Опис ResourceDescription, у свою чергу, може містити інформацію про пристрій, на якому розміщений ресурс.

На рисунку 2.10 показано взаємозв'язок між концепціями моделі домену та елементами інформаційної моделі IoT. Основними концепціями моделі домену, явно представленими в системі IoT, є віртуальна сутність та сервіс. Сервіс також включає поняття ресурсу та пристрою. Оскільки віртуальна сутність є

цифровим відображенням фізичної сутності, то не може бути іншого представлення фізичної сутності в інформаційній моделі IoT.

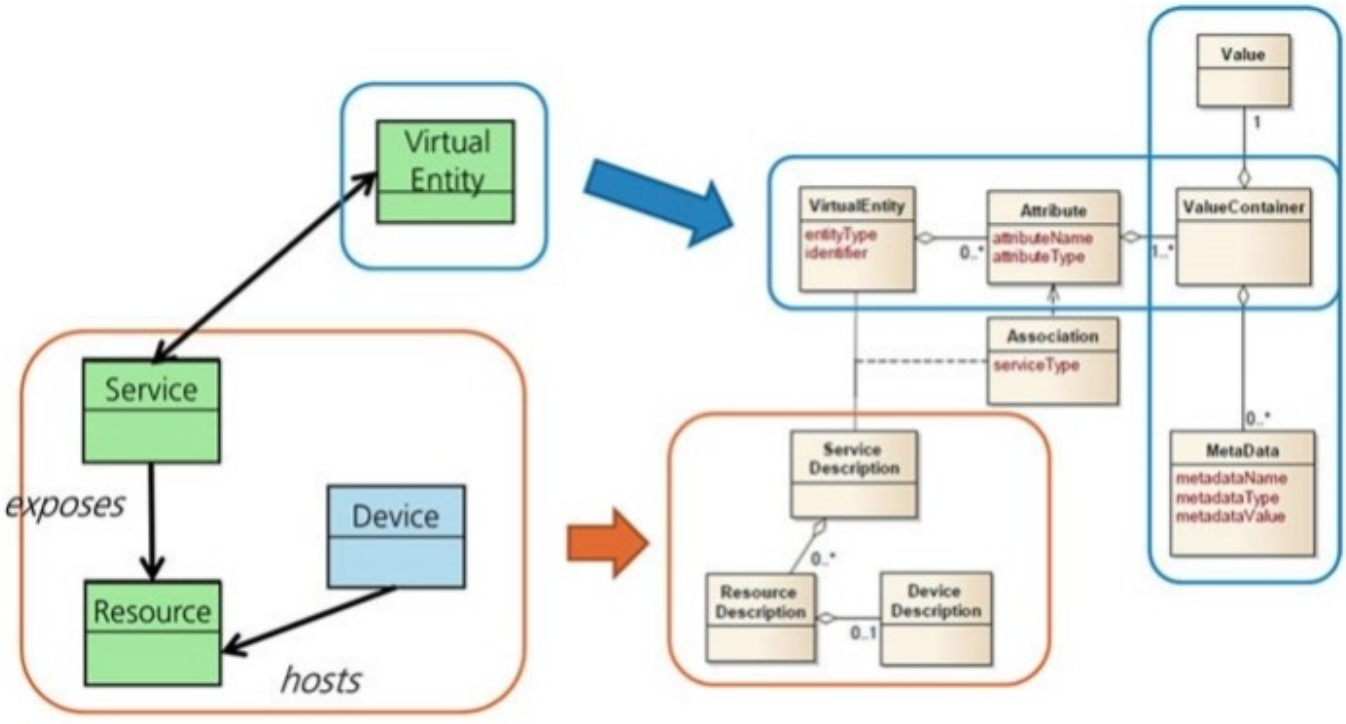

Рисунок 2.10 — Як інформаційна модель відноситься до моделі домену IoT

Інформаційна модель IoT передбачає всі концепції моделі домену, які мають бути чітко відображені та доступні для маніпуляції в цифровому світі. Крім того, інформаційна модель IoT моделює відносини між цими концепціями. Інформаційна модель IoT - це метамодель, яка забезпечує структуру інформації, яку обробляють IoT системи. Ця структура забезпечує основу для всіх аспектів системи, що стосуються представлення, збору, обробки, зберігання та пошуку інформації, і як така використовується як основа для визначення функціональних інтерфейсів системи IoT.

2.5 Розробка онтології для формалізації взаємодії інтелектуальних агентів у галузі інтернету речей

В цьому розділі на основі наведених у попередніх розділах моделей та методів буде описана онтологія для формалізації інтелектуальних агентів у галузі Інтернету речей.

При розробці онтології було взято до уваги наступні особливості пов'язані із роботою інтелектуальних агентів у контексті Інтернету речей згідно з еталонною моделлю IoT(розділ 2.4), та концепцією інтелектуальних агентів(розділ 2.1):

1 Річчю може бути будь-який фізичний об'єкт у навколишньому середовищі, у тому числі люди, тварини, тощо.

2 Кожна річ має деякі атрибути, які описують її стан, цікавлять користувача і які можна спостерігати та змінювати.

3 Для того щоб збирати інформацію про стан речі, або впливати на неї необхідно мати відповідні пристрої, які бувають трьох основних типів:

– сенсорними — для збору інформації про річ;

– тегами — для ідентифікації конкретної речі серед інших подібних речей;

– актуаторами — для впливу на стан речі;

4 Кожну річ в у цифровому світі представляє віртуальна сутність, через яку користувач або інші віртуальні сутності можуть отримувати інформацію про річ або взаємодіяти з нею. За допомогою віртуальної сутності річ стає частиною Інтернету речей.

5 В контексті мультиагентної системи віртуальною сутністю для речі буде інтелектуальний агент, який буде не тільки давати змогу збирати інформацію про річ та впливати на неї, але й надасть можливість речі самій проявляти ініціативу, взаємодіяти та кооперуватися за для виконання складних завдань з іншими речами, роблячи поведінку системи по-справжньому розумною.

6 Для того, щоб взаємодіяти з віртуальною сутністю(агентом) їй треба мати відповідний сервіс, інтерфейс, який буде надавати доступний функціонал речі.

7 Програма агенту може бути розгорнута безпосередньо на пристрої речі, або віддалено, наприклад, у хмарі та взаємодіяти з ним через мережу Інтернет.

8 Для того, щоб взаємодіяти з пристроєм речі йому треба мати відповідний сервіс, інтерфейс, який буде надавати доступний функціоналу речі.

9 Інтелектуальні агенти можуть проявляти ініціативу, демонструвати складну поведінку, можуть взаємодіяти один з одним, об'єднуватися для виконання встановлених цілей, тобто інтелектуальний агент має певний набір задач, який встановив для нього користувач.

На рисунку 2.11, зображений контекст онтології для формалізації взаємодії інтелектуальних агентів у контексті Інтернету речей.

Рисунок 2.11 ─ Контекст онтології для формалізації взаємодії інтелектуальних агентів у контексті Інтернету речей

Таким чином, онтологія для формалізації взаємодії інтелектуальних агентів у контексті Інтернету речей буде мати клас речей, від якого користувачі, що будуть використовувати онтологію, зможуть зробити конкретний підклас речей, екземплярами якого вже будуть окремі фізичні сутності. Річ буде мати набір атрибутів, які будуть її характеризувати та за якими будуть слідкувати відповідні пристрої. Клас речі буде позначатись “Object”(див. рисунок 2.12).

Було прийняте рішення винести в окремий клас людину, як окремий вид речі, тому що вона може мати спеціальні атрибути, такі як телефон, та контакти у соціальних мережах. Клас людини буде позначатись “Human”(див. рисунок 2.12).

Клас речі буде асоційований із класом інтелектуального агенту, який буде представляти кожну окрему річ у мультиагентній системі. Кожен агент може відповідати лише за одну річ, коли річ може бути асоційована із декількома агентами. Для того щоб створити конкретний тип агента, онтологію можна розширити відповідним підкласом. Агент може мати власні атрибути. Клас агента буде позначатись “Agent”(див. рисунок 2.12).

Клас речі буде асоційований із класом пристрою, який буде взаємодіяти з нею у фізичному світі. Кожен пристрій може взаємодіяти із багатьма речами, як і річ може мати багато асоційованих з нею пристроїв. Клас пристрою буде позначатись “Device”(див. рисунок 2.12). Пристрій може мати власні атрибути. Також буде наступні підкласи класу пристрою: “Sensor”, “Tag” та “Actuator”, які будуть представляти сенсори, теги та актуатори відповідно, які можуть бути розширені до конкретних марок пристроїв користувачем онтології.

Також кожен агент, як програмний ресурс має бути десь розгорнутий. Він може бути розгорнутий безпосередньо на пристрої або віддалено, наприклад, у хмарі. Для цього буде передбачений клас “Deployment”(див. рисунок 2.12), який буде мати підклас “DeviceDeployment”, що буде асоційований із класом

пристроїв. Кожний агент може посилатись лише до одного екземпляру класу “Deployment”. Кожний екземпляр класу “DeviceDeployment” може посилатись лише до одного екземпляру класу “Device”.

Класи пристроїв та агентів будуть посилатись на клас “Service”(див. рисунок 2.12), який буде представляти сервіс для взаємодії з ними. Кожен екземпляр класу агентів або пристроїв має лише один сервіс. Для того щоб описати власні типи сервісів користувач онтології може розширити її відповідним підкласом

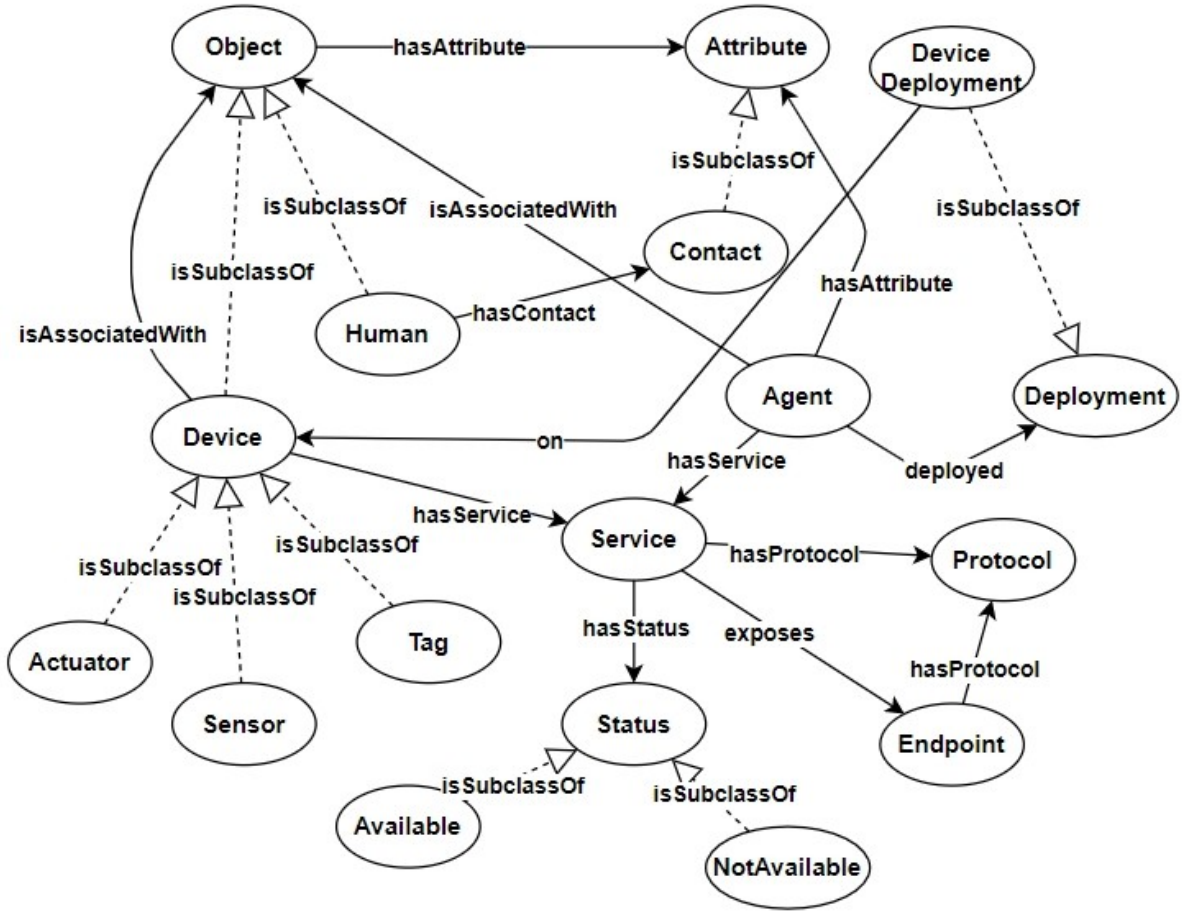

Рисунок 2.12 ─ Схема онтології частина 1

Також для роботи із сервісом необхідно знати протокол (наприклад HTTP, UDP, ZigBee, який широко на сьогоднішній день використовується у галузі Інтернету речей) за допомогою якого відбуватиметься спілкування. Для цього було створено клас “Protocol” (див. рисунок 2.12). Клас сервісу може мати безліч посилань на клас протоколу.

Також у сервісу будуть кінцеві точки, до яких будуть звертатись клієнти по заданому протоколу, і які будуть відповідні за одну конкретну послугу сервісу (наприклад, щоб отримати температуру у приміщені). До кінцевої точки можна буде звернутись по різним протоколам. Клас агента буде позначатись “Endpoint”(див. рисунок 2.12).

Так як спілкування може бути невдалим, коли, наприклад, агент звертається до пристрою, який може бути несправним або тимчасово недоступним, то треба ввести клас який буде позначати доступність агенту чи пристрою. Такий клас буде позначатись “Status” (див. рисунок 2.12). У кожного агенту чи пристрою може бути лише одне посилання на цей клас. Також у цього класу буде два підкласи “Available” та “NotAvailable”, які будуть означати доступність та недоступність пристрою чи агенту відповідно.

Агент, що буде діяти у системі Інтернету речей та представлятиме окрему річ може мати набір завдань, що назначив йому користувач, і для виконання яких він може співпрацювати з іншими агентами. Клас завдання буде позначатись “Task”(див. рисунок 2.13). Кожне завдання може мати лише одного власника в обличчі екземпляра класу агента, лише одного замовника, та безліч підписників. Клас завдання може мати власні атрибути.

Також, подібно до класу статусу доступності пристрою слід впровадити клас, який буде позначати статус виконання завдання. Такий клас буде позначатись “TaskStatus”(див. рисунок 2.13). Для кожної задачі можливий лише один екземпляр цього класу. Також він буде мати наступні підкласи: “Completed”, “Interrupted”, “InProcces”,“InQueue”, які відповідно будуть позначати що завдання виконано, припинено, виконується або чекає своєї черги.

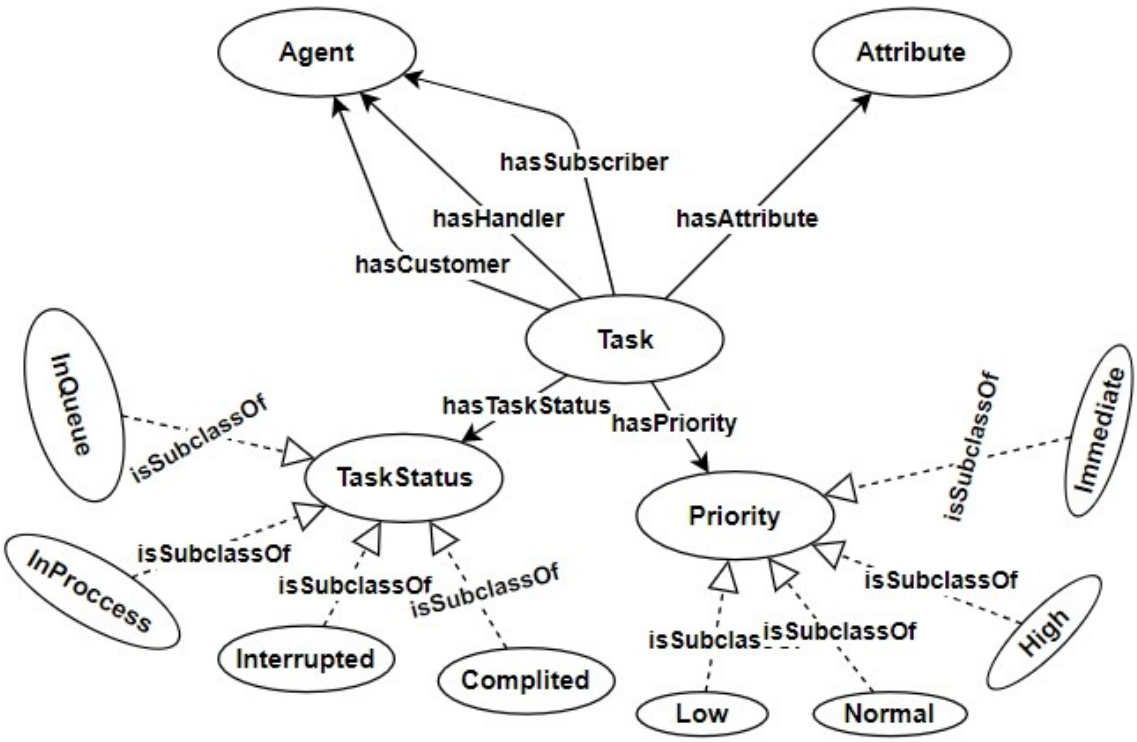

Рисунок 2.13 ─ Схема онтології частина 2

В завдання може бути пріоритет, щоб агент міг вибирати, які завдання треба виконувати негайно як, наприклад, завдання виклику бригади

пожежників, коли відповідний датчик сповістив про пожежу, а які завдання не такі критичні і їх можна відкласти у порівнянні з більш терміновими, як наприклад прибирання приміщення розумним роботом, коли господаря немає вдома. Клас пріоритету завдання буде позначатись “Priority”(див. рисунок 2.13). В кожного завдання може бути лише один екземпляр класу “Priority”. Також у класу пріоритету завдання буде декілька стандартних підкласів, які користувач онтології зможе розширити своїми. Стандартний набір підкласів буде наступним: “Low”, “Normal”, “High”, “Immediate”(див. рисунок 2.13).

Так як многоагентна система передбачає повсюдне спілкування агентів один з одним, які можуть бути об’єднанні єдиною метою або обмінюється інформацією про речі, за які вони відповідають, було прийняте рішення впровадити клас повідомлення у онтологію, так як це є важливою характеристикою мультиагентної системи. Клас повідомлення буде позначатись “Message”(див. рисунок 2.14). Він може посилатись лише на один екземпляр класу “Endpoint”. Клас повідомлення може мати власні атрибути. Повідомлення повинно мати час коли воно було відправлено або отримано. Повідомлення буде мати клас статус, що буде описувати результат повідомлення та буде позначатись “MessageStatus”(див. рисунок 2.14). Клас “MessageStatus” буде мати стандартний набір підкласів: “Send”, “Recieved”, “SendError”(див. рисунок 2.14), які будуть означати що повідомлення було відправлено, отримано або сталося помилка при відправленні відповідно. В кожного повідомлення може бути лише один екземпляр класу “MessageStatus”.

Так як повідомлення може бути з приводу якогось завдання, то клас повідомлення буде мати підклас “TaskMessage” який буде асоційований з відповідним завданням. Повідомлення може буде посилатись на безліч екземплярів класу завдання.

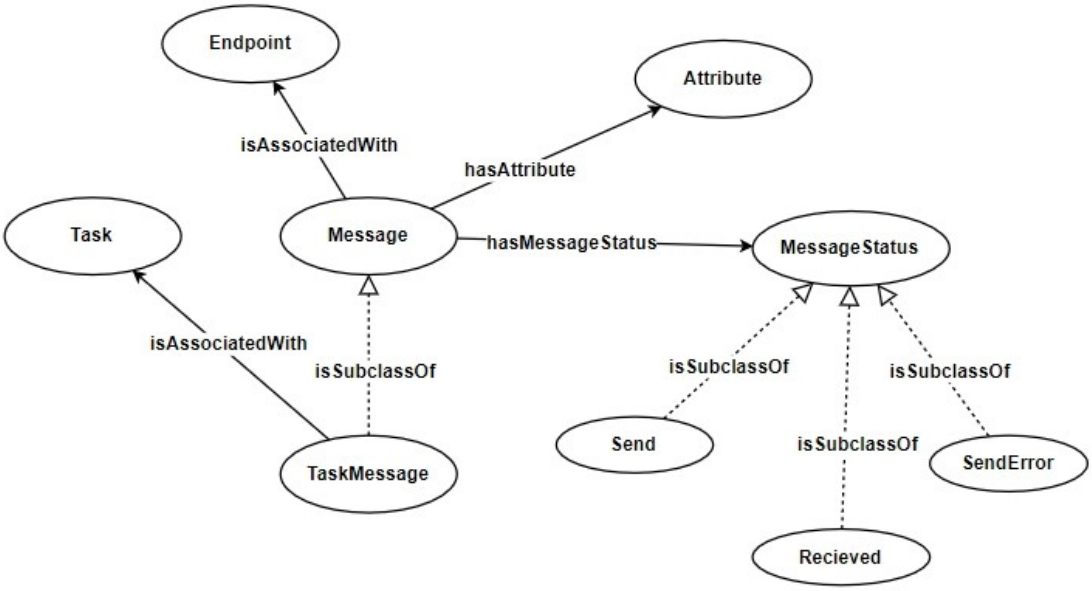

Рисунок 2.14 ─ Схема онтології частина 3

Клас атрибуту необхідний для опису характеристик екзамплярів якогось класу, які постійно змінюються та як наслідок мають набір значень замість одного. У онтології, яка розроблюється для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей, атрибути як правило будуть мати речі, та інші фізичні об’єкти наприклад, пристрої. Але так як онтологія призначена для більш широкого використання, то всі основні приведені класи онтології можуть мати атрибути, як було зазначено вище. Так як вимір атрибутів проходить не однократно, то один атрибут буде мати набір значень. Кожне значення в свою чергу буде мати набір метаданих, які можуть описувати час заміру, тип заміру, тощо. Атрибут в онтології буде представляти клас “Attribute”(див. рисунок 2.15), кожний клас може мати безліч посилань на клас атрибута. Кожний замір атрибута буде асоційований із класом “ValueContainer”, яких, як було вже зазначено, може бути декілька. Кожний екземпляр “ValueContainer” буде мати обов’язкове посилання на конкретне значення, описане в онтології класом “Value”, та необов'язкове посилання на клас “MetaData”. Клас “MetaData” може посилатись сам на себе.

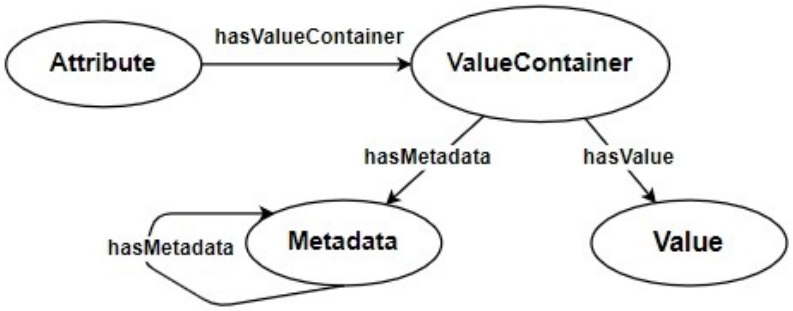

Рисунок 2.15 ─ Схема онтології частина 4

У наступних підрозділах буде детально описано онтологію для формалізації взаємодії інтелектуальних агентів у галузі Інтернету речей, її класи, об’єктні властивості, властивості даних та аксіоми. Код формалізації онтології у форматі OWL наведено у додатку А.

2.5.1 Класи онтології для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей

У таблиці 2.1 наведений детальний опис класів онтології для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей. Вона має колонки ім'я класу, ім'я батьківського класу ─ тобто класу із яким даний клас є у відношенні "isSubclassOf", кожний клас може мати лише один батьківський клас ─ та колонку опису. Також треба зазначити, що усі класи онтології будуть у підкласами загального класу "Thing".

Таблиця 2.1 ─ Детальне описання класів онтології для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей

Ім'я класу	Ім'я батьківського класу	Короткий опис
Thing	─	Загальний клас онтології, до якого всі інші класи будуть у відношенні "isSubclassOf"
Object	Thing	Клас, що представляє фізичні об'єкти, речі в онтології
Human	Object	Клас для описує людину, як фізичний об'єкт, є підкласом класу "Object"
Device	Object	Клас для опису пристрою, є підкласом класу "Object"
Sensor	Device	Клас для опису сенсорного пристрою, є підкласом класу "Device"
Tag	Device	Клас для опису пристрою типа тег, є підкласом класу "Device"
Actuator	Device	Клас для опису типа актуатор, є підкласом класу "Device"
Agent	Thing	Клас для опису віртуальної сутності речей в онтології, інтелектуальних агентів
Service	Thing	Клас для опису сервісу для агентів чи пристроїв
Protocol	Thing	Клас для опису протоколу спілкування у онтології

Продовження таблиці 2.1

Ім’я класу	Ім’я батьківського класу	Короткий опис
Endpoint	Thing	Клас для опису кінцевої точки, який відноситься до сервісу для спілкування із пристроями чи агентами
Deployment	Thing	Клас для описує розгортання агенту
DeviceDeployment	Deployment	Підклас класу “Deployment”, який описує розгортання агенту на пристрої, та має посилання на пристрій
Status	Thing	Клас, який описує статус доступності сервісу
Available	Status	Підклас класу “Status”, що позначає доступність сервісу
NotAvailable	Status	Підклас класу “Status”, що позначає недоступність сервісу
Task	Thing	Клас, що описує завдання агенту у онтології
TaskStatus	Thing	Клас, що описує поточний статус завдання агенту у онтології
Completed	TaskStatus	Підклас класу “TaskStatus”, що позначає завдання виконано
Interrupted	TaskStatus	Підклас класу “TaskStatus”, що позначає, що виконання завдання було припинене
InProccess	TaskStatus	Підклас класу “TaskStatus”, що позначає, що завдання наразі виконується
InQueue	TaskStatus	Підклас класу “TaskStatus”, що позначає, завдання наразі знаходиться у черзі на виконання
Priority	Thing	Клас, що описує пріоритет завдання агенту

Продовження таблиці 2.1

Ім’я класу	Ім’я батьківського класу	Короткий опис
Low	Priority	Підклас класу “Priority”, що позначає низький пріоритет завдання агенту
High	Priority	Підклас класу “Priority”, що позначає високий пріоритет завдання агенту
Normal	Priority	Підклас класу “Priority”, що позначає звичайний пріоритет завдання агенту
Immediate	Priority	Підклас класу “Priority”, що позначає критичний пріоритет завдання агенту
Message	Thing	Клас, що описує повідомлення між між інтелектуальними агентами
TaskMessage	Message	Клас, що описує повідомлення, що стосується одного чи декількох завдань агенту, є підкласом класу “Message”
MessageStatus	Thing	Клас, що описує статус результату повідомлення
Send	MessageStatus	Підклас класу “MessageStatus”, що позначає, що повідомлення було відправлено
Received	MessageStatus	Підклас класу “MessageStatus”, що позначає, що повідомлення було отримано
SendError	MessageStatus	Підклас класу “MessageStatus”, що позначає, що при відправленні сталося помилка
Attribute	Thing	Клас онтології, що представляє атрибут
Contact	Attribute	Підклас класу “Attribute”, що позначає контакт людини

Продовження таблиці 2.1

Ім’я класу	Ім’я батьківського класу	Короткий опис
ValueContainer	Thing	Клас для, що представляє конкретний значення атрибуту, має посилання на клас значення атрибути та на метадані значення
Metadata	Thing	Клас, що описує метадані значення атрибуту
Value	Thing	Значення атрибуту

2.5.2 Об’єктні властивості онтології для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей

У таблиці 2.2 наведений детальний опис властивостей онтології для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей. Вона має наступні колонки:

- ім’я властивості;
- домен, що описує якому класу може належати властивість;
- діапазон, що описує на які класи може вказувати властивість;
- функціональність, що позначає скільки таких властивостей може мати один екземпляр класів домену, та скільки посилань такої властивості можуть мати екземпляр класів діапазону. Значення ставляться через тире. Можливі наступні значення:
  - “0..1” ─ нуль, або один;
  - “1” ─ один;
  - “0..*” ─ від нуля та більше;
  - “1..*” ─ не менш одного;
- опис, коротке описання властивості.

Також слід відзначити, що всі властивості в онтології будуть підвласивостями загальної властивості “topObjectPropery”. Всі властивості онтології є об’єктними, тобто у їх діапазон можуть належати лише об’єкти класів, додаванням властивостей даних буде займатись користувач онтології.

Таблиця 2.2 ─ Детальне описання властивостей онтології для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей

Ім’я	Домен	Діапазон	Ф	Опис
topObjectProperty	─	─	─	Загальна властивість, підкласами якої будуть усі інші властивості онтології
hasAttribute	Object, Task, Message, Deployment, Agent	Attribute	0..* ─ 0..*	Властивість, що дозволяє екземплярам класів онтології мати атрибути. Атрибутів може бути багато або жодного.
hasValueContainer	Attribute	Value-Container	0..* ─ 1	Властивість атрибуту може мати контейнер значення разом із метаданими
hasMetadata	Value-Container, Metadata	Metadata	0..* ─ 1	Властивість контейнера значення атрибута мати метадані. Контейнер може мати багато метаданих, коли метадані можуть мати лише один контейнер
hasValue	Value-Container	Value	1 ─ 1	Властивість контейнера значення атрибута мати значення, контейнер і значення можуть мати лише одне відношення “hasValue”
isAssociatedWith-Object	Agent	Object	1 ─ 0..*	Властивість агенту бути асоційованим з річчю, один агент може бути асоційований лише з однією річчю, коли річ може мати багато агентів
isWorkingWithObject	Device	Object	0..* ─ 0..*	Властивість пристрою бути асоційованим із об’єктом, пристрій може багато об’єктів, та об’єкт може мати багато пристроїв.
hasService	Agent, Device	Service	1 ─ 0..*	Властивість пристрою та агенту мати сервіс. Можна мати лише один сервіс

Ім’я	Домен	Діапазон	Ф	Опис
hasProtocol	Service	Protocol	1..* — 0..*	Властивість сервісу та кінцевої точки сервісу мати протокол спілкування, можна мати не менше одного протоколу
hasEndpoint	Service	Endpoint	1..* — 1	Властивість сервісу мати кінцеву точку спілкування, сервіс може мати не менше одної кінцевої точки
hasWorkingStatus	Service	Status	1 — 1	Властивість сервісу мати статус роботи, сервіс може мати лише один статус роботи
hasSubscriber	Task	Agent	0..* — 0..*	Властивість завдання агента мати підписників, можна мати жодного або безліч підписників
hasCustomer	Task	Agent	1 — 0..*	Властивість завдання агента мати замовника, можна мати лише одного замовника
hasHandler	Task	Agent	1..* — 0..*	Властивість завдання агента мати поточного агента, можна мати лише одного поточного агента
hasPriority	Task	Priority	1 — 1	Властивість завдання агента мати пріоритет, можна мати лише одне посилання на екземпляр класу пріоритет
hasTaskStatus	Task	TaskStatus	1 — 1	Властивість завдання агента мати статус виконання, можна мати лише одне посилання на екземпляр класу статусу виконання
hasDeployment	Agent	Deployment	1 — 1	Властивість агента мати екземпляр класу розгортання, можна мати лише одне посилання на екземпляр класу розгортання
onDevice	Device-Deployment	Device	1 — 0..*	Властивість класу розгортання на пристрої мати посилання на клас пристрою, можливе лише одне таке посилання

Продовження таблиці 2.2

Ім’я	Домен	Діапазон	Ф	Опис
hasMessageStatus	Message	Message-Status	1 — 1	Властивість повідомленню мати статус. Повідомлення може мати один статус як і екземпляр класу статусу виконання посилатись лише на одне повідомлення
isAssociatedWith-Endpoint	Message	Endpoint	1 — 0..*	Властивість повідомлення бути асоційованим із деякою кінцевою точкою через яку йде спілкування, можливо бути асоційованим лише із однією кінцевою точкою
isAssociatedWith-Task	Task-Message	Task	1 — 0..*	Властивість повідомлення бути асоційованим із деяким завданням агента

2.6 Висновки до розділу 2

У цьому розділі ставилась мета розглянути моделі та методі для формалізації взаємодії інтелектуальних агентів у галузі Інтернету речей та розробка відповідної онтології.

Для цього була розглянута теорія інтелектуальних агентів; було проаналізоване поняття онтології та супутніх понять онтологічного інжинірингу; була розглянута програмна платформа JADE, яка використовується для написання програм інтелектуальних агентів; була проаналізована архітектура речі з точки зору еталонної моделі Інтернету речей; та розглянутий процес побудови онтології для формалізації взаємодії інтелектуальних агентів у галузі Інтернету речей на основі наведених моделей та методів.

3 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ ОНТОЛОГІЇ ДЛЯ ФОРМАЛІЗАЦІЇ ВЗАЄМОДІЇ ІНТЕЛЕКТУАЛЬНИХ АГЕНТІВ У ГАЛУЗІ ІНТЕРНЕТУ РЕЧЕЙ

Для того щоб оцінити побудовану онтологію для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей було прийняте рішення спроектувати мультиагентну систему у контексті Інтернету речей. Тобто розробити кілька агентів та на прикладі їх взаємодії показати, що розроблена онтологія придатна для потенційного використання в реальних многоагентних системах. Таким чином, доказом придатності онтології для подальшого використання будемо вважати успішну робота розроблених на її основі агентів. Для перевірки цього будемо використовувати інтеграційне тестування програм агентів.

Для тестування онтології агенти будуть побудовані у програмній платформі JADE(див. розділ 2.2). Розроблені агенти будуть взаємодіяти у рамках одного контейнера агентів.

Визначимо вимоги, яким повинні відповідати агенти, для тестування онтології:

1 Розроблена онтологія призначена для використання інтелектуальними агентами у контексті Інтернету речей, тобто майбутня система також має діяти у цьому контексті, тобто або взаємодіяти з річчю, або представляти річ у рамках концепції Інтернету речей.

2 Так як онтологія була створена у першу чергу для формалізації взаємодії агентів, то агентів повинно бути не менше двох, щоб перевірити класи онтології, що призначені для спілкування агентів.

3 Так як розроблена онтологія для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей була створена для загального використання, вона є верхньою онтологією(див. розділ 2.3) для домену Інтернету речей, отже для того щоб її використовувати агенти мають розширити її класами та екземплярами конкретної бізнес-логіки.

4 Майбутні агенти повинні передбачати використання наступних класів(див. розділ 2.5), так як вони є ключовими у онтології:

- клас фізичного об’єкту “Object”;
- клас агенту “Agent”;
- клас пристрою “Device”;
- клас сервісу “Service”;
- клас завдання агенту “Task”;
- клас повідомлення “Message”.

5 Хоча розроблена онтологія призначена для використання саме інтелектуальними агентами для її перевірки достатньо, щоб агенти діяли, як звичайні програми на основі заданих алгоритмів.

3.1 Описання роботи мультиагентної системи світу теплиці з помідорами

Для тестування онтології була обрана наступна життєва ситуація. Існує теплиця, у якій фермер вирощує помідори і для того, щоб помідори були відбірними і фермер зміг їх продати йому необхідно контролювати кількість світла, яку помідори споживають. Для контролю постачання світла фермер використовує спеціальний датчик, який розпізнає ступінь червоності помідорів та спеціальну лампу, щоб надавати необхідну кількість світла. Теплиця фермера поділена на окремі грядки і до кожної грядки прикріплений свій датчик і своя лампа і раніше постачання світла зменшувалось чи збільшувалось одразу відповідно до змін ступеня червоності помідорів, тобто процес контролю світла був автоматизованим. Так як у минулому році фермер не зміг продати усі свої помідори, то в цьому році йому доводиться економити і використовувати лише одну лампи на всі грядки теплиці. І щоб не збанкрутувати він вирішує поступити таким чином: впровадити інтелектуального агента який буде відповідати за лампу, реєструвати датчики грядок через мережу для того, щоб вони надсилали йому показники червоності помідорів у грядках і залежно від них вирішувати на яку грядку налаштовувати лампу. Завдання агенту через свій пристрій ставить фермер. Також фермер бажає щоб агент міг повідомляти його, якщо один із датчиків перестане відповідати, або якщо лампа перестане рухатись.

Таким чином, маємо наступні типи агентів у мультиагентній системі теплиці фермера:

– агент лампи, отримує завдання від фермера, реєструє датчики грядок налаштовує лампу на відповідну грядку та сповіщає фермера про несправність приладів;

– агент датчику, який регулярно надсилає показники червоності помідорів грядки агенту лампи;

Також можемо виділити наступні сценарії взаємодії агентів:

– фермер активує лампу, вона реєструє датчиків грядки, які потім відправляють їй показники червоності помідорів до тих пір поки фермер не припинить завдання (див. рисунок 3.1);

– фермер ставить завдання для агента лампи сповіщати його, якщо один із датчиків перестане відповідати до тих пір поки фермер не припинить завдання (див. рисунок 3.3);

– фермер ставить завдання для агента лампи сповіщати його, якщо один лампа перестане рухатись до тих пір поки фермер не припинить завдання (див. рисунок 3.2);

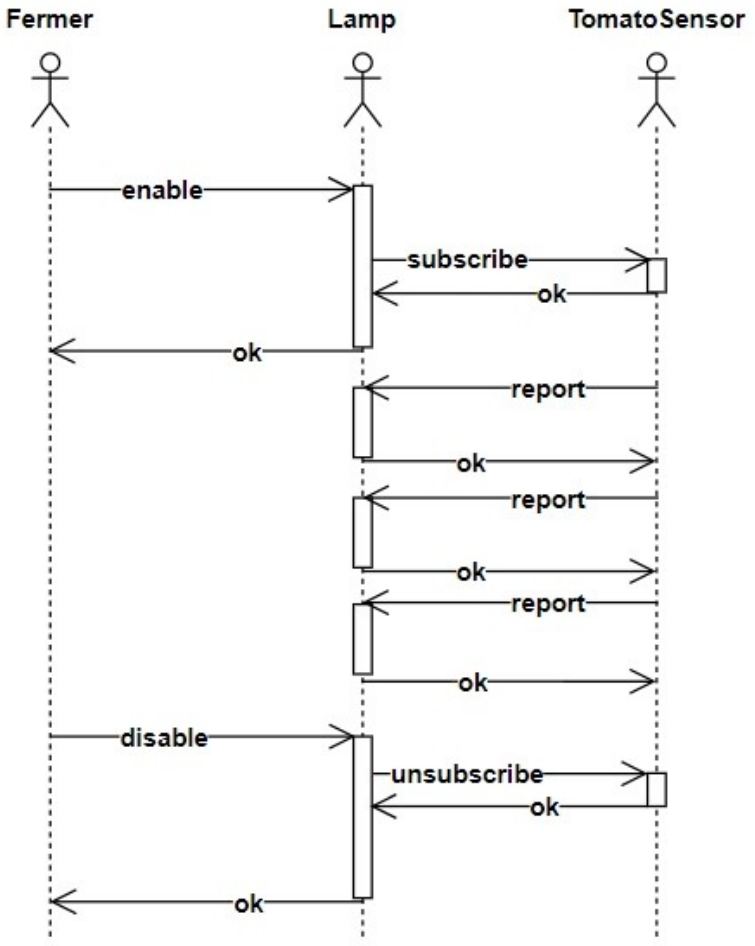

Рисунок 3.1 ─ UML діаграма взаємодії агентів у завданні слідкувати за показником червоності помідорів на грядці

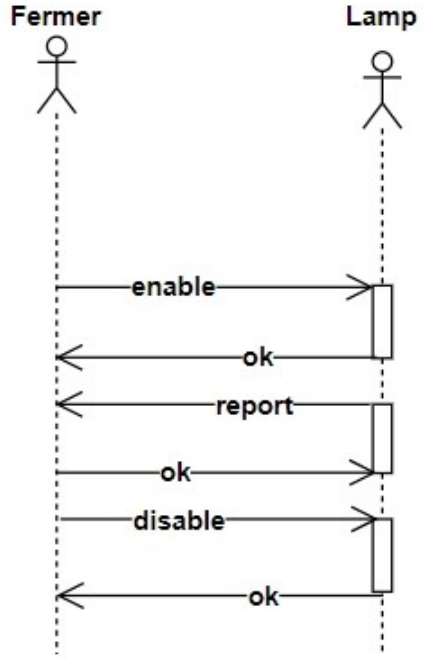

Рисунок 3.2 ─ UML діаграма взаємодії агентів у завданні повідомляти про несправність лампи

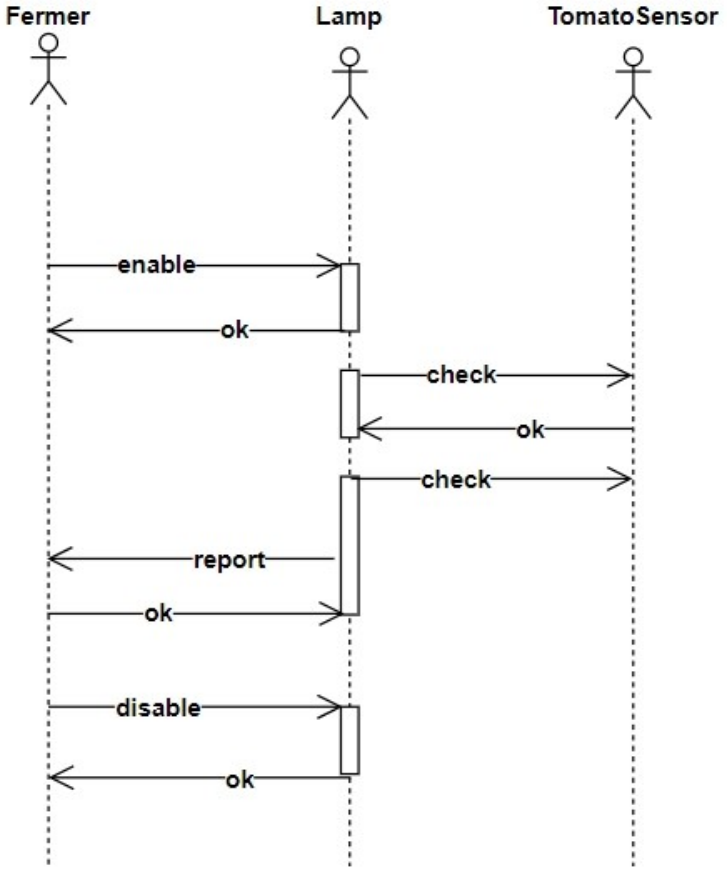

Рисунок 3.3 ─ UML діаграма взаємодії агентів у завдані повідомляти про недоступність датчиків

Слід відзначити, що розроблений світ теплиці з помідорами є досить умовним і необхідний лише для тестування онтології, у реальному випадку логіка системи для рішення наведеної проблеми буде іншої.

У наступних підрозділах буде розглянуто проектування агентів лампи та датчиків грядок теплиці. У додатку А наведений програмний код агентів на мові Java на основі програмної платформи JADE.

3.2 Проектування агенту лампи теплиці

3.2.1 Задачі агенту лампи теплиці

Агент лампи грядки, як вже було відзначено має три типи завдання, це:

1 Завдання повороту лампи на відповідну грядку, згідно показників червоності помідорів. Сценарій роботи з агентом при цьому виглядає наступним чином:

1.1 агент лампи отримує завдання повороту лампи згідно з показників червоності помідорів від фермера;

1.2 агент лампи реєструється у всіх агентів датчиків грядок теплиці для отримання відповідних показників;

1.3 агент лампи отримує відповідні показники від датчиків грядок та змінює кут повороту лампи;

1.4 агент лампи отримує повідомлення від фермера про припинення виконання завдання;

1.5 агент лампи відписується від отримання показників в агентів датчиків грядок теплиці;

2 Завдання сповіщення про несправність датчиків, сценарій роботи якого відбувається наступним чином:

2.1 агент лампи отримує завдання про сповіщення про несправність датчиків;

2.2 агент лампи регулярно опитує агентів датчиків грядок;

2.3 якщо агенти датчиків не відповідають, то агент лампи сповіщую про це фермера;

2.4 агент лампи отримує повідомлення від фермера про припинення виконання завдання.

3 Завдання сповіщення про несправність лампи, сценарій роботи якого відбувається наступним чином:

3.1 агент отримує завдання сповіщення про несправність лампи;

3.2 якщо лампа несправна, агент лампи сповіщає про це фермера;

3.3 агент отримує повідомлення про припинення завдання;

Класи агента лампи теплиці та класи завдань, які він виконує, розроблені за допомогою програмної платформи JADE(див. розділ 2.1), представлені у додатку А.

3.2.2 Розширення онтології для агенту лампи теплиці

Агент лампи у мультиагентній системі теплиці повинен розширювати розроблену онтологію власними класами для того, щоб можна було описати агентів датчиків грядки та задачі. У таблиці 3.1 наведені нові класи онтології агенту лампи.

Агент лампи повинен мати свої версії підкласів класу “Task” для того щоб мати змогу розуміти, що від нього вимагає фермер. Так як завдання буде три зробимо три підкласи “TomatoTask”, “SensorTask” та “LampTask”, які будуть відповідно позначати завдання повороту лампи згідно з показниками червоності, завдання сповіщення про несправність датчиків, та завдання сповіщення про несправність лампи.

Потім в онтології необхідно передбачити клас для агентів датчиків грядок, клас їхнього сервісу, та кінцевих точок. Фермер у світі теплиці теж буде представлений агентом, що буде ініціювати та припиняти роботу завдань для агенту лампи, тобто для нього також необхідно передбачити клас, клас для його сервісу, та кінцевих точок, яких буде дві “FermerLampAlertEndpoint” та “FermerSensorsAlertEndpoint”, які будуть призначені для сповіщення про несправність лампи та датчиків грядки відповідно.

Із агентом фермера агент лампи без спілкуватись по протоколу HTTP, коли з агентами датчиків ─ по протоколу ZigBee, для чого в онтології були передбачені відповідні класи.

Класи онтології, розробленої у попередніх розділах для формалізації взаємодії інтелектуальних агентів в контексті Інтернету речей та її розширеної версії для агента лампи, розроблених за допомогою програмної платформи JADE представлені у додатку Б. Вони дозволяють працювати з класами онтології як з класами мови Java, а екземплярами онтології як з їх об’єктами подібно до Object Relational Model.

Таблиця 3.1 — Описання підкласів класів розробленої онтології для світу агенту лампи теплиці

Ім’я класу	Батьківський клас	Опис
TomatoTask	Task	Представляє завдання налаштування постачання світла на грядки
SensorTask	Task	Представляє завдання перевірки роботи датчиків, та сповіщення про несправність
LampTask	Task	Представляє сповіщення про несправність лампи
TomatoAgent	Agent	Клас агенту, який працює із датчиком показників червоності помідора
TomatoAgentService	Service	Клас сервісу агенту, який працює із датчиком показників червоності помідора
TomatoAgentEndpoint	Endpoint	Кінцева точка сервісу агенту, через яку агент лампи буде відправляти повідомлення агенту
ZigBeeProtocol	Protocol	Протокол ZigBee, який буде використовуватись для взаємодії агентів у межах теплиці
HttpProtocol	Protocol	Протокол Http, який буде використовуватись для взаємодії агенту лампи та фермера
TomatoDevice	Sensor	Пристрій датчику показників червоності помідора
FermerAgent	Agent	У світі агента лампи фермер також буде представлений агентом
FermerAgentService	Service	Клас сервісу агенту, який представляє фермера
FermerLamp-AlertEndpoint	Endpoint	Кінцева точка, сервісу агента фермера для повідомлення про несправність лампи
FermerSensors - AlertEndpoint	Endpoint	Кінцева точка, сервісу агента фермера для повідомлення про несправність датчиків

3.3 Проектування агенту датчику грядки

3.3.1 Задачі агенту датчику грядки

Агент датчику грядки має лише одну задачу, це задача виміру показників червоності відповідної грядки, сценарій роботи якої виглядає наступним чином:

1 агент датчику отримує повідомлення про завдання заміру червоності помідорів грядки від фермера;

2 агент отримує повідомлення від агенту лампи про підписку на отримання показників червоності грядки;

3 агент відправляє повідомлення усім підписникам про показники червоності;

4 агент отримує повідомлення про припинення підписки на отримання показників червоності грядки;

5 агент отримує повідомлення про відміну завдання.

Класи агента датчику грядки та класи завдань, які він виконує, розроблені за допомогою програмної платформа JADE(див. розділ 2.1) представлені у додатку А.

3.3.2 Розширення онтології для агенту датчику грядки

Агент датчику виміру ступеню червоності помідора у мультиагентній системі теплиці повинен розширювати розроблену онтологію власними класами для того, щоб можна було описати інших агентів та задачі. У таблиці 3.2 наведені нові класи онтології агенту лампи.

Агент датчику грядки повинен мати підкласи класу “Task” для того щоб мати змогу розуміти, що від нього вимагає агент лампи, також мати уявлення про самого агента лампи, його сервіс, протоколи через які він працює та відповідні кінцеві точки.

Класи онтології, розробленої у попередніх розділах для формалізації взаємодії інтелектуальних агентів в контексті Інтернету речей та її розширеної версії для агента датчику грядки, розроблених за допомогою програмної платформи JADE представлені у додатку Б.

Таблиця 3.2 — Описання підкласів класів розробленої онтології для світу агенту лампи теплиці

Ім’я класу	Батьківський клас	Опис
ObserveTask	Task	Представляє завдання виміру ступеню червоності помідора
LampAgent	Agent	Клас агенту, який лампи, яка підписалась на завдання виміру ступеню червоності помідорів грядки
LampAgentService	Service	Клас сервісу агенту лампи для відправлення показників ступеню червоності помідорів грядки
Tomato	Object	Клас помідорів грядки
Redness	Attribute	Атрибут показнику червоності помідорів грядки, який може змінюватись з часом
LampAgent-Endpoint	Endpoint	Кінцева точка сервісу агенту лампи, через яку агент датчику буде відправляти повідомлення агенту лампи
ObserveMessage	TaskMessage	Повідомлення про завдання виміру показників червоності грядки
ReportMessage	Message	Повідомлення показнику червоності підписникам
AliveMessage	TaskMessage	Повідомлення про справність

3.4 Тестування взаємодії агентів системи теплиці розроблених на основі представленої онтології

У програмній платформі JADE кожен агент працює як окрема програма на своєму хості, яку можна сприймати як окремий екземпляр JVM, хоча насправді вони мають єдину платформу для обміну повідомленнями та керуванням запуску та припиненням роботи агентів платформи (див. розділ 2.1). Для тестування взаємодії агентів світу теплиці з помідорами візьмемо:

- одного агента фермера для ініціювання та припинення завдань, через нього будуть ставитись завдання агенту лампи та агентам датчиків грядок, який буде позначатись “F”;
- одного агента лампи, для повороту лампи та спілкування з агентами датчиків грядок, який буде позначатись “L”;
- трьох агентів датчиків грядок, які будуть відповідати за три грядки у теплиці, які будуть позначатись “S1”, “S2”, “S3”;

В ході тестування ми будемо взаємодіяти з програмою агентом фермера, для цього в нас буде два поля:

- поле “task”, яке буде позначати яке завдання ми наразі хочемо ініціювати або припинити. У нього буде три значення відповідно до трьох типів завдань системи теплиці:
  - “tomato” ― для завдання повороту лампи згідно показників червоності помідорів у грядках;
  - “lamp” ― для завдання сповіщення агенту фермера, коли лампа перестане рухатись;
  - “tomato” ― для завдання сповіщення агенту фермера, коли датчики перестануть відповідати;
- поле “action”, яке буде позначати дію, яку ми хочемо зробити. У нього буде лише два значення, виходячи з простоти представленого світу теплиці:
  - “start” для ініціювання завдання;
  - “stop” для припинення завдання;

Для того, щоб імітувати роботу датчиків, які вимірюють ступінь червоності помідорів, кожна програма агентів датчиків матиме поле redness для позначення показника червоності. Це поле прийматиме значення від одного до п’яти. Також для ініціювання випадків поломки датчиків грядки у цього поля буде два спеціальних значення:

- “down” ― для імітування поломки приладу;
- “up” ― для відновлення роботи приладу;

Для ініціювання випадків поломки лампи програма агенту лампи матиме поле “lamp” в якого також буде значення “down” та “up”.

Нижче приведена таблиця 3.3 тестування взаємодії мультиагентної системи світу теплиці. Вона складається з наступних полів:

- “№” ― для позначення номеру тесту;
- “Умови”, яке буде містити інформацію про статус завдань на момент звернення до агенту;

- “A” ─ для позначення агенту, до якого буде йти звернення
- “Вхідні дані” ─ значення полів агенту;
- “F” ─ логи агенту фермера;
- “L” ─ логи агенту лампи;
- “S1” ─ логи агенту датчику грядки 1;
- “S2” ─ логи агенту датчику грядки 2;
- “S3” ─ логи агенту датчику грядки 3;
- “П” ─ для позначення успішності тесту.

На момент проходження тестів усі програми агентів запущені, зображення роботи програм агентів приведені у додатку В.

Таблиця 3.3 ─ Тестування взаємодії агентів світу теплиці з помідорами

№	Умови	A	Вхідні дані	F	L	S1	S2	S3	П
1	Tomato:- Lamp: - Sensors: -	F	task: tomato action: start	starting tomato task	starting tomato task subscribing to S1 subscribing to S2 subscribing to S1	L is subscribed	L is subscribed	L is subscribed	+
2	Tomato:+ Lamp: - Sensors: -	S1	redness: 4	─	S1, redness: 4 current position: - current redness: - rotating lamp to S1	redness: 4 sending to L	─	─	+
3	Tomato:+ Lamp: - Sensors: -	S3	redness: 2	─	S3, redness: 2 current position: S1 current redness: 4 rotating lamp to S3	─	─	redness: 2 sending to L	+
4	Tomato:+ Lamp: - Sensors: -	S2	redness: 3	─	S2, redness: 3 current position: S3 current redness: 2	─	redness: 3 sending to L	─	+
5	Tomato:+ Lamp: - Sensors: -	F	task: tomato action: start	task tomato is already started	─	─	─	─	+
6	Tomato:- Lamp: - Sensors: -	F	task: tomato action: stop	interrupting tomato task	interrupting tomato task unsubscribing of S1 unsubscribing of S2 unsubscribing of S3	L is unsubscribed	L is unsubscribed	L is unsubscribed	+
7	Tomato:- Lamp: - Sensors: -	F	task: tomato action: stop	task tomato is already stopped	─	─	─	─	+

Продовження таблиці 3.3

№	Умови	A	Вхідні дані	F	L	S1	S2	S3	П
8	Tomato:- Lamp: - Sensors: -	F	task: lamp action: start	starting lamp task	starting lamp task	–	–	–	+
9	Tomato:- Lamp: + Sensors: -	L	action: down	L is down	action: down	–	–	–	+
10	Tomato:- Lamp: + Sensors: -	L	action: up	–	action: up	–	–	–	+
11	Tomato:- Lamp: + Sensors: -	F	task: lamp action: stop	interrupting lamp task	interrupting lamp task	–	–	–	+
12	Tomato:- Lamp: - Sensors: -	F	task: sensors action: start	starting sensors task	starting sensors task checking S1 checking S2 checking S3	I am OK	I am OK	I am OK	+
13	Tomato:- Lamp: - Sensors: +	S3	redness: down	S3 is down	checking S1 checking S2 S3 is down	I am OK	I am OK	I am down	+
14	Tomato:- Lamp: - Sensors: +	F	task: sensors action: stop	interrupting sensors task	interrupting sensors task	–	–	–	+
15	Tomato:- Lamp: - Sensors: -	S2	redness: down	–	–	–	–	I am down	+

3.5 Висновки до розділу 3

У цьому розділі ставилась задача перевірити представлену у розділі 2 онтологію для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей, щоб її в подальшому можна було використовувати при проектуванні реальних багатоагентних систем у галузі Інтернету речей. Для цього було прийняте рішення розробити прототипи інтелектуальних агентів з використанням розробленої онтології та на прикладі їх взаємодії довести, що онтологія придатна для використання.

Для цього було сформульовано набір вимог до майбутньої мультиагентної системи; було створено умовний світ теплиці з помідорами, в рамках якого взаємодіють декілька агентів; розроблені алгоритми роботи агентів; для кожного агенту було розширено представлену онтологію власними сутностями та класами, які описують її світ. Також були написані програми агентів за допомогою програмної платформи JADE. Після чого взаємодія агентів була протестована.

Таким чином, можна зробити висновок, що розроблена онтологія для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей потенційно може використовуватись у реальних проектах.

ВИСНОВКИ

В ході виконання дипломного проекту на тему “Формалізація взаємодії інтелектуальних агентів у галузі Інтернету речей за допомогою онтологічного інжинірингу” були пройдені наступні етапи.

Сформульовані мета та завдання дипломного проекту; сформульовані об’єкт та предмет дослідження; та доведена його актуальність і практична значимість.

В першому розділі були розглянуті основні принципи і архітектура Інтернету речей, розвиток і перспективи, а також головні проблеми; розглянуті поняття інтелектуальних агентів, класифікація, переваги та використання їх в області Інтернету речей; доведена актуальність формалізації взаємодії інтелектуальних агентів в області Інтернету речей.

У другому розділі була розглянута теорія роботи інтелектуальних агентів; детально розглянуте поняття онтології, і теорія побудови онтології; був описаний процес розробки онтології для формалізації взаємодії інтелектуальних агентів в області Інтернету речей.

У третьому розділі були розроблені прототипи агентів, на прикладі взаємодії яких було доведено можливість застосування розробленої онтології для проектування реальних систем інтелектуальних агентів в галузі Інтернету речей.

Таким чином, мету дипломного проекту можна вважати виконаною: онтологія для формалізації взаємодії інтелектуальних агенті в у галузі Інтернету речей розроблена, та успішно протестована та може бути використана для розробки реальних мультиагентних систем в контексті Інтернету речей.

ПЕРЕЛІК ПОСИЛАНЬ

1 Что такое IoT, или интернет вещей [Электронный ресурс] Режим доступа: https://coinspot.io › beginners › chto-takoe-iot-ili-internet-veshhej

2 Многоагентные системы [Электронный ресурс]/ Режим доступа: http://www.aiportal.ru/articles/multiagent-systems/multiagent-systems.html

3 Интеллектуальный агент [Электронный ресурс]/ Режим доступа: https://ru.wikipedia.org/wiki/Интеллектуальный_агент

4 Онтологический инжиниринг в информационной науке [Электронный ресурс]/ Режим доступа: https://cyberleninka.ru/article/n/ontologicheskiy-inzhiniring-v-informatsionnoy-nauke-zarubezhnyy-opyt

5 Искусственный интеллект (ИИ, Artificial intelligence, AI) [Электронный ресурс]/Режим доступа: http://www.tadviser.ru/index.php/Продукт:Искусственный_интеллект_(ИИ,_Artificial_intelligence,_AI)

6 Интернет вещей: сетевая архитектура и архитектура безопасности [Электронный ресурс]/ Режим доступа: http://internetinside.ru/internet-veshhey-setevaya-arkhitektura-i/

7 Интеллектуальные агенты [Электронный ресурс]/ Режим доступа: https://helpiks.org/6-88718.html

8 Интеллектуальные агенты [Электронный ресурс]/ Режим доступа: https://ru.wikipedia.org/wiki/Интеллектуальный_агент

9 Городецкий В.И., Скобелев П.О. Многоагентные технологии для индустриальных приложений: реальность и перспектива. /Навчальний посібник

10 В.А. Лапшин Онтологии в информационных системах: Современный подход /Учебное пособие. 2010

11 Онтологии и тезаурусы: модели, инструменты, приложения НОУ «ИНТУИТ» [Электронный ресурс]/ Режим доступа: https://www.intuit.ru/studies/courses/1078/270/info

12 Городецкий В.И., Скобелев П.О. Многоагентные технологии для индустриальных приложений: реальность и перспектива. /Навчальний посібник

13 Рассел, Стюарт, Норвиг, Искусственный интеллект: современный подход/Навчальний посібник 2007.

14 Логика первого порядка [Электронный ресурс]/ Режим доступа: https://math.wikia.org/ru/wiki/Логика_первого_порядка.

15 Логика предикатов первого порядка [Электронный ресурс]/ Режим доступа: https://sites.google.com/site/anisimovkhv/learning/iis/lecture/tema9
16 Логіка першого порядку – [Электронный ресурс]/ Режим доступа: https://uk.wikipedia.org/wiki/Логіка_першого_порядку
17 An introduction to multiagent systems. Michael Wooldridge. /Навчальний посібник 2002
18 Constructive dialogue modelling: speech interaction and rational agents/Kristiina Jokinen / Навчальний посібник 2009
19 Developing multi-agent systems with JADE / Fabio Bellifemine, Giovanni Caire, Dominic Greenwood./ Навчальний посібник 2004
20 Introduction to JADE [Электронный ресурс]/ Режим доступа]: https://jade.tilab.com/documentation/tutorials-guides/introduction-to-jade/
21 JADE Programming tutorial [Электронный ресурс]/ Режим доступа]:https://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf
22 JADE programmers guide: [Электронный ресурс]/ Режим доступа]: https://jade.tilab.com/doc/programmersguide.pdf
23 Ontology-Based Multi-Agent Systems /Maja Hadzic, Pornpit Wongthongtham, Tharam Dillon, and Elizabeth Chang / Навчальний посібник 2011
24 Building Ontologies with Basic Formal Ontology / Robert Arp, Barry Smith, and Andrew D. Spear / Навчальний посібник 2015
25 Ontologies for the Internet of Things[Электронный ресурс]/ Режим доступа]:https://www.researchgate.net/publication/254004296_Ontologies_for_the_Internet_of_Things
26 A Survey on Ontology Evaluation Methods/Tharam Dillon, and Elizabeth Chang [Электронный ресурс]/ Режим доступа]:
27 Building Ontologies with Basic Formal Ontology / Robert Arp, Barry Smith, and Andrew D. Spear / Навчальний посібник 2015
28 Internet of Things – Architecture IoT-A Deliverable D1.5 – Final architectural reference model for the IoT v3.0 [Электронный ресурс]/ Режим доступа]:
https://www.researchgate.net/publication/299135606_A_Survey_on_Ontology_Evaluation_Methods
https://www.researchgate.net/publication/272814818_Internet_of_Things_-_Architecture_IoT-A_Deliverable_D15_-_Final_architectural_reference_model_for_the_IoT_v30
29 Internet of things (IoT) : technologies, applications, challenges, and solutions /B.K. Tripathy, J. Anuradha. / Навчальний посібник 2018

30 Enabling Things to Talk Designing IoT solutions with the IoT Architectural Reference Model/ Alessandro Bassi, Martin Bauer / Навчальний посібник 2015

ДОДАТОК А

Онтологія призначена для формалізації взаємодії інтелектуальних агентів в галузі Інтернету речей

Онтологія представлена у форматі OWL.

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.semanticweb.org/denys_levashov/iot#"
     xml:base="http://www.semanticweb.org/denys_levashov/iot"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:xml="http://www.w3.org/XML/1998/namespace"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
    <owl:Ontology rdf:about="http://www.semanticweb.org/denys_levashov/iot"/>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://www.semanticweb.org/denys_levashov/iot#hasAttribute -->

    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasAttribute">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Agent"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Deployment"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Device"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Human"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Message"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Object"/>
        <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Task"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Attribute"/>
    </owl:ObjectProperty>



    <!-- http://www.semanticweb.org/denys_levashov/iot#hasCustomer -->

    <owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasCustomer">
        <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Task"/>
        <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Agent"/>
    </owl:ObjectProperty>
```

```
<!-- http://www.semanticweb.org/denys_levashov/iot#hasDeployment -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasDeployment">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Agent"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Deployment"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasMessageStatus -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasMessageStatus">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Message"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#MessageStatus"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasMetadata -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasMetadata">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#MetaData"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#ValueContainer"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#MetaData"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasPriority -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasPriority">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Task"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskPriority"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasProtocol -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasProtocol">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Service"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#ServiceEndpoint"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Protocol"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasService -->
```

```
<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasService">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Agent"/>
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Device"/>
   <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Service"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasSubscriber -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasSubscriber">
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Task"/>
   <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Agent"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasTaskStatus -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasTaskStatus">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Task"/>
   <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskStatus"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasValue -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasValue">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#ValueContainer"/>
   <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Value"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasValueContainer -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasValueContainer">
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Attribute"/>
   <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#ValueContainer"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasWorkingStatus -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasWorkingStatus">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Agent"/>
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Device"/>
```

```
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#WorkingStatus"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#isAccosiatedWithObject -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#isAccosiatedWithObject">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Agent"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Device"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Human"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Object"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#isAssociatedWithEndpoint -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#isAssociatedWithEndpoint">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Message"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#ServiceEndpoint"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#isAssociatedWithTask -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#isAssociatedWithTask">
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskMessage"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Task"/>
</owl:ObjectProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#onDevice -->

<owl:ObjectProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#onDevice">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#DeviceDeployment"/>
    <rdfs:range rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Device"/>
</owl:ObjectProperty>

<!--
///////////////////////////////////////////////////////////////////////////////////
//
// Data properties
//
///////////////////////////////////////////////////////////////////////////////////
 -->

 <!-- http://www.semanticweb.org/denys_levashov/iot#hasEndTime -->
```

```
<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasEndTime">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>


   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Task"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTimeStamp"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasMessageTime -->

<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasMessageTime">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Message"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTimeStamp"/>
</owl:DatatypeProperty>



<!-- http://www.semanticweb.org/denys_levashov/iot#hasStartTime -->

<owl:DatatypeProperty rdf:about="http://www.semanticweb.org/denys_levashov/iot#hasStartTime">
   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   <rdfs:domain rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Task"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTimeStamp"/>
</owl:DatatypeProperty>



<!--
///////////////////////////////////////////////////////////////////////////////////
//
// Classes
//
///////////////////////////////////////////////////////////////////////////////////
 -->




<!-- http://www.semanticweb.org/denys_levashov/iot#Actuator -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Actuator">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Device"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#Agent -->
```

```
<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Agent"/>


<!-- http://www.semanticweb.org/denys_levashov/iot#Attribute -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Attribute"/>


<!-- http://www.semanticweb.org/denys_levashov/iot#Available -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Available">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#WorkingStatus"/>
</owl:Class>


<!-- http://www.semanticweb.org/denys_levashov/iot#Complited -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Complited">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskStatus"/>
</owl:Class>


<!-- http://www.semanticweb.org/denys_levashov/iot#Contact -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Contact">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Attribute"/>
</owl:Class>


<!-- http://www.semanticweb.org/denys_levashov/iot#Deployment -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Deployment"/>


<!-- http://www.semanticweb.org/denys_levashov/iot#Device -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Device"/>


<!-- http://www.semanticweb.org/denys_levashov/iot#DeviceDeployment -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#DeviceDeployment">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Deployment"/>
</owl:Class>
```

```
<!-- http://www.semanticweb.org/denys_levashov/iot#High -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#High">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskPriority"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#Human -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Human"/>



<!-- http://www.semanticweb.org/denys_levashov/iot#Immediate -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Immediate">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskPriority"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#InProccess -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#InProccess">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskStatus"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#InQueue -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#InQueue">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskStatus"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#Interrupted -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Interrupted">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskStatus"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#Low -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Low">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskPriority"/>
</owl:Class>
```

```
<!-- http://www.semanticweb.org/denys_levashov/iot#Message -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Message"/>



<!-- http://www.semanticweb.org/denys_levashov/iot#MessageStatus -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#MessageStatus"/>



<!-- http://www.semanticweb.org/denys_levashov/iot#MetaData -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#MetaData"/>



<!-- http://www.semanticweb.org/denys_levashov/iot#Normal -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Normal">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#TaskPriority"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#NotAvailable -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#NotAvailable">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#WorkingStatus"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#Object -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Object"/>



<!-- http://www.semanticweb.org/denys_levashov/iot#Protocol -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Protocol"/>



<!-- http://www.semanticweb.org/denys_levashov/iot#Recieved -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Recieved">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#MessageStatus"/>
</owl:Class>
```

```
<!-- http://www.semanticweb.org/denys_levashov/iot#Send -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Send">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#MessageStatus"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#SendError -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#SendError">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#MessageStatus"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#Sensor -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Sensor">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Device"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#Service -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Service"/>



<!-- http://www.semanticweb.org/denys_levashov/iot#ServiceEndpoint -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#ServiceEndpoint"/>



<!-- http://www.semanticweb.org/denys_levashov/iot#Tag -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Tag">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Device"/>
</owl:Class>



<!-- http://www.semanticweb.org/denys_levashov/iot#Task -->

<owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Task"/>
```

```
    <!-- http://www.semanticweb.org/denys_levashov/iot#TaskMessage -->

    <owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#TaskMessage">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/denys_levashov/iot#Message"/>
    </owl:Class>

   <!-- http://www.semanticweb.org/denys_levashov/iot#TaskPriority -->

    <owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#TaskPriority"/>


    <!-- http://www.semanticweb.org/denys_levashov/iot#TaskStatus -->

    <owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#TaskStatus"/>


    <!-- http://www.semanticweb.org/denys_levashov/iot#Value -->

    <owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#Value"/>



    <!-- http://www.semanticweb.org/denys_levashov/iot#ValueContainer -->

    <owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#ValueContainer"/>


    <!-- http://www.semanticweb.org/denys_levashov/iot#WorkingStatus -->

    <owl:Class rdf:about="http://www.semanticweb.org/denys_levashov/iot#WorkingStatus"/>
</rdf:RDF>



<!-- Generated by the OWL API (version 4.5.9.2019-02-01T07:24:44Z) https://github.com/owlcs/owlapi -->
```

ДОДАТОК Б

Програмний код прототипів агентів мультиагентної системи теплиці з помідорами на основі програмної платформи JADE

Нижче представлені декілка класів які було згенеровано на основі онтології у програмній платформі Protege.

Agent.java

```
package pack1;

import java.net.URI;
import java.util.Collection;
import javax.xml.datatype.XMLGregorianCalendar;

import org.protege.owl.codegeneration.WrappedIndividual;

import org.semanticweb.owlapi.model.OWLNamedIndividual;
import org.semanticweb.owlapi.model.OWLOntology;

/**
 *
 * <p>
 * Generated by Protege (http://protege.stanford.edu). <br>
 * Source Class: Agent <br>
 * @version generated on Mon Dec 14 22:29:45 EET 2020 by Mykyta_Levashov
 */

public interface Agent extends WrappedIndividual {

    /* ***************************************************
     * Property http://www.semanticweb.org/denys_levashov/iot#hasAttribute
     */

    /**
     * Gets all property values for the hasAttribute property.<p>
     *
     * @returns a collection of values for the hasAttribute property.
     */
    Collection<? extends Attribute> getHasAttribute();

    /**
     * Checks if the class has a hasAttribute property value.<p>
     *
     * @return true if there is a hasAttribute property value.
     */
    boolean hasHasAttribute();
```

```
/**
 * Adds a hasAttribute property value.<p>
 *
 * @param newHasAttribute the hasAttribute property value to be added
 */
void addHasAttribute(Attribute newHasAttribute);

/**
 * Removes a hasAttribute property value.<p>
 *
 * @param oldHasAttribute the hasAttribute property value to be removed.
 */
void removeHasAttribute(Attribute oldHasAttribute);


/* ************************************************
 * Property http://www.semanticweb.org/denys_levashov/iot#hasDeployment
 */

/**
 * Gets all property values for the hasDeployment property.<p>
 *
 * @returns a collection of values for the hasDeployment property.
 */
Collection<? extends Deployment> getHasDeployment();

/**
 * Checks if the class has a hasDeployment property value.<p>
 *
 * @return true if there is a hasDeployment property value.
 */
boolean hasHasDeployment();

/**
 * Adds a hasDeployment property value.<p>
 *
 * @param newHasDeployment the hasDeployment property value to be added
 */
void addHasDeployment(Deployment newHasDeployment);

/**
 * Removes a hasDeployment property value.<p>
 *
 * @param oldHasDeployment the hasDeployment property value to be removed.
 */
void removeHasDeployment(Deployment oldHasDeployment);


/* ************************************************
 * Property http://www.semanticweb.org/denys_levashov/iot#hasService
 */
```

```
/**
 * Gets all property values for the hasService property.<p>
 *
 * @returns a collection of values for the hasService property.
 */
Collection<? extends Service> getHasService();

/**
 * Checks if the class has a hasService property value.<p>
 *
 * @return true if there is a hasService property value.
 */
boolean hasHasService();

/**
 * Adds a hasService property value.<p>
 *
 * @param newHasService the hasService property value to be added
 */
void addHasService(Service newHasService);

/**
 * Removes a hasService property value.<p>
 *
 * @param oldHasService the hasService property value to be removed.
 */
void removeHasService(Service oldHasService);


/* ************************************************
 * Property http://www.semanticweb.org/denys_levashov/iot#hasWorkingStatus
 */

/**
 * Gets all property values for the hasWorkingStatus property.<p>
 *
 * @returns a collection of values for the hasWorkingStatus property.
 */
Collection<? extends WorkingStatus> getHasWorkingStatus();

/**
 * Checks if the class has a hasWorkingStatus property value.<p>
 *
 * @return true if there is a hasWorkingStatus property value.
 */
boolean hasHasWorkingStatus();

/**
 * Adds a hasWorkingStatus property value.<p>
 *
 * @param newHasWorkingStatus the hasWorkingStatus property value to be added
 */
void addHasWorkingStatus(WorkingStatus newHasWorkingStatus);
```

```
    /**
     * Removes a hasWorkingStatus property value.<p>
     *
     * @param oldHasWorkingStatus the hasWorkingStatus property value to be removed.
     */
    void removeHasWorkingStatus(WorkingStatus oldHasWorkingStatus);


    /* ************************************************
     * Property http://www.semanticweb.org/denys_levashov/iot#isAccosiatedWithObject
     */

    /**
     * Gets all property values for the isAccosiatedWithObject property.<p>
     *
     * @returns a collection of values for the isAccosiatedWithObject property.
     */
    Collection<? extends Object> getIsAccosiatedWithObject();

    /**
     * Checks if the class has a isAccosiatedWithObject property value.<p>
     *
     * @return true if there is a isAccosiatedWithObject property value.
     */
    boolean hasIsAccosiatedWithObject();

    /**
     * Adds a isAccosiatedWithObject property value.<p>
     *
     * @param newIsAccosiatedWithObject the isAccosiatedWithObject property value to be added
     */
    void addIsAccosiatedWithObject(Object newIsAccosiatedWithObject);

    /**
     * Removes a isAccosiatedWithObject property value.<p>
     *
     * @param oldIsAccosiatedWithObject the isAccosiatedWithObject property value to be removed.
     */
    void removeIsAccosiatedWithObject(Object oldIsAccosiatedWithObject);


    /* ************************************************
     * Common interfaces
     */

    OWLNamedIndividual getOwlIndividual();

    OWLOntology getOwlOntology();

    void delete();

}
```

DefaultAgent.java

```
package pack1.impl;

import pack1.*;


import java.net.URI;
import java.util.Collection;
import javax.xml.datatype.XMLGregorianCalendar;

import org.protege.owl.codegeneration.WrappedIndividual;
import org.protege.owl.codegeneration.impl.WrappedIndividualImpl;

import org.protege.owl.codegeneration.inference.CodeGenerationInference;

import org.semanticweb.owlapi.model.IRI;
import org.semanticweb.owlapi.model.OWLOntology;


/**
 * Generated by Protege (http://protege.stanford.edu).<br>
 * Source Class: DefaultAgent <br>
 * @version generated on Mon Dec 14 22:29:45 EET 2020 by Mykyta_Levashov
 */
public class DefaultAgent extends WrappedIndividualImpl implements Agent {

    public DefaultAgent(CodeGenerationInference inference, IRI iri) {
        super(inference, iri);
    }

    /* ***************************************************
     * Object Property http://www.semanticweb.org/denys_levashov/iot#hasAttribute
     */

    public Collection<? extends Attribute> getHasAttribute() {
        return getDelegate().getPropertyValues(getOwlIndividual(),
                                    Vocabulary.OBJECT_PROPERTY_HASATTRIBUTE,
                                    DefaultAttribute.class);
    }

    public boolean hasHasAttribute() {
                return !getHasAttribute().isEmpty();
    }

    public void addHasAttribute(Attribute newHasAttribute) {
        getDelegate().addPropertyValue(getOwlIndividual(),
                                Vocabulary.OBJECT_PROPERTY_HASATTRIBUTE,
                                newHasAttribute);
    }
```

```
public void removeHasAttribute(Attribute oldHasAttribute) {
   getDelegate().removePropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_HASATTRIBUTE,
                           oldHasAttribute);
}


/* ***************************************************
 * Object Property http://www.semanticweb.org/denys_levashov/iot#hasDeployment
 */

public Collection<? extends Deployment> getHasDeployment() {
   return getDelegate().getPropertyValues(getOwlIndividual(),
                              Vocabulary.OBJECT_PROPERTY_HASDEPLOYMENT,
                              DefaultDeployment.class);
}

public boolean hasHasDeployment() {
            return !getHasDeployment().isEmpty();
}

public void addHasDeployment(Deployment newHasDeployment) {
   getDelegate().addPropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_HASDEPLOYMENT,
                           newHasDeployment);
}

public void removeHasDeployment(Deployment oldHasDeployment) {
   getDelegate().removePropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_HASDEPLOYMENT,
                           oldHasDeployment);
}


/* ***************************************************
 * Object Property http://www.semanticweb.org/denys_levashov/iot#hasService
 */

public Collection<? extends Service> getHasService() {
   return getDelegate().getPropertyValues(getOwlIndividual(),
                              Vocabulary.OBJECT_PROPERTY_HASSERVICE,
                              DefaultService.class);
}

public boolean hasHasService() {
            return !getHasService().isEmpty();
}

public void addHasService(Service newHasService) {
   getDelegate().addPropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_HASSERVICE,
                           newHasService);
}
```

```
public void removeHasService(Service oldHasService) {
   getDelegate().removePropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_HASSERVICE,
                           oldHasService);
}


/* ***************************************************
 * Object Property http://www.semanticweb.org/denys_levashov/iot#hasWorkingStatus
 */

public Collection<? extends WorkingStatus> getHasWorkingStatus() {
   return getDelegate().getPropertyValues(getOwlIndividual(),
                              Vocabulary.OBJECT_PROPERTY_HASWORKINGSTATUS,
                              DefaultWorkingStatus.class);
}

public boolean hasHasWorkingStatus() {
            return !getHasWorkingStatus().isEmpty();
}

public void addHasWorkingStatus(WorkingStatus newHasWorkingStatus) {
   getDelegate().addPropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_HASWORKINGSTATUS,
                           newHasWorkingStatus);
}

public void removeHasWorkingStatus(WorkingStatus oldHasWorkingStatus) {
   getDelegate().removePropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_HASWORKINGSTATUS,
                           oldHasWorkingStatus);
}


/* ***************************************************
 * Object Property http://www.semanticweb.org/denys_levashov/iot#isAccosiatedWithObject
 */

public Collection<? extends Object> getIsAccosiatedWithObject() {
   return getDelegate().getPropertyValues(getOwlIndividual(),
                              Vocabulary.OBJECT_PROPERTY_ISACCOSIATEDWITHOBJECT,
                              DefaultObject.class);
}

public boolean hasIsAccosiatedWithObject() {
            return !getIsAccosiatedWithObject().isEmpty();
}

public void addIsAccosiatedWithObject(Object newIsAccosiatedWithObject) {
   getDelegate().addPropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_ISACCOSIATEDWITHOBJECT,
                           newIsAccosiatedWithObject);
```

```
    }

    public void removeIsAccosiatedWithObject(Object oldIsAccosiatedWithObject) {
        getDelegate().removePropertyValue(getOwlIndividual(),
                                  Vocabulary.OBJECT_PROPERTY_ISACCOSIATEDWITHOBJECT,
                                  oldIsAccosiatedWithObject);
    }


}
```

Device.java

```
        package pack1;

import java.net.URI;
import java.util.Collection;
import javax.xml.datatype.XMLGregorianCalendar;

import org.protege.owl.codegeneration.WrappedIndividual;

import org.semanticweb.owlapi.model.OWLNamedIndividual;
import org.semanticweb.owlapi.model.OWLOntology;

/**
 *
 * <p>
 * Generated by Protege (http://protege.stanford.edu). <br>
 * Source Class: Device <br>
 * @version generated on Mon Dec 14 22:29:45 EET 2020 by Mykyta_Levashov
 */

public interface Device extends WrappedIndividual {

    /* ***************************************************
     * Property http://www.semanticweb.org/denys_levashov/iot#hasAttribute
     */

    /**
     * Gets all property values for the hasAttribute property.<p>
     *
     * @returns a collection of values for the hasAttribute property.
     */
    Collection<? extends Attribute> getHasAttribute();

    /**
     * Checks if the class has a hasAttribute property value.<p>
     *
     * @return true if there is a hasAttribute property value.
     */
    boolean hasHasAttribute();
```

```
/**
 * Adds a hasAttribute property value.<p>
 *
 * @param newHasAttribute the hasAttribute property value to be added
 */
void addHasAttribute(Attribute newHasAttribute);

/**
 * Removes a hasAttribute property value.<p>
 *
 * @param oldHasAttribute the hasAttribute property value to be removed.
 */
void removeHasAttribute(Attribute oldHasAttribute);


/* ************************************************
 * Property http://www.semanticweb.org/denys_levashov/iot#hasService
 */

/**
 * Gets all property values for the hasService property.<p>
 *
 * @returns a collection of values for the hasService property.
 */
Collection<? extends Service> getHasService();

/**
 * Checks if the class has a hasService property value.<p>
 *
 * @return true if there is a hasService property value.
 */
boolean hasHasService();

/**
 * Adds a hasService property value.<p>
 *
 * @param newHasService the hasService property value to be added
 */
void addHasService(Service newHasService);

/**
 * Removes a hasService property value.<p>
 *
 * @param oldHasService the hasService property value to be removed.
 */
void removeHasService(Service oldHasService);


/* ************************************************
 * Property http://www.semanticweb.org/denys_levashov/iot#hasWorkingStatus
 */
```

```
/**
 * Gets all property values for the hasWorkingStatus property.<p>
 *
 * @returns a collection of values for the hasWorkingStatus property.
 */
Collection<? extends WorkingStatus> getHasWorkingStatus();

/**
 * Checks if the class has a hasWorkingStatus property value.<p>
 *
 * @return true if there is a hasWorkingStatus property value.
 */
boolean hasHasWorkingStatus();

/**
 * Adds a hasWorkingStatus property value.<p>
 *
 * @param newHasWorkingStatus the hasWorkingStatus property value to be added
 */
void addHasWorkingStatus(WorkingStatus newHasWorkingStatus);

/**
 * Removes a hasWorkingStatus property value.<p>
 *
 * @param oldHasWorkingStatus the hasWorkingStatus property value to be removed.
 */
void removeHasWorkingStatus(WorkingStatus oldHasWorkingStatus);


/* ************************************************
 * Property http://www.semanticweb.org/denys_levashov/iot#isAccosiatedWithObject
 */

/**
 * Gets all property values for the isAccosiatedWithObject property.<p>
 *
 * @returns a collection of values for the isAccosiatedWithObject property.
 */
Collection<? extends Object> getIsAccosiatedWithObject();

/**
 * Checks if the class has a isAccosiatedWithObject property value.<p>
 *
 * @return true if there is a isAccosiatedWithObject property value.
 */
boolean hasIsAccosiatedWithObject();

/**
 * Adds a isAccosiatedWithObject property value.<p>
 *
 * @param newIsAccosiatedWithObject the isAccosiatedWithObject property value to be added
 */
void addIsAccosiatedWithObject(Object newIsAccosiatedWithObject);
```

```
    /**
     * Removes a isAccosiatedWithObject property value.<p>
     *
     * @param oldIsAccosiatedWithObject the isAccosiatedWithObject property value to be removed.
     */
    void removeIsAccosiatedWithObject(Object oldIsAccosiatedWithObject);


    /* ***************************************************
     * Common interfaces
     */

    OWLNamedIndividual getOwlIndividual();

    OWLOntology getOwlOntology();

    void delete();

}
```

DefaultDevice.java

```
package pack1.impl;

import pack1.*;


import java.net.URI;
import java.util.Collection;
import javax.xml.datatype.XMLGregorianCalendar;

import org.protege.owl.codegeneration.WrappedIndividual;
import org.protege.owl.codegeneration.impl.WrappedIndividualImpl;

import org.protege.owl.codegeneration.inference.CodeGenerationInference;

import org.semanticweb.owlapi.model.IRI;
import org.semanticweb.owlapi.model.OWLOntology;


/**
 * Generated by Protege (http://protege.stanford.edu).<br>
 * Source Class: DefaultDevice <br>
 * @version generated on Mon Dec 14 22:29:45 EET 2020 by Mykyta_Levashov
 */
public class DefaultDevice extends WrappedIndividualImpl implements Device {

    public DefaultDevice(CodeGenerationInference inference, IRI iri) {
        super(inference, iri);
    }
```

```
/* ************************************************
 * Object Property http://www.semanticweb.org/denys_levashov/iot#hasAttribute
 */

public Collection<? extends Attribute> getHasAttribute() {
   return getDelegate().getPropertyValues(getOwlIndividual(),
                                Vocabulary.OBJECT_PROPERTY_HASATTRIBUTE,
                                DefaultAttribute.class);
}

public boolean hasHasAttribute() {
            return !getHasAttribute().isEmpty();
}

public void addHasAttribute(Attribute newHasAttribute) {
   getDelegate().addPropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_HASATTRIBUTE,
                           newHasAttribute);
}

public void removeHasAttribute(Attribute oldHasAttribute) {
   getDelegate().removePropertyValue(getOwlIndividual(),
                             Vocabulary.OBJECT_PROPERTY_HASATTRIBUTE,
                             oldHasAttribute);
}


/* ************************************************
 * Object Property http://www.semanticweb.org/denys_levashov/iot#hasService
 */

public Collection<? extends Service> getHasService() {
   return getDelegate().getPropertyValues(getOwlIndividual(),
                                Vocabulary.OBJECT_PROPERTY_HASSERVICE,
                                DefaultService.class);
}

public boolean hasHasService() {
            return !getHasService().isEmpty();
}

public void addHasService(Service newHasService) {
   getDelegate().addPropertyValue(getOwlIndividual(),
                           Vocabulary.OBJECT_PROPERTY_HASSERVICE,
                           newHasService);
}

public void removeHasService(Service oldHasService) {
   getDelegate().removePropertyValue(getOwlIndividual(),
```

```
                        Vocabulary.OBJECT_PROPERTY_HASSERVICE,
                        oldHasService);
}


/* ************************************************
 * Object Property http://www.semanticweb.org/denys_levashov/iot#hasWorkingStatus
 */

public Collection<? extends WorkingStatus> getHasWorkingStatus() {
   return getDelegate().getPropertyValues(getOwlIndividual(),
                            Vocabulary.OBJECT_PROPERTY_HASWORKINGSTATUS,
                            DefaultWorkingStatus.class);
}

public boolean hasHasWorkingStatus() {
            return !getHasWorkingStatus().isEmpty();
}

public void addHasWorkingStatus(WorkingStatus newHasWorkingStatus) {
   getDelegate().addPropertyValue(getOwlIndividual(),
                        Vocabulary.OBJECT_PROPERTY_HASWORKINGSTATUS,
                        newHasWorkingStatus);
}

public void removeHasWorkingStatus(WorkingStatus oldHasWorkingStatus) {
   getDelegate().removePropertyValue(getOwlIndividual(),
                        Vocabulary.OBJECT_PROPERTY_HASWORKINGSTATUS,
                        oldHasWorkingStatus);
}


/* ************************************************
 * Object Property http://www.semanticweb.org/denys_levashov/iot#isAccosiatedWithObject
 */

public Collection<? extends Object> getIsAccosiatedWithObject() {
   return getDelegate().getPropertyValues(getOwlIndividual(),
                            Vocabulary.OBJECT_PROPERTY_ISACCOSIATEDWITHOBJECT,
                            DefaultObject.class);
}

public boolean hasIsAccosiatedWithObject() {
            return !getIsAccosiatedWithObject().isEmpty();
}

public void addIsAccosiatedWithObject(Object newIsAccosiatedWithObject) {
   getDelegate().addPropertyValue(getOwlIndividual(),
                        Vocabulary.OBJECT_PROPERTY_ISACCOSIATEDWITHOBJECT,
                        newIsAccosiatedWithObject);
}

public void removeIsAccosiatedWithObject(Object oldIsAccosiatedWithObject) {
```

```
        getDelegate().removePropertyValue(getOwlIndividual(),
                                Vocabulary.OBJECT_PROPERTY_ISACCOSIATEDWITHOBJECT,
                                oldIsAccosiatedWithObject);
    }


}
```

Нижче представлені основні класи програми.

FermerAgent.java

```
package jadedemo;

import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;
import jade.wrapper.StaleProxyException;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;


public class FermerAgent extends Agent {
  private static final long serialVersionUID = 1L;

  protected void setup() {


    CyclicBehaviour loop = new CyclicBehaviour(this) {
      private static final long serialVersionUID = 1L;

      @Override
      public void action() {


        ACLMessage aclMsg = receive();


        if (aclMsg != null) {

          System.out
              .println(myAgent.getLocalName() + "> Received message from: " + aclMsg.getSender());
          System.out.println("Received solution: " + aclMsg.getContent());
        }

        System.out.println("Enter the task:");

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String in;
        try {
          in = reader.readLine();
          if (!in.equals("stop")) {
            System.out.println("Assigning task ...");
            ACLMessage newMsg = new ACLMessage(ACLMessage.REQUEST);
            newMsg.addReceiver(new AID("AgentTwo", AID.ISLOCALNAME));
```

```
          newMsg.setContent(in);
          send(newMsg);
         } else {
          System.out.println("Stopping ...");
          // Shut down main container
          Thread stopContainer = new Thread(() -> {
           try {
            getContainerController().kill();
           } catch (StaleProxyException e) {
                  e.printStackTrace();
           }
          });
          stopContainer.start();
         }
        } catch (IOException e) {
         e.printStackTrace();
        }
        block(); // Stop the behaviour until next message is received
      }
     };
            addBehaviour(new OneShotBehaviour() {
                  @Override
                  public void action() {
                        System.out.println("new LampAgent agent                    Created !!");
                  }
             });

             addBehaviour(new ReceiveAndOrderProductsBehavior());

     addBehaviour(loop);
   }

public int getDesiredQuantityOf(LampAgent redness) {
             double pmax = getPmax(product.getUnitPrice());
             int qmax = getQmax(product.getUnitPrice());
             double m = -qmax/pmax;
             double p = product.getUnitPrice();
             int q = (int) (Math.round(m*p + DEMAND_CONSTANT));
             return q>0 ? q:0;
       }


       private int getQmax(TomatoSensorAgent agent) {
             return  (int)Math.round(MAX_BUDGET*(1-Math.random()) / productPrice);
       }

       private double getPmax() {
             double augmentation = Math.random();
             while (augmentation > MAX_AUGMENTATION) augmentation = Math.random();
             return productPrice * ( 1 + augmentation );
       }


}
```

TomatoSensorAgent.java

```
package jadedemo;
```

```
import jade.core.Agent;
import jade.core.behaviours.CyclicBehaviour;
import jade.lang.acl.ACLMessage;



public class TomatoSensorAgent extends Agent {
  private static final long serialVersionUID = 1L;

  protected void setup() {


    CyclicBehaviour loop = new CyclicBehaviour(this) {
      private static final long serialVersionUID = 1L;

      @Override
      public void action() {


        ACLMessage aclMsg = receive();


        if (aclMsg != null) {
          System.out.println(myAgent.getLocalName()
              + "> Received message from: " + aclMsg.getSender());
          System.out.println("Message content: " + aclMsg.getContent());

        }
        block();
      }
    };
    addBehaviour(loop);
  }
}
```

LampAgent.java

```
package jadedemo;

import java.awt.EventQueue;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JOptionPane;

import jade.core.Agent;
import jade.core.behaviours.OneShotBehaviour;
import jade.wrapper.AgentController;
import jade.wrapper.ContainerController;
import jade.wrapper.StaleProxyException;
import ma.ensias.sma.behaviors.AdvertiseProductBehavior;
import ma.ensias.sma.behaviors.ReceiveOrdersBehavior;
import ma.ensias.sma.views.TomatoGUI;

public class LampAgent extends Agent {

        private ProducerGUI window;
        private List<String> tomatoAgentsNames = new ArrayList<>();
```

```
private List<Order> orders = new ArrayList<>();

public Product getProduct() {
	return product;
}
public List<String> getLampAgentsNames() {
	return tomatoAgentsNames;
}

@Override
protected void setup() {
	addBehaviours();
	showGUI();
}

private void addBehaviours() {
	addBehaviour(new OneShotBehaviour() {
		@Override
		public void action() {
			System.out.println("tomatoAgent Agent Created !!");
		}
	});
	/* The Behavior to Send the Product is added
	 * Dynamiccaly via the GUI
	 * */
	addBehaviour(new ReceiveOrdersBehavior());
}

private void showGUI() {
	EventQueue.invokeLater(new Runnable() {
		public void run() {
			try {
				window = new TomatoAgentGUI(Tomato.this);
				window.setVisible(true);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	});
}

public void advertiseProduct(Product product) {
	this.product = product;
	this.orders.clear();
	System.out.println("Advertising " + product.toString() + "...");
	addBehaviour(new AdvertiseProductBehavior());
}

public int createLampAgent() throws StaleProxyException {
	int numberOfLampAgents = LampAgentsNames.size() + 1;
	String tomatoAgentName = "Consommateur" + numberOfLampAgents;
	ContainerController container = this.getContainerController();
	if (container != null) {
		AgentController LampAgent = container.createNewAgent(LampAgentName,
				LampAgent.class.getName(), new Object[]{});
		lampAgent.start();
	} else {
		System.out.println("getContainerController() returns NULL !!");
	}
```

```
        lampAgentsNames.add(LampAgentName);
        return numberOfLampAgents;
    }

        public void saveOrderIfBuyerCanAfford(Order order, String tomatoAgentName) {
        order.setProduct(product);
        if (order.getQuantity() < 1) {
            window.showAlert(tomatoAgentName+" don't want this product");
            return;
        }
        orders.add(order);
        window.showAlert(tomatoAgentName+" wants "+order.getQuantity()+" of "+product.getName());
        updateSellingReport();
    }

    private void updateSellingReport() {
        int totalQuantitySold = 0;
        double amountOfProfit = 0;

        for(Order o : orders) {
            Product soldProduct = o.getProduct();
            totalQuantitySold += o.getQuantity();
            amountOfProfit += o.getQuantity() * (soldProduct.getUnitPrice() - soldProduct.getUnitCost());
        }

        window.updateProductReport(totalQuantitySold, amountOfProfit);
    }

}
```

ДОДАТОК В

Екранні форми програм агентів для тестування розробленої онтології

Рисунок В.1 – Вхідні данні 1

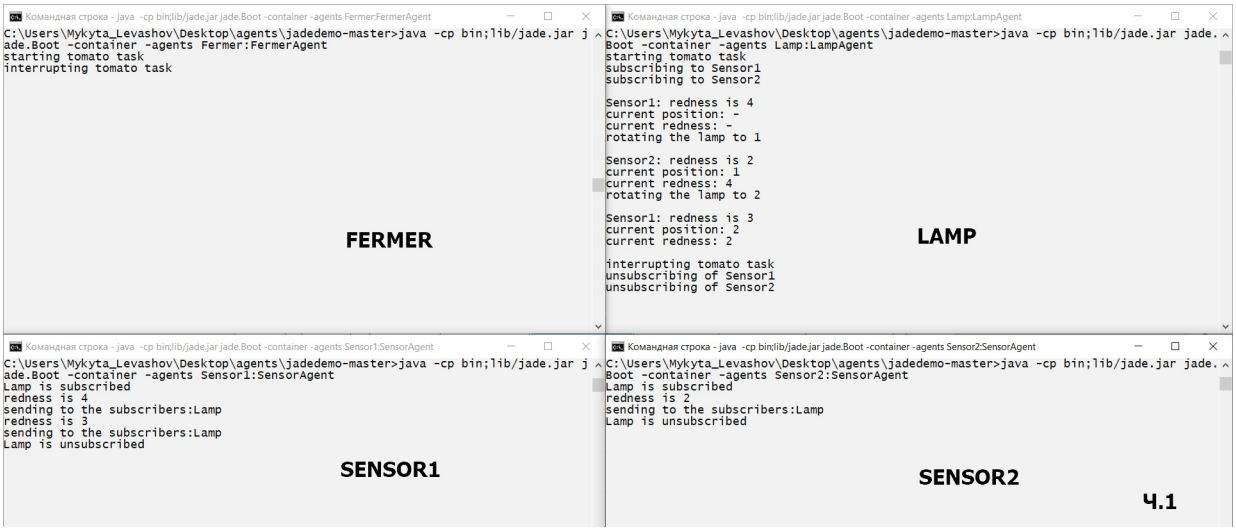

Рисунок В.2 – Взаємодія агентів у задачі виміру показників червності

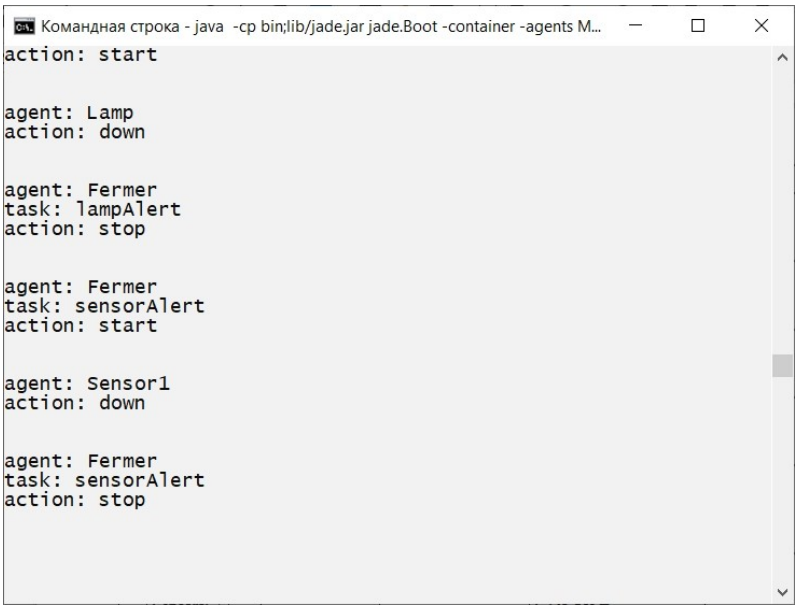

Рисунок В.3 – Вхідні данні 2

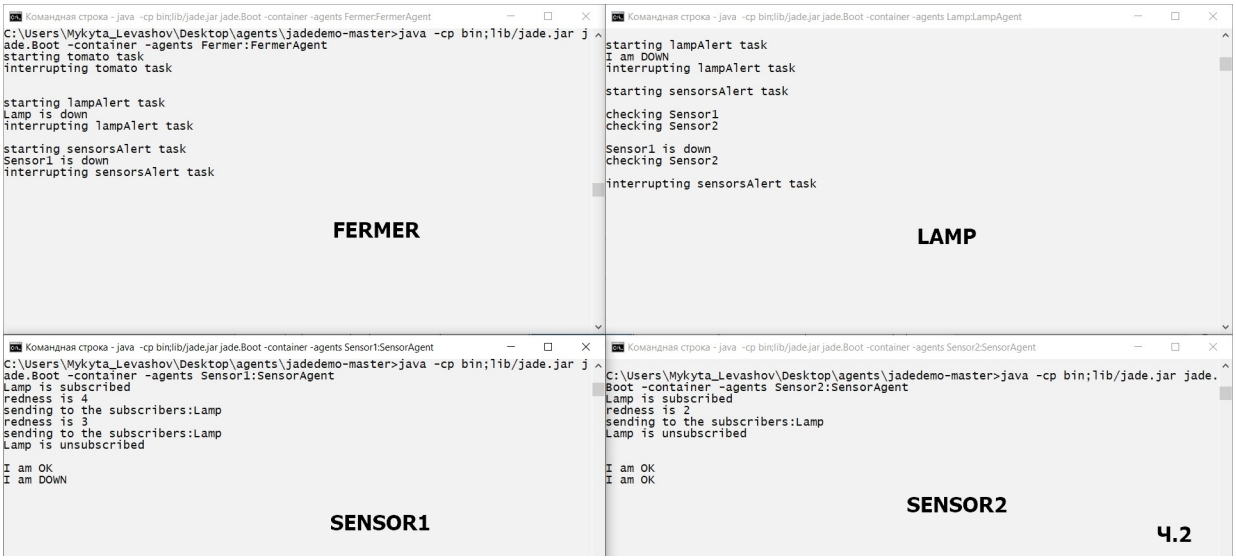

Рисунок В.4 – Взаємодія агентів агентів в задачах сповіщення про несправність лампи та датчиків