Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

# КВАЛІФІКАЦІЙНА РОБОТА

Метод удосконаленого модульного проєктування агентно-орієнтованих
Назва теми
програмних систем з підтримкою розширюваності та повторного використання

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ. 240168.01.10.ПЗ

Виконав студент 2 курсу, група ІПЗм-24-1 _____ Павло СТЕЦЮК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. техн. наук, доцент _____ Юрій ФОРКУН
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент _____ Оксана ЯШИНА
Підпис Ім'я, ПРІЗВИЩЕ

**До захисту допускаю**:
Завідувач кафедри
інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

15 грудня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Другий (магістерський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Леонід БЕДРАТЮК

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Стецюк Павло Петрович
Прізвище, ім'я, по батькові здобувача

1. Тема роботи Метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання

Керівник роботи Форкун Ю. В. , кандидат технічних наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету 25.08.2025 р. № 65

2. Строк подання студентом роботи на кафедру 01.12.2025 р.

3. Вихідні дані до роботи Матеріали науково-дослідної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження предметної області та постановка задачі
2. Розробка методу удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання
3. Технологія реалізації методу удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання
4. Реалізація та тестування програмного засобу

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Презентаційні матеріали (слайди)

6. Консультанти розділів кваліфікаційної роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| Антиплагіат | Форкун Ю. В., доцент | 1.12.25 | 14.12.25 |
| Нормоконтроль | Яшина О. М., доцент | 1.12.25 | 14.12.25 |

7. Дата видачі завдання «1» 09 2025 р.

КАЛЕНДАРНИЙ ПЛАН

| Назва етапів (розділів) кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|
| 1 Вивчення предметної області, формулювання мети, завдань, об'єкта й предмета дослідження; | 20.10-26.10.2025 | 02.09-10.09.2024 |
| 2 Розділ 1 кваліфікаційної роботи — аналіз предметної області та постановка задач і завдання; | 20.10-26.10.2025 | *виконано* |
| 3 Розділ 2 — визначення теоретичних засад модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання | 27.10-02.11.2025 | *виконано* |
| 4 Робота над науковими статтями | 03.11-09.11.2025 | *виконано* |
| 5 Розділ 3 — формування архітектурної моделі та визначення методів забезпечення узгодженості й стійкості системи; | 10.11-16.11.2025 | *виконано* |
| 6 Робота над розділом 4 кваліфікаційної роботи - програмна реалізація рішень, тестування та валідація ПС; | 17.11-23.11.2025 | *виконано* |
| 7 Попередній захист кваліфікаційної роботи | 24.11-31.11.2025 | *виконано* |
| 8 Узгодження постановки задачі, отриманих результатів, висновків; оформлення пояснювальної записки, оформлення графічних матеріалів згідно чинних вимог | 01.12-07.12.2025 | *виконано* |
| 9 Перевірка роботи на наявність плагіату, нормоконтроль, брошурування пояснювальної записки; підготовка супровідних документів | 08.12-14.12.2025 | *виконано* |
| 10 Підготовка до захисту кваліфікаційної роботи | з 15.12.2025 р. | *виконано* |

Студент _____ П. П. СТЕЦЮК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Ю. В. ФОРКУН
Підпис Ім'я, ПРІЗВИЩЕ

# РЕФЕРАТ

Тема дипломної роботи: «Метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання».

Автор: Стецюк Павло Петрович

Керівник: Форкун Юрій Вікторович

Магістерська робота складається з: 94 с., 28 рис., 4 дод., 47 джерел.

ПРОГРАМНА СИСТЕМА, АГЕНТ, МОДУЛЬНЕ ПРОЕКТУВАННЯ, АГЕНТНО-ОРІЄНТОВАНЕ ПРОЕКТУВАННЯ.

Об’єктом дослідження є процес моделювання та проектування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання.

Предметом дослідження є методи, архітектурні принципи агентно-орієнтованих програмних систем та програмні засоби, які забезпечують удосконалення та використання з підтримкою розширюваності та повторного використання.

Мета нашого дослідження полягає розробці методу удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання з застосуванням прототипування, за рахунок використання агентно-орієнтованого підходу в програмуванні для розширення властивостей програмних систем та їх потенційних можливостей, а також для підвищення швидкості їх розробки.

У процесі роботи для вирішення поставлених в кваліфікаційній роботі задач нами було застосовано різноманітні емпіричні, теоретичні та наукові методи дослідження: абстрагування, порівняння, аналізу та синтезу, гіпотезу, формалізацію. У роботі застосовано агентно-орієнтований підхід в програмуванні в поєднанні з обєктно орінтованою технологією, засоби проектування архітектури та програмних рішень. Також було використано теорію множин і реліційну алгебру

та відповідних властивостей властивостей об'єктів, та використання методів математичної статистики, як при оброці даних дослідження процесу та тестування застосунку.

У роботі було запропоновано метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання. За рахунок удосконалення нами методу агентно-орієнтованого програмування при розробці програмних систем це дозволило пришвидшити та підвищити ефективність як самих систем так і їх роботи загалом.

В проведеному дослідженні процесу проєктування програмного застосунку, в ході якого було удосконалено модель реалізації удосконаленого модульного проєктування агентно-орієнтованих програмних систем при розробці програмних систем, було встановлено швидкість реалізації програмного інтерфейсу користувача зменшилась на декілька відсотків в залежності від типу застосунку.

Практична значимість результатів роботи полягає в забезпечені більш швидкої реалізації модульного проєктування агентно-орієнтованих програмних систем та швидкості розробки інтерфейсу програмних систем, за рахунок удосконалення підтримки розширюваності та повторного використання коду з застосуванням прототипування.

15.12.2025

Підпис

Дата

# ABSTRACT

Thesis topic: "Method of advanced modular design of agent-oriented software systems with support for extensibility and reuse".

Author: Stetsyuk Pavlo Petrovich

Head: Yuriy Viktorovich Forkun

The master's thesis consists of: 94 pages, 28 figs., 4 additions, 47 sources.

SOFTWARE SYSTEM, AGENT, MODULAR DESIGN, AGENT-ORIENTED DESIGN.

The object of research is the process of modeling and designing agent-oriented software systems with support for extensibility and reuse.

The subject of the study is the methods, architectural principles of agent-oriented software systems and software tools that provide improvement and use with support for extensibility and reuse.

The purpose of our research is to develop a method for advanced modular design of agent-based software systems with support for extensibility and reuse using prototyping, through the use of an agent-based approach in programming to extend the properties of software systems and their potential, as well as to increase the speed of their development.

In the process of work, to solve the problems set in the qualification work, we applied a variety of empirical, theoretical and scientific research methods: abstraction, comparison, analysis and synthesis, hypothesis, formalization. The work uses an agent-based approach to programming in combination with object-oriented technology, architecture design tools and software solutions. Set theory and relational algebra were also used to develop models of input data sets for the database and the application itself and the corresponding properties of object properties, and the use of mathematical statistics methods such as process research data and application testing.

software systems with support for extensibility and reuse. Due to our improvement of the method of agent-oriented programming in the development of software systems, this made it possible to speed up and increase the efficiency of both the systems themselves and their work in general.

In the conducted study of the software application design process, during which the model for the implementation of advanced modular design of agent-oriented software systems in the development of software systems was improved, the speed of implementation of the program user interface was found to have decreased by several percent, depending on the type of application.

The practical significance of the results of the work lies in ensuring a faster implementation of modular design of agent-oriented software systems and the speed of development of the interface of software systems, due to the improvement of extensibility support and reuse of code using prototyping.

Signature

15.12.2025

Date

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

| | | |
|---|---|---|
| АОПП | | Агентно-орієнтований підхід пророектування |
| ЖЦ | | життєвий цикл |
| ООПП | | об’єктно-орієнтований підхід проектування |
| ПОА | - | програмний орієнтований агент; |
| ПС | - | програмна система |
| ACL | | Agent Communication Language |
| AI | - | Artificial Intelligence |
| AOSE | | Agent-Oriented Software Engineering |
| LLM | - | Large Language Model |
| NLP | - | Natural Language Processing (обробка природної мови) |
| MCA | - | Multi-Channel Aggregation |
| IMC | - | Intelligent Message Classification (інтелектуальна класифікація повідомлень) |
| JADE | - | Платформа розробки агентів Java |
| MAS | - | Multi-Agent System |
| UI | | інтерфейс користувача |

# ВСТУП

Головна ідея проєкту полягає у створенні удосконаленого методу модульного проєктування агентно-орієнтованих програмних систем, спрямованого на забезпечення розширюваності, адаптивності та повторного використання архітектурних рішень і програмних компонентів. Сучасні багатокомпонентні системи базуються на складних взаємодіях автономних програмних агентів, поведінка яких повинна не лише відповідати поточним вимогам, але й легко масштабуватися та модифікуватися у процесі еволюції системи. Існуючі методи проєктування забезпечують важливі підходи до специфікації ролей, організаційних структур і комунікаційних протоколів, проте вони часто не забезпечують достатнього рівня модульності та стандартизованого повторного використання модулів поведінки, агентних шаблонів та міжагентних інтерфейсів.

Запропонований у проєкті метод вирішує ці обмеження через побудову модульної архітектури нового типу, у якій кожен агент розглядається як сукупність формалізованих поведінкових модулів, що мають чітко визначені інтерфейси, залежності та правила комбінування. Такий підхід дає змогу створювати агенти шляхом композиції перевірених модулів, а також спрощує подальші зміни у системі без необхідності перебудови всієї агентної взаємодії. Основною особливістю методу є інтеграція каталогу повторно використовуваних компонентів, який містить типові модулі сприйняття, планування, комунікації, адаптації, навчання та координації. Завдяки цьому, проєктувальник може обирати готові модулі або створювати нові за уніфікованими правилами.

Ключовим елементом методу є впровадження формальної моделі модульної структури агента, що поєднує концепції інкапсульованої поведінки, слабкого зв’язування та стандартизованих контрактів між модулями. Це забезпечує сумісність компонентів, можливість автоматизованої перевірки коректності їх взаємодії та підвищує надійність розроблених систем. Додатково пропонується використовувати механізми просунутого повторного використання, що охоплюють: шаблони комунікаційних протоколів, типові конфігурації

мультиагентних організацій, фрагменти поведінкових моделей та варіанти реалізації стратегій координації.

Актуальність дослідження на відміну від інших підходів щодо розробки програмних систем, на даний момент, здійснюється на основі програмно орієнтованих агентів і є досить перспективною гілкою розвитку для проектування сучасних рішень. Програмно орієнтовані агенти є однією з найбільш перспективних галузей інженерії програмного забезпечення.

Сучасні системи проектування програмного забезпечення здебільшого створюються на основі об'єктно-орієнтованих технологій. Проте застосування цього підходу часто зумовлює труднощі з подальшим розвитком архітектури й розширенням функціональних можливостей програмного продукту. До характерних недоліків також належать складнощі під час перенесення програмного забезпечення між різними серверними середовищами, ризики невчасного виконання робіт у процесі розроблення, зростання проблем в управлінні та координації планувальних задач при збільшенні їх кількості, а також ускладнене створення інтерфейсу користувача та обмеження у варіативності властивостей і функціональних опцій. Альтернативні методології, зокрема агентно-орієнтовані підходи, дають змогу мінімізувати перелічені проблеми, знижуючи ступінь зв'язності компонентів, оптимізуючи кількість задіяних системних процесів і забезпечуючи вищий рівень автономності окремих частин програмної системи.

Мета нашого дослідження полягає розробці методу удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання з застосуванням прототипування, за рахунок використання агентно-орієнтованого підходу в програмуванні для розширення властивостей програмних систем та їх потенційних можливостей, а також для підвищення швидкості їх розробки. адаптивних відповідей за допомогою мовних моделей штучного інтелекту.

Об'єктом дослідження є процес моделювання та проектування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання.

Предметом дослідження є методи, архітектурні принципи агентно-орієнтованих програмних систем та програмні засоби, які забезпечують удосконалення та використання з підтримкою розширюваності та повторного використання.

У процесі роботи для вирішення поставлених в кваліфікаційній роботі задач нами було застосовано різноманітні емпіричні, теоретичні та наукові методи дослідження: абстрагування, порівняння, аналізу та синтезу, гіпотезу, формалізацію. У роботі застосовано агентно-орієнтований підхід в програмуванні в поєднанні з обєктно орінтованою технологією, засоби проектування архітектури та програмних рішень. Також було використано теорію множин і реліційну алгебру для розробки моделей множин вхідних даних для бази даних і самого застосунку та відповідних властивостей властивостей об’єктів, та використання методів математичної статистики, як при оброці даних дослідження процесу та тестування застосунку.

У роботі було запропоновано метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання. За рахунок удосконалення нами методу агентно-орієнтованого програмування при розробці програмних систем це дозволило пришвидшити та підвищити ефективність як самих систем так і їх роботи загалом.

В проведеному дослідженні процесу проектування програмного застосунку, в ході якого було удосконалено модель реалізації удосконаленого модульного проєктування агентно-орієнтованих програмних систем при розробці програмних систем, було встановлено швидкість реалізації програмного інтерфейсу користувача зменшилась на декілька відсотків в залежності від типу застосунку.

Практична значимість результатів роботи полягає в забезпечені більш швидкої реалізації модульного проєктування агентно-орієнтованих програмних систем та швидкості розробки інтерфейсу програмних систем, за рахунок удосконалення підтримки розширюваності та повторного використання коду з застосуванням прототипування.

Удосконалення методу модульного проєктування шляхом підтримки розширюваності та повторного використання дозволяє підвищити рівень стандартизації та структурованості процесу створення агентних систем. Він сприяє зменшенню трудомісткості розробки, полегшує супровід та модернізацію програмного забезпечення, а також відкриває можливість швидкого адаптування систем до змін у вимогах та зовнішньому середовищі. Запропонований підхід є універсальним і може застосовуватися у широкому спектрі галузей: від інтелектуального управління ресурсами та автоматизації виробництва до інформаційних систем підтримки прийняття рішень, робототехнічних платформ і кіберфізичних комплексів.

# 1. ПРЕДМЕТНА ОБЛАСТЬ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1. Дослідження предметної області, наукових досліджень та рішень

Зростаюча потреба у створенні високоскладних обчислювальних рішень для інженерії програмного забезпечення, а також активний розвиток агентно-орієнтованих технологій, зокрема агентного проектування та програмування, формують підґрунтя для впровадження сучасних підходів до розроблення програмного забезпечення у цій сфері. Ефективне розв’язання подібних задач можливе лише за умови використання новітніх методів моделювання та формалізації алгоритмів, які описують процеси проектуваннння програмних систем.

Об’єктно-орієнтований підхід проектування (ООПП) часто розглядають, як одне з найпоширеніших рішень для подолання дефіциту фінансових ресурсів на початкових стадіях створення комерційного програмного забезпечення в софтверних компаніях. Його застосування дозволяє передавати продукт замовнику безпосередньо від розробника, офіційного дистриб’ютора чи постачальника, що дає змогу оптимізувати витрати й спростити логістичні процеси. Такий підхід нерідко використовується й у випадках обмеженого бюджету, або потреби швидкого запуску мінімально життєздатної версії продукту, коли важливо скоротити частину витрат, пов’язаних із традиційними моделями поставки.

Разом із тим, об’єктно-орієнтований підхід проектування залишається популярним варіантом організації розробки в новостворених компаніях, оскільки забезпечує порівняно простий старт, зрозумілу структуру побудови застосунків та достатню гнучкість у процесі розширення системи. Його привабливість посилюється також тим, що нові команди можуть використовувати наявні бібліотеки, шаблони та фреймворки, що зменшує початкові витрати часу й ресурсів. У підсумку такий підхід створює сприятливі умови для швидкого формування програмного продукту, дозволяючи компанії зосередитися на ключових функціях та покращенні якості без додаткових організаційних і фінансових навантажень.

Центральним елементом методології, яку представляє об’єктно-орієнтований

підхід проектування, виступає формування архітектури програмного забезпечення. Компанії, що працюють за цим підходом, здебільшого створюють власні програмні продукти на його основі, що слугує важливим показником ефективності застосування зазначеної методики. В інших випадках вони можуть адаптувати вже існуючі програмні рішення, модифікуючи їх відповідно до потреб конкретного проєкту чи замовника.

Разом із тим, під час виконання замовлень нерідко виникають типові ризики та труднощі, пов’язані з обробкою даних у процесі програмування. Значна частина таких проблем пов’язана з недостатньою передбаченістю складу необхідних компонентів або з браком оптимального балансу функціональності в майбутньому продукті. Чітке визначення ризиків, пов’язаних із відсутністю критичних модулів, а також своєчасне коригування архітектурних рішень дають змогу зменшити кількість помилок і підвищити якість розробки.

У багатьох випадках наявні програмні системи не забезпечують того функціоналу, який потрібен для розв’язання спеціалізованих чи нестандартних задач. Саме тому процес створення програмного забезпечення потребує уважного вибору методів і технологій, що дозволяють гарантувати відповідність кінцевого продукту вимогам замовника та специфіці предметної області.

Автоматизація регулярних операцій на веб-серверах часто реалізується шляхом запуску сценаріїв, написаних, мовами Java, PHP, у поєднанні з планувальником завдань. Такий механізм передбачає виконання скриптів за наперед визначеним графіком, який задається у конфігурації планувальника, що забезпечує їх систематичний і передбачуваний запуск. Популярність цього підходу пояснюється тим, що він доступний майже на всіх типах хостингу, легко налаштовується навіть некваліфікованими адміністраторами, а також дозволяє вносити зміни до розкладу чи додавати нові завдання без необхідності втручання у внутрішню логіку застосунку.

Під час тестування прототипу, який використовував цей механізм, окрім очевидних переваг, було виявлено низку істотних обмежень.

По-перше, працездатність системи залежить від наявності та коректної

роботи планувальника завдань у конкретному серверному середовищі. Це може створювати проблеми при перенесенні програмного забезпечення на інший сервер, де конфігурація відрізняється або планувальник відсутній.

По-друге, відлагодження програмного коду ускладнюється, оскільки виконання сценаріїв відбувається фоново та не забезпечує прямого доступу до поточних логів, або проміжних результатів їх роботи на екрані.

По-третє, зі збільшенням кількості сценаріїв суттєво зростають витрати на їх підтримку: необхідно контролювати коректність розкладу виконання, вести логування результатів, відстежувати використання спільних ресурсів, а також керувати залежностями між різними завданнями, що додає додаткової складності адмініструванню системи.

Актуальність проведеного дослідження визначається потребою подолати недоліки, властиві традиційному підходу до створення програмних систем на основі об’єктно-орієнтованого підходу проектування (ООПП), і замінити його вдосконаленим методом, що ґрунтується на агентно-орієнтованому підході проектування (АОПП). Застосування цього нового підходу дає можливість істотно розширити функціональні властивості майбутнього програмного забезпечення, підвищити його адаптивність, покращити масштабованість і забезпечити появу додаткових потенційних можливостей, недосяжних у межах традиційної парадигми (Таблиця 1.1).

Додатково актуальність зумовлюється зростанням складності сучасних інформаційних систем та необхідністю забезпечення їх здатності до самостійного прийняття рішень, координації дій і взаємодії в динамічних середовищах. Агентно-орієнтований підхід проектування створює підґрунтя для формування систем із високим рівнем автономності та інтелектуальної поведінки, що особливо важливо для програмного забезпечення обробки даних, де обсяги даних, швидкість їх обробки та вимоги до персоналізації постійно зростають. Упровадження такого підходу дозволяє не лише подолати обмеження традиційних архітектур, але й формувати програмні рішення нового покоління, які здатні до адаптації, самонавчання та гнучкого реагування на зміни бізнес-процесів.

Таблиця 1.1 – Актуальність агентно-орієнтованого підходу проектування з підтримкою розширюваності та повторного використання

| Недоліки ООПП до розробки програмних систем | Переваги використання (АОПП) при розробці програмних систем |
|---|---|
| Утруднений або навіть неможливий подальший розвиток архітектури та розширення функціонального наповнення програмного застосунку | Зменшення залежностей між компонентами та зниження рівня зв’язності системи |
| Виникнення труднощів при перенесенні програмного забезпечення на інші сервери або інфраструктури | Скорочення кількості системних процесів, що необхідні для підтримки функціональності. |
| Імовірність виникнення затримок та труднощів із дотриманням термінів на різних етапах створення продукту | Спрощення реалізації елементів інтерфейсу та пришвидшення роботи над його удосконаленням |
| Складнощі у керуванні та координації великої кількості задач планувальника, які зростають разом зі масштабом системи | Підвищена автономність окремих модулів та компонентів системи |
| Порівняно складний процес розроблення інтерфейсу користувача та високі витрати на його підтримку | Можливість реалізації механізмів самонавчання та адаптивної поведінки, що враховує індивідуальні звички користувачів |
| Обмеженість доступних властивостей системи та звужений спектр можливих розширень функціоналу | Значне розширення переліку характеристик програмного забезпечення та потенціалу його подальшої модернізації |

Проведений аналіз демонструє, що традиційні підходи до розроблення програмних систем, засновані на об’єктно-орієнтованому проектуванні, хоч і

забезпечують певні переваги на ранніх етапах створення ПЗ, проте мають низку системних обмежень, які ускладнюють подальший розвиток архітектури та масштабування рішень. Виявлені проблеми у сфері обробки даних, перенесення ПЗ, налагодження сценаріїв та управління завданнями підтверджують потребу у більш гнучких і автономних підходах до проектування. Агентно-орієнтований підхід проектування постає перспективною альтернативою, оскільки дозволяє створювати програмні системи з високим рівнем адаптивності, автономності та здатності до самонавчання. Такий підхід забезпечує суттєве розширення функціональності та підвищення ефективності обробки даних у динамічних умовах сучасних інформаційних середовищ. Отже, впровадження агентно-орієнтованих технологій є ключовим напрямом подолання недоліків традиційних архітектур та формування інтелектуальних програмних рішень нового покоління.

## 1.2. Аналіз існуючих методів їх застосування та їх особливості

У ранніх дослідженнях існувала думка, що будь-які системи керування, а також численні програмні демони операційних систем, які працюють у фоновому режимі, відстежують стан середовища та виконують дії для його зміни, можна трактувати як агентів. Згодом така інтерпретація зазнала суттєвої критики, оскільки виявилося, що подібне узагальнення надто розмите й не відображає специфіки агентної поведінки. У сучасній науковій літературі зазначається, що агент може мати різні формальні визначення залежно від контексту, сфери застосування та рівня складності, і наразі існують десятки варіантів його опису. У найзагальнішому вигляді агент визначається, як система, здатна здійснювати дії на основі інформації, що надходить із середовища, яке він сприймає, а також, як суб’єкт, що функціонує автономно та може виконувати завдання від імені іншого учасника взаємодії.

До основних вимірів, які характеризують агентів, належать поведінкові прояви, автономність, реактивність і проактивність, вміння адаптуватися, навчатися, здатність до співпраці та мобільність. Частину цих характеристик прийнято розглядати як базові риси агентської діяльності — автономність,

реактивність, проактивність і соціальні можливості, тоді як інші властивості трактують як більш складні, «людиноподібні» когнітивні утворення, що в агентних системах часто називають ментальними станами. До них належать вірування, бажання, наміри та зобов'язання, які лежать в основі архітектури BDI (belief–desire–intention), у межах якої агент описується множинами B, D та I, що визначають його внутрішню модель поведінки.

Також, крім автономності, яка є ключовою характеристикою агентних систем, усім агентам притаманна спільна особливість – вони виконують дії, делеговані користувачем, представляючи його інтереси у взаємодії з середовищем. Термін «програмний агент» часто використовується, як синонім поняття «агент» у науковій спільноті, і ним позначають програмний об'єкт, здатний реалізувати перелічені властивості, або функціонувати як інкапсульована програмна одиниця. Така система може бути інтелектуальною або автоматизованою, причому автоматизована форма зазвичай є реалізацією конкретних алгоритмів, призначених для вирішення визначених задач.

До програмних агентів також відносять програмні компоненти, які координують обмін інформацією, ведуть діалог, узгоджують дії та підтримують комунікаційну взаємодію між системами. У літературі розрізняють «сильне» трактування агентів, характерне для сфери штучного інтелекту та пов'язане з реалізацією когнітивних моделей, і «слабке», що описує більш спрощені автоматизовані структури, які лише частково демонструють ознаки агентної поведінки.

Дослідники вважають агентно-орієнтований підхід проектування (АОП) до розробки ПЗ природною та логічною еволюцією поточних підходів, таких як структурований та об'єктно-орієнтований підхід проектування (ООПП). АОПП представляє собою вищий рівень інкапсуляції та абстракції, яка є більш гнучкою, ніж попередні підходи до програмування (наприклад, абстракція об'єкта), та якій прогнозується революційність у розробці ПЗ, а саме значне розширення можливостей моделювання, проектування та створювання складних розподілених програмних систем. Вважається, що ПА та мультиагентні системи (Multi-Agent

Systems, MAS) будуть однією із знакових технологій в інформатиці, яка принесе додаткову теоретичну потужність, методи та прийоми, які розширять область застосування ЕОМ.

У науковій спільноті агентно-орієнтований підхід проектування розглядають як закономірний етап розвитку методологій створення програмного забезпечення, що постає після структурованих та об'єктно-орієнтованих підходів проектування. На відміну від попередніх парадигм, АОПП забезпечує значно вищий рівень абстракції й інкапсуляції, пропонуючи більш природну модель представлення складних програмних сутностей та їх взаємодій. Такий підхід вирізняється підвищеною гнучкістю та здатністю описувати поведінку систем, що функціонують у динамічному, розподіленому середовищі, де традиційна об'єктна абстракція виявляється недостатньою. Дослідники прогнозують, що поширення агентних концепцій матиме революційний вплив на практику розробки програмних систем, оскільки вони відкривають нові можливості у сфері моделювання, проектування та побудови інтелектуальних розподілених систем, які здатні до автономної взаємодії та прийняття рішень.

Більше того, програмні агенти та мультиагентні системи (Multi-Agent Systems, MAS) дедалі частіше визначають як одну з ключових технологій майбутнього, що здатна суттєво збагатити теоретичний і практичний апарат інформатики. Завдяки впровадженню нових методів формалізації поведінки, координації та комунікації між автономними компонентами, MAS розширюють функціональність комп'ютерних систем далеко за межі традиційних підходів. Очікується, що такі технології значно розширять сферу застосування електронно-обчислювальних машин, забезпечуючи можливість створення адаптивних, самоорганізованих і стійких до змін програмних комплексів, орієнтованих на вирішення задач підвищеної складності.

Попри значний прогрес у створенні одноагентних рішень, окремий програмно-орієнтований агент (ПОА) як обчислювальна одиниця зазвичай не діє повністю самостійно. Зазвичай його можливості обмежені доступними знаннями, ресурсами та обсягом задач, які він здатен виконувати. Саме тому мультиагентні

системи (Multi-Agent Systems, MAS) тлумачаться як сукупність взаємодіючих ПОА, що працюють у спільному середовищі та використовують загальні ресурси чи сервіси. Подібні системи мають широке коло застосувань: від електронної комерції, мережевих технологій та інформаційної безпеки до транспорту, медицини, систем керування, робототехніки й оборонної сфери тощо. З ускладненням завдань у цих галузях стає дедалі очевиднішим, що традиційні методології розробки ПС уже не забезпечують достатню гнучкість, тоді як MAS позиціонуються як один із найперспективніших напрямів побудови сучасних інтелектуальних систем, хоча процес їх створення залишається концептуально та технологічно непростим.

ПОА активно використовують для реалізації програм із підвищеним рівнем складності, тоді як агентно-орієнтована інженерія програмного забезпечення (AOSE) пропонує методологічну основу для виявлення та усунення численних проєктних проблем. Цей підхід розглядають як наступний етап еволюції після процедурних, об’єктно-орієнтованих і компонентно-орієнтованих моделей розробки. На відміну від класичних об’єктів, ПОА здатні демонструвати автономну поведінку, взаємодіяти з іншими учасниками середовища, адаптуватися до змін та співпрацювати для досягнення спільних цілей. Ці характеристики особливо важливі в застосунках електронної комерції, де необхідно автоматизувати бізнес-процеси, або підвищувати ефективність електронних сервісів завдяки індивідуальній роботі агентів від імені користувача.

Часто ПОА інтегрують із технологіями штучного інтелекту, і MAS розглядаються як частковий випадок систем розподіленого штучного інтелекту, чия популярність швидко зростає з більш як десятиліття. Хоча історично АОПП формувалося в межах штучного інтелекту, останні десятиліття спостерігається активний рух у бік тіснішої інтеграції з інженерією програмного забезпечення та розподіленими обчисленнями. Деякі дослідники наполягають, що АОПП варто трактувати радше як піддисципліну інженерії програмного забезпечення та інформатики, ніж як похідну від штучного інтелекту. Саме в цьому контексті виникла агентно-орієнтована інженерія програмного забезпечення (AOSE),

завданням якої є створення формальних, модельно-орієнтованих підходів до розробки надійних MAS та керування їхнім життєвим циклом. Таким чином AOSE сформувалося як окрема галузь, що поєднує інструментарій інженерії програмного забезпечення та розподіленого штучного інтелекту, забезпечуючи системну підтримку створення складних агентних систем. При цьому інтелектуальність не є обов'язковою властивістю агента, вона залежить від моделі та набору характеристик, закладених у конкретну реалізацію ПОА. Водночас поглиблення зв'язку між AOSE та ШІ сприяє появі нових гібридних підходів, що об'єднують поведінкові та когнітивні моделі агентів. Розвиток цих напрямів формує можливість створення більш адаптивних та автономних систем, здатних ефективно працювати в умовах високої динамічності та невизначеності.

Попри відчутні успіхи агентно-орієнтованої інженерії програмного забезпечення, її вплив на індустріальні процеси розроблення агентних систем поки що не повністю реалізований. У впровадженні цього підходу все ще залишається низка труднощів, які необхідно враховувати під час вибору відповідної технології для практичних рішень. Особливо вагому роль ПОА відіграють у комерційних галузях та виробництві, де вони застосовуються для автоматизації бізнес-процесів, виконання дій від імені користувача та підвищення ефективності управлінських процесів. До типової групи таких агентів належать керуючі агенти, агенти-помічники користувача, моніторингові агенти та агенти збору інформації. Основними перевагами їх використання є зменшення навантаження на кінцевого користувача, адаптація до індивідуальних уподобань, можливість взаємодії з іншими агентами та оптимальне використання ресурсів.

Крім того, сильними сторонами ПОА та MAS є властивості, пов'язані з інтелектом, наявністю знань, здатністю до міркування та навчання, що дає можливість розширювати функціональні можливості системи та підвищувати її ефективність у складних середовищах. Узагальнений перелік таких властивостей подано на відповідних схемах (Рисунок 1.1 та Рисунок 1.2), що демонструє широкий спектр застосувань і потенціал агентно-орієнтованого підходу в сучасній інженерії програмного забезпечення.


Рисунок 1.1 – Властивості агентно-орієнтованого підходу до проектування


Рисунок 1.2 – Напрямки розвитку агентно-орієнтованого підходу до проектування

Внесок програмно-орієнтованих агентів у модульне проєктування агентно-орієнтованих програмних систем є надзвичайно значущим, особливо коли йдеться

про забезпечення можливості масштабування та повторного використання програмних компонентів. ПОА здатні реалізовувати широкий спектр функцій, від кооперації та інтеграції при створенні програмних систем до підвищення економічної ефективності системи. Сучасні тенденції розвитку програмування вимагають створення рішень, що можуть навчатися та генерувати нову корисну поведінку без необхідності детального ручного програмування кожного окремого сценарію, окрім визначених базових операцій.

У більш загальному розумінні ПОА виступають проміжною ланкою між дільнітю софтверних компаній по створенню програмних систем та традиційними бізнес-процесами, оскільки дозволяють передавати запити користувачів відповідним агентам (наприклад, мобільним агентам для обробки специфічних даних), які далі перетворюють ці вимоги у відповідні дії, або операції на окремі відділи та групи.

Підсумовуючи відмітимо, як бачимо аналіз еволюції поняття агента та підходів до його формального визначення свідчить, що сучасні програмно-орієнтовані агенти значно виходять за межі ранніх уявлень про прості системи керування чи фонові процеси. Їм притаманні складні поведінкові характеристики, включно з автономністю, проактивністю, соціальною взаємодією, а також ментальними станами, що лежать в основі когнітивних моделей типу BDI. Агентно-орієнтований підхід проектування, який розвинувся з попередніх парадигм інженерії програмного забезпечення, забезпечує суттєво вищий рівень абстракції й інкапсуляції, що робить його придатним для побудови динамічних і розподілених інтелектуальних систем. Розширення цього підходу в контексті мультиагентних систем дедалі більше утверджує його, як один із ключових напрямів розвитку сучасної інформатики та складних програмних технологій.

Комплексне впровадження програмно-орієнтованих агентів та агентно-орієнтованої інженерії програмного забезпечення демонструє значний потенціал для розв’язання задач, з якими традиційні методології вже не справляються. ПОА та MAS забезпечують можливість створення адаптивних, самонавчальних і здатних до автономної взаємодії систем, що особливо важливо в галузях із високою

динамічністю, таких як електронна комерція, виробництво чи інформаційно-аналітичні системи. Водночас їхня інтеграція зі штучним інтелектом та розподіленими обчисленнями відкриває нові напрями для формування гібридних моделей, здатних ефективно працювати в умовах невизначеності та великої складності. Таким чином, розвиток АОПП і AOSE формує методологічну основу для побудови програмних систем нового покоління, орієнтованих на гнучкість, масштабованість і високий ступінь інтелектуальності.

## 1.3. Висновки першого розділу та постановка задач

Метою першого розділу було комплексне виявлення ключових характеристик агентно-орієнтованого підходу проектування програмних систем, які можуть слугувати концептуальною основою для подолання визначених проблем, що виникають у процесі створення та експлуатації програмних засобів за умов використання моделі еволюційного прототипування та підходу прямих поставок в електронній комерції. Реалізація поставленої мети передбачала глибоке опрацювання теоретичних та прикладних аспектів функціонування агентних систем, що дозволить обґрунтувати доцільність застосування агентно-орієнтованого підходу проектування програмних систем у контексті вирішення існуючих обмежень традиційних методологій розробки.

Для досягнення сформульованої мети у межах розділу було:

– здійснено детальний огляд і системний аналіз наукових публікацій щодо проблем, які розв’язує застосування агентно-орієнтованого підходу;

– досліджено особливості визначення агентів та програмно орієнованих агентів, їх основні характеристики, поведінкові виміри та сфери використання;

– розглянуто класифікаційні ознаки програмних агентів, формальні моделі їх поведінки, механізми міжагентної комунікації, а також визначити їх властивості, сильні та слабкі сторони й актуальні платформи для їх розроблення;

– проведено поглиблений аналіз переваг і недоліків підходу проектування на основі об’єктно-орієнтованого підходу;

– виділено та систематизовано критичні недоліки цього підходу, що можуть бути усунені шляхом використання результатів дослідження процесів обробки вхідних даних, на яких базуються поведінкові механізми програмних агентів під час виконання операцій ціноутворення;

– виконано огляд і порівняльний аналіз основних методів проектування програмних систем, їх можливостей, обмежень та умов застосування у контексті автоматизованих програмних систем;

–проаналізовано та співставлено сучасні підходи до розроблення програмного забезпечення з метою виявлення їхніх переваг, недоліків та потенційних напрямів інтеграції з агентно-орієнтованими технологіями.

Таким чином, у першому розділі здійснено комплексний аналіз сучасних підходів до розроблення програмних систем, який показав, що традиційні методології, основані на об’єктно-орієнтованому проектуванні, хоча й залишаються ефективними на початкових етапах створення програмного забезпечення, все ж демонструють низку структурних обмежень у процесі подальшого розвитку програмних рішень. Особливо це проявляється під час масштабування програмних систем, ускладнення архітектури, зростання кількості взаємозалежних компонентів та необхідності забезпечення високої гнучкості в умовах змін бізнес-вимог.

Виявлені проблеми, пов’язані з обробкою значних обсягів даних, перенесенням застосунків між різними інфраструктурами, налагодженням розподілених сценаріїв виконання та управлінням планувальними завданнями, підтверджують, що традиційні підходи поступово втрачають свою ефективність у контексті сучасних інформаційних систем.

У цьому аспекті агентно-орієнтований підхід проектування демонструє здатність подолати зазначені недоліки, оскільки він формує умови для створення систем із високим рівнем адаптивності, автономності та можливістю до самонавчання.

Такий підхід відкриває широкі можливості щодо розширення функціональності ПЗ, оптимізації обробки даних і забезпечення стійкості систем у

динамічних, розподілених і потенційно нестабільних середовищах.

Таким чином, упровадження агентно-орієнтованих технологій може розглядатися як стратегічний напрям розвитку сучасної інженерії ПЗ, що забезпечує формування інтелектуальних програмних систем нового покоління.

Подальший аналіз еволюції поняття агента та його ролі в сучасних системах дав змогу встановити, що програмно-орієнтовані агенти суттєво виходять за межі традиційного уявлення про автоматизовані компоненти або фонові процеси. Вони характеризуються цілою низкою поведінкових властивостей: автономністю, реактивністю, проактивністю, здатністю до соціальної взаємодії, а також когнітивними характеристиками, що включають вірування, бажання та наміри, характерні для BDI-моделей.

Агентно-орієнтований підхід проектування, який сформувався на перетині класичних інженерних парадигм, забезпечує значно вищий рівень абстракції та є природною основою для побудови складних, розподілених та інтелектуальних систем, здатних до координації, аналізу та прийняття рішень у реальному часі. У межах мультиагентних систем ці переваги проявляються ще яскравіше, поступово перетворюючи АОПП на один із найперспективніших і найдинамічніших напрямів розвитку сучасної інформатики.

Комплексне впровадження ПОА та методів AOSE демонструє значний потенціал розв’язання задач, які виявилися надмірно складними для традиційних технологій, особливо в таких сферах, як електронна комерція, виробничі системи, обробка даних і аналітичні платформи. Додаткова інтеграція агентних технологій зі штучним інтелектом, машинним навчанням і розподіленими обчисленнями створює передумови для формування гібридних архітектур, здатних ефективно діяти в умовах невизначеності, значної складності та швидкої змінності середовища.

Таким чином, результати аналізу підтверджують, що розвиток АОПП та AOSE відіграє ключову роль у становленні програмних систем нового покоління, орієнтованих на інтелектуальність, гнучкість і стійку масштабованість.

Отже, на основі проведеного аналізу для досягнення поставленої мети нами

поставлено такі завдання:

– виконати теоретичне дослідження процесу моделювання програмного забезпечення автоматизованої інформаційної системи, призначеної для актуалізації роздрібних цін;

– встановити кількість задіяних системних процесів, охарактеризувати рівень взаємозалежності між компонентами архітектури, оцінити можливості розширення множини функціональних властивостей та потенційних характеристик програмного продукту, а також здійснити аналіз впливу програмних агентів на апаратні ресурси й визначити обсяг їх використання;

– провести порівняльний аналіз існуючих підходів до розроблення програмного забезпечення, доповнивши його експериментальним дослідженням практичного процесу створення ПС;

– визначити швидкість реалізації користувацького інтерфейсу та ефективність організації асинхронної взаємодії між автономними компонентами системи;

– здійснити експериментальне дослідження алгоритмізації процесу проектування агентно-орієнтованих програмних систем, що виконується програмно орієнтованим агентом

– розробити метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання;

– зпроектувати архітектуру агентно-орієнтованої архітектури програмної системи за розорбленим методом;

– здійснити , та встановити показники точності й загальної ефективності програмної реалізації розробленого алгоритму оброблення даних;

– здійснити програмну реалізацію на основі розробленої архітектури;

– здійснити тестування та валідацію програмного застосунку.

# 2. РОЗРОБКА УДОСКОНАЛЕНОГО МОДУЛЬНОГО ПРОЄКТУВАННЯ АГЕНТНО-ОРІЄНТОВАНИХ ПРОГРАМНИХ СИСТЕМ З ПІДТРИМКОЮ РОЗШИРЮВАНОСТІ ТА ПОВТОРНОГО ВИКОРИСТАННЯ

## 2.1. Методологічні засади вирішення задачі

При визначені методологічних підходів до розробки і самої програмної системи загалом необхідно здійснити порівняльний аналіз підходів до розробки програмних систем та оцінити моделі життєвого циклу проектування програмного застосунку. Однак, перш ніж проводити огляд життєвого циклу програмної системи нам в першу чергу слід встановити, що процес її проектування опирається на структуру та тип самих програмно-орієнтованих агентів.

Отже розробку та проектування методу і самої програмної системи потрібно починати з визначення типу агентів, з якими ми будемо працювати в подальшому, У науковій літературі, присвяченій дослідженню агентних систем, традиційно виділяють дві фундаментальні групи архітектур: реактивні та дорадчі. Реактивні моделі розглядають як відносно прості механізми, що забезпечують миттєве реагування на зміни у зовнішньому середовищі без складних внутрішніх обчислень чи побудови моделей. На противагу їм, дорадчі архітектури демонструють здатність до формування значно складнішої поведінки, оскільки оперують ментальними станами такими як цілі, наміри, переконання та використовують техніки планування, що ґрунтуються на символічних поданнях і внутрішніх моделях середовища. Вони забезпечують можливість стратегічного прийняття рішень, що робить їх придатними для задач, де простого реагування недостатньо. Крім того, у практиці проектування виділяють гібридні архітектури, які поєднують елементи обох підходів, інтегруючи переваги швидкої реакції з механізмами раціонального вибору дій.

Більш глибока технічна класифікація агентів стосується їх внутрішніх станів, тобто моментальних «знімків» системи, що відображають її інформаційний зміст

(наприклад пам’ять чи база даних) у конкретний момент дискретного часу. У випадку агентів така дискретизація зазвичай прив’язана до циклу «відчуття- аналіз-дія», в межах якого агент сприймає вхідні дані, виконує процедури прийняття рішень різної складності та генерує вихідні дії. Наприклад, веб-агент може отримати запит користувача, оновити базу знань і сформувати відповідь протягом одного циклу. Якщо архітектура не оперує множиною станів і не зберігає інформації між циклами, таку модель часто називають архітектурою типу «стимул-реакція».

Оскільки сфери застосування агентів є надзвичайно різноманітними, вони можуть мати широкий спектр характеристик, що проявляються у різних комбінаціях: автономність, конкурентність, кооперативність, соціальність, доброзичливість тощо. Ці риси нерідко використовують як основу для створення класифікаційних схем програмно-орієнтованих агентів. Важливою є також класифікація за призначенням: виділяють інтерфейсні агенти, Інтернет-агенти, Інтранет-агенти та гетерогенні агенти.

Інтерфейсні агенти спостерігають за користувачем, навчаються на основі його поведінки та допомагають оптимізувати робочі процеси. Інтернет-агенти відповідають за пошук, вилучення, фільтрацію та обробку інформаційних ресурсів, тобто будь-яких даних, що мають цінність у певній предметній області та можуть бути використані людиною або програмними засобами. Інтранет-агенти застосовують для автоматизації внутрішніх бізнес-процесів, а гетерогенні поєднують властивості кількох типів.

Окремо у літературі виділяють сім категорій ПОА: співробітницькі, інтерфейсні, мобільні, інформаційні, реактивні, інтелектуальні та гібридні.

З погляду просторової організації, агенти можуть бути статичними або мобільними. Статичні агенти функціонують у фіксованому середовищі, тоді як мобільні здатні переміщуватися між вузлами мережі або різними середовищами виконання, що пов’язано з проблематикою мобільності коду у розподілених системах. Водночас мобільність супроводжується низкою викликів, зокрема питань безпеки, контролю доступу, захисту мережевих ресурсів. Попри це,

мобільні агенти можуть істотно підвищувати ефективність складних операцій, які потребують взаємодії різних обчислювальних компонентів, зокрема у завданнях моніторингу мереж, балансування навантаження, або проведення розподілених обчислень.

З огляду на те, що в рамках розроблюваного програмного забезпечення не передбачається виконання подібних складних операцій, а також зважаючи на ризики, пов’язані з мобільністю, у дослідженні прийнято рішення зосередитися на статичних агентних структурах.

Ще одним важливим критерієм класифікації програмно-орієнтованих агентів є їхній розмір, під яким зазвичай розуміють обсяг внутрішніх ресурсів агента, складність його структури, кількість доступних поведінкових правил та рівень залучених когнітивних механізмів, а також відповідний рівень інтелектуальності, що виявляється у його здатності виконувати цілеспрямовані дії. Зі збільшенням розміру агента зростає кількість оброблюваних ним даних, масштаб внутрішніх моделей середовища та обсяг знань, на основі яких він приймає рішення, що в багатьох випадках прямо впливає на складність і гнучкість його поведінки. У цьому контексті інтелект агента тлумачать не як фіксовану властивість, а як динамічну характеристику, що відображає його можливість формувати розумні дії, адаптуватися до змін та вирішувати проблеми в ситуаціях, де не існує повного чи однозначного алгоритмічного опису шляхів досягнення цілі.

Зазвичай вважається, що інтелектуальність агента корелює з його складністю й обсягом знань, тому виокремлюють агенти великого, середнього та мікророзміру. Агенти великого розміру володіють достатнім набором компетенцій для автономного виконання корисних завдань. Агенти середнього розміру здатні виконувати нетривіальні функції у взаємодії з іншими компонентами. Мікроагенти, іноді розглядаються, як частина Society of Mind окремо не виявляють помітного рівня інтелектуальності, проте можуть демонструвати складну поведінку як результат колективної взаємодії багатьох простих агентів – що є важливим концептуальним положенням агентної теорії.

Беручи до уваги простоту операцій, які мають виконувати агенти в межах

нашої кваліфіувційної роботи, а також враховуючи наведені класифікаційні ознаки, розроблювані програмно-орієнтовані агенти доцільно віднести до класу рефлексивних статичних інтернет-мікроагентів. Статичного агента (Рисунок 2.1) можна формально описати у вигляді відповідного кортежу, що визначає його структуру та функції.


Рисунок 2.1 – Статичний програмно-орієнтований агент

Приклад рефлексивного реагуючого агента наведено на рисунку 2.2.

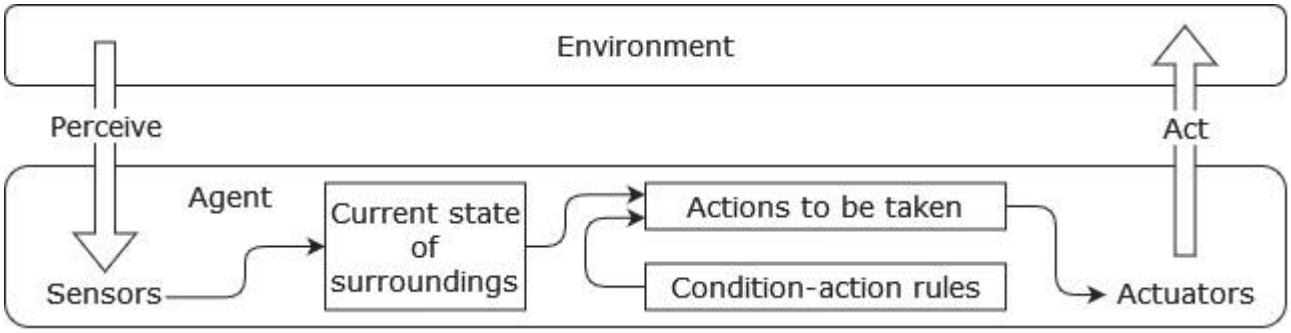



Рисунок 2.2 – Схема взаємодії рефлексивного реагуючого програмно орієнтованого агента з середовищем

Статичний програмно-орієнтований агент (ПОА) концептуально розглядається як класична програмна система, у структурі якої закладено наперед визначений набір алгоритмічних знань. Така система містить як декларативні, так і процедурні компоненти, що разом формують множину поведінок, доступних агенту в процесі виконання. Декларативні (фактографічні) знання подають собою твердження виду «X є Y» або інші форми опису сталих фактів, властивостей і

відношень у предметній області, які не передбачають активних операцій зміни стану. Натомість процедурні знання охоплюють правила, методики та алгоритми, що визначають, яким чином агент може інтерпретувати, трансформувати та застосовувати декларативні відомості для розв’язання поставлених завдань.

У контексті статичного ПОА сукупність цих знань утворює основу його поведінкової моделі, обмеженої рамками попереднього програмування та не здатної до самостійної модифікації під час виконання. Такий агент діє в суворо визначених межах: він не змінює свою структуру, не пересувається між середовищами та не розширює власні функціональні можливості поза тим набором дій, що був закладений розробником. Незважаючи на це, поєднання декларативних і процедурних знань дозволяє статичному агенту ефективно виконувати різноманітні операції, зокрема аналіз даних, прийняття рішень у межах заданих правил і реалізацію алгоритмів, що стосуються конкретних процесів предметної області.

Для комунікації агентів між собою використовують мови зв’язку програмно орієнтованих агентів (ACL). Комунікація є одним із ключових елементів функціонування мультиагентних систем, оскільки саме через обмін повідомленнями агенти координують свої дії та реалізують спільні цілі. Мови агентної взаємодії становлять собою складні формальні конструкції, що включають низку компонентів: синтаксичні правила побудови повідомлень, семантику їхнього змісту, прагматику намірів, а також набір параметрів, таких як ідентифікатор відправника, одержувача або тип комунікаційного акту. До найвідоміших і найбільш поширених мов агентної комунікації належать FIPA-ACL, KQML, ARCOL, COOL та інші специфікації, що застосовуються у різних типах MAS.

Процес взаємодії між агентами не обмежується лише передаванням повідомлення: він також охоплює його інтерпретацію та підтвердження того, що отримана інформація була коректно зрозуміла. Це забезпечує узгодженість дій агентів і зменшує ризики виникнення семантичних непорозумінь у розподіленому середовищі.

Мови ACL забезпечують можливість стандартизованої комунікації між

агентами, незалежно від того, якою мовою програмування вони були реалізовані або в межах якої агентної платформи функціонують. Завдяки цьому агенти різних мультиагентних систем можуть обмінюватися даними та повідомленнями, залишаючись взаємодійними навіть у гетерогенних середовищах.

ACL визначають низку важливих складових: технічні атрибути повідомлення (зокрема відправник, одержувач, групова трансляція, одноранговий канал), тип комунікаційного акту (наприклад, запит, інформування, підтвердження) та мову представлення змісту (зокрема логіку предикатів або інші формальні мови). Окрім цього, стандарт включає системи протоколів, які регламентують очікувану поведінку агентів у відповідь на той чи інший тип повідомлення, наприклад – реагування інформаційним повідомленням на комунікаційний акт запиту.

## 2.2. Проектування програмно-орієнтованих агентів

Термін агентно-орієнтований підхід до програмування та проектування позначає сукупність нових, нетрадиційних концепцій, які, подібно до мультиагентних систем, формують альтернативний погляд на розроблення програмного забезпечення. Цей підхід описує обчислювальну модель, у центрі якої розташований агент, програмний компонент із властивостями, що виходять за межі класичних компонентних або об’єктних структур. Агент трактується як автономна сутність, наділена елементами «ментальності» (переконаннями, цілями, намірами), здатністю до комунікації з іншими агентами та прив’язкою до часових аспектів своєї поведінки. Таким чином, АОПП забезпечує основу для створення програм, які діють у спосіб, ближчий до інтелектуальних систем, ніж до традиційних алгоритмічних структур.

Фреймворки, що належать до сфери агентно-орієнтованої інженерії програмного забезпечення (AOSE), орієнтовані на побудову інструментарію для створення окремих агентів або їхніх груп, які можуть функціонувати як незалежно, так і у складі колективів. У найширшому розумінні агентний фреймворк — це інфраструктура програмної системи, доступна у формі бібліотеки, спеціального

мовного середовища або їх поєднання, яка надає розробнику базові артефакти та механізми, необхідні для формування каркаса MAS: моделі поведінки, комунікаційні протоколи, шаблони планування, засоби керування життєвим циклом агента тощо. Програмний пакет, що забезпечує функціональні можливості для запуску та підтримки мультиагентних систем у реальному середовищі, традиційно називається агентською платформою. Ще ширшим є поняття агентного інструментарію (agent toolkit), який охоплює не лише засоби виконання, а й засоби моделювання, проектування, автоматичної генерації коду, тестування та розгортання, фактично забезпечуючи повний цикл розробки MAS — від аналізу вимог до етапів обслуговування й подальшого розвитку.

Програмний код агента може створюватися за допомогою мов програмування загального призначення, у поєднанні з відповідними бібліотеками, доступними через API фреймворку, або ж за допомогою спеціалізованих агентних мов, орієнтованих на декларативне визначення поведінки й комунікацій. Незважаючи на наявність таких засобів, побудова ПОА залишається складним завданням, оскільки вимагає врахування інтелектуальних, поведінкових, комунікаційних та архітектурних аспектів системи. Для спрощення розробки складних агентних рішень було запропоновано адаптувати принципи об’єктно-орієнтованого підходу, що дозволило зробити агентне програмування більш доступним, структурованим і практичним для широкого кола програмістів.

## 2.3 Проектування методу удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання

Відповідно до визначених низькорівневих вимог з боку розробника, програмна система повинна забезпечувати виконання таких функцій і властивостей:

1. Функції обробки вхідних даних

1.1. Регулярне виконання моніторингу наявності файлу з актуальними

вхідними даними та його автоматичне завантаження з сервера постачальника послуг на сервер рітейлера.

1.2. Реалізацію процесу актуалізації роздрібних цін на основі отриманих вхідних даних.

1.3. Періодичне здійснення прогнозування рівня роздрібних цін із використанням показників середньої ціни, фінансового балансу та часу, що минув від попереднього прогнозування.

2. Робота з базою даних

2.1. Збереження в базі даних інформації про наявність екстремальних значень характеристик потрібних модулів, виявлених у процесі обробки вхідних даних.

3. Продуктивність системи

3.1. Підвищення швидкості оновлення даних у базі даних без залучення додаткових апаратних ресурсів.

4. Розширення функціональності програмного забезпечення

4.1. Забезпечення розширення функціональних можливостей системи без необхідності використання додаткових апаратних засобів.

4.1.1. Реалізацію інтерфейсу інтерактивної взаємодії оператора з програмною системою, а також механізму надсилання оператору запитів для прийняття рішень щодо обробки екстремальних значень вхідних даних і внесення відповідних змін до бази даних.

4.2.1. Автоматизовану взаємодію між ключовими структурними компонентами програмного забезпечення, що виконують різні делеговані завдання, із застосуванням протоколів обміну повідомленнями з метою узгодження порядку використання спільних програмних ресурсів (зокрема бази даних).

4.3.1. Підтримку механізмів динамічного створення та видалення програмних агентів.

5. Перспективи розвитку

5.1. Розширення потенціалу подальшого розвитку програмного забезпечення шляхом інтеграції сучасних технологій, зокрема методів штучного інтелекту.

6. Інтерфейс користувача

6.1. Забезпечення можливості використання користувацького інтерфейсу в режимі командного рядка (Command Line Interface, CLI.

7.1. Підтримку децентралізованого розміщення програмних компонентів користувацького інтерфейсу з можливістю їх подальшого використання на різних електронно-обчислювальних машинах.

8. Підхід до розробки

8.1. Застосування еволюційного прототипування як основного підходу до розроблення програмної системи, що найбільш повно відповідає початковим вимогам проєкту.

9. Формування вихідних даних

9.1. Автоматичну генерацію CSV-файлу з вихідними даними після кожного виконання процесів актуалізації або прогнозування цін.

З огляду на те, що сукупність властивостей, притаманних платформам агентно-орієнтованого програмування, задовольняє 6 із 12 первинних вимог (Рисунок 2.3), було ухвалено рішення застосовувати даний підхід на етапі розробки програмного забезпечення з метою вдосконалення програмної реалізації системи, який представлений на рисунку 2.4.


Рисунок 2.3 – Властивості ООПП та АОПП


Рисунок 2.4 –Метод розробки ПС з підтримкою розширюваності та повторного використання з використанням АОПП

Розроблену концептуальну модель ПС з підтримкою розширюваності та повторного використання з використанням АОПП предметної області нашої програмної системи наведено на рисунку 2.5

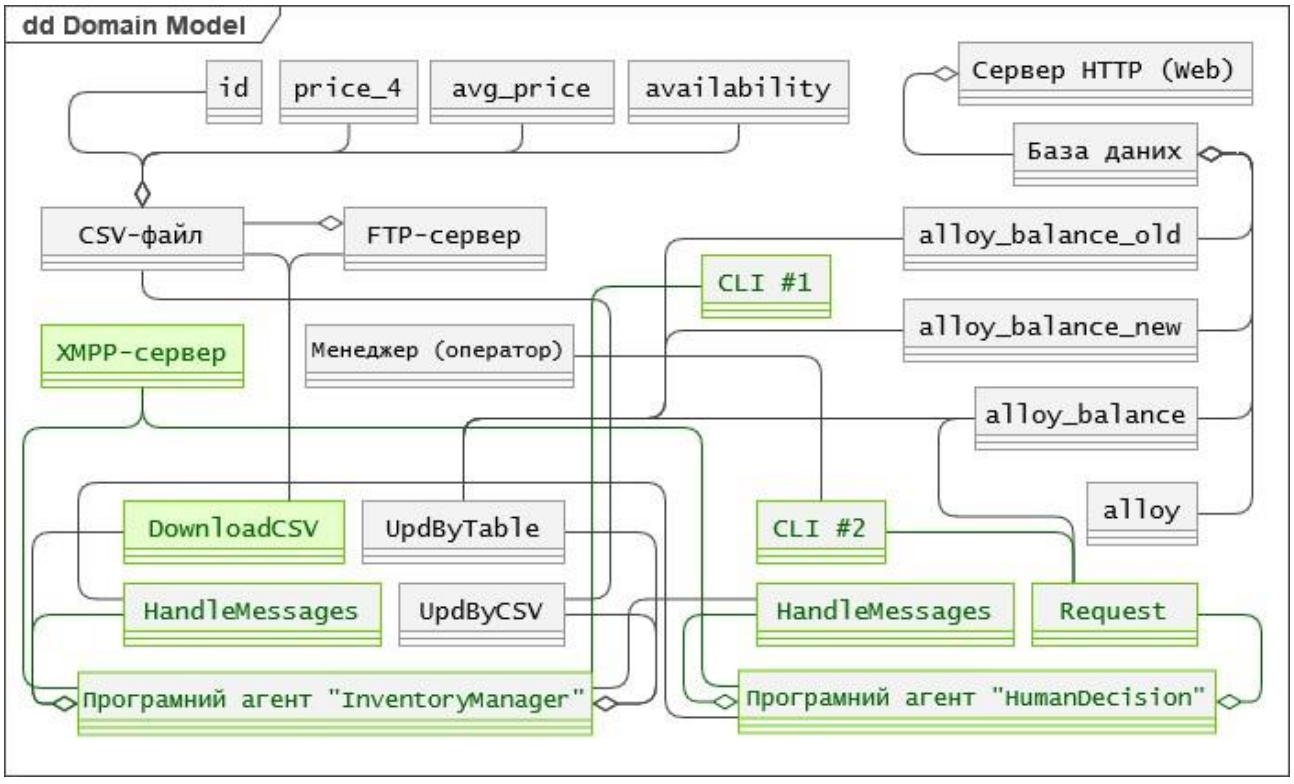



Рисунок 2.5 – Модель предметної області ПС з підтримкою розширюваності та повторного використання з використанням АОПП

У результаті формування та аналізу низькорівневих вимог до програмної

системи було визначено комплекс функціональних і нефункціональних характеристик, необхідних для ефективної обробки, актуалізації та прогнозування роздрібних цін у середовищі електронної комерції. Зокрема, система має забезпечувати автоматизований моніторинг і завантаження вхідних даних, їх подальшу обробку з фіксацією екстремальних значень у базі даних, а також підтримувати високу продуктивність без залучення додаткових апаратних ресурсів. Важливою складовою визначених вимог є орієнтація на розширюваність і гнучкість програмного забезпечення, що проявляється у підтримці динамічного керування програмними агентами, автоматизованої взаємодії між компонентами та можливості використання різних форм інтерфейсу користувача, зокрема CLI та децентралізованих UI-модулів.

Проведений аналіз показав, що агентно-орієнтований підхід проектування найбільш повно відповідає значній частині сформульованих вимог і створює методологічну основу для побудови програмної системи з підтримкою розширюваності та повторного використання. Застосування АОПП у поєднанні з еволюційним прототипуванням дозволяє поетапно розвивати програмний засіб, адаптуючи його до змін вимог і забезпечуючи готовність до інтеграції сучасних технологій, зокрема методів штучного інтелекту. Розроблена концептуальна модель предметної області відображає структурні та функціональні взаємозв’язки ключових компонентів системи й підтверджує доцільність обраного підходу для створення гнучкого, масштабованого та перспективного програмного рішення.

На основі цього було створено метод розробки ПС з підтримкою розширюваності та повторного використання з використанням АОПП тта побудовано модель предметної області ПС з підтримкою розширюваності та повторного використання з використанням АОПП.

## 2.4. Висновки

Загалом у розділі встановлено, що вибір методологічних засобів проєктування програмної системи має ґрунтуватися не лише на порівнянні моделей

життєвого циклу розробки, а передусім на коректному визначенні типу та архітектурних характеристик програмно-орієнтованих агентів, які становлять основу цільового рішення. Проведений огляд агентних архітектур підтвердив доцільність розмежування реактивних, дорадчих і гібридних підходів, де реактивні агенти забезпечують швидку реакцію на зміни середовища, тоді як дорадчі – реалізують більш складну поведінку за рахунок використання ментальних станів, планування та внутрішніх моделей. Поглиблена класифікація агентів через поняття «стану» та цикл «відчуття–аналіз–дія» дозволила уточнити межі моделей типу «стимул–реакція» та визначити, в яких випадках є необхідним збереження інформації між циклами. Також показано, що різноманітність сфер застосування агентів зумовлює широке поле характеристик (автономність, кооперативність, конкурентність тощо) та класифікацій за призначенням (інтерфейсні, Інтернет-, Інтранет- та гетерогенні агенти), а також за категоріями (співробітницькі, інформаційні, реактивні, інтелектуальні, гібридні та ін.). Додатково проаналізовано організаційний вимір агентів – статичні та мобільні, з акцентом на те, що мобільність, попри потенційні переваги для розподілених сценаріїв, супроводжується суттєвими викликами безпеки та ускладненням інфраструктури. З огляду на специфіку предметної області та відсутність вимог до складних мережевих переміщень коду, у роботі обґрунтовано фокус на статичних агентних структурах як більш керованих і прогнозованих.

Важливим результатом розділу є уточнення ролі розміру агента як критерію, що корелює зі складністю його внутрішньої структури, обсягом знань, числом поведінок і потенційною інтелектуальністю. Під інтелектуальністю в даному контексті доцільно розуміти здатність агента до цілеспрямованого міркування і прийняття рішень у ситуаціях, де відсутній повний алгоритм розв’язання задачі, причому ця властивість трактується як динамічна та залежна від реалізованої моделі знань. На основі співвідношення вимог майбутньої системи та складності необхідних операцій зроблено висновок, що розроблювані агенти раціонально віднести до класу рефлексивних статичних інтернет-мікроагентів, здатних ефективно виконувати обробку даних у межах наперед визначеного набору

поведінок. Показано, що статичний ПОА може розглядатися як традиційна програмна система з поєднанням декларативних (фактографічних) і процедурних знань, які формують основу його поведінкової моделі та визначають правила трансформації інформації для виконання делегованих функцій. Окремо підкреслено, що комунікація є центральним механізмом мультиагентних систем, а застосування мов ACL (FIPA-ACL, KQML та інших) забезпечує стандартизований обмін повідомленнями незалежно від мов програмування та платформ, включаючи опис параметрів повідомлень, типів комунікаційних актів і протоколів очікуваних відповідей. У межах розділу також сформовано підґрунтя для переходу до AOSE: агентно-орієнтований підхід проектування розглянуто як обчислювальну модель, де агент виступає центральною композиційною одиницею, а агентні фреймворки та платформи забезпечують інфраструктуру для повного циклу створення MAS.

Ключовим практичним підсумком розділу стало формулювання низькорівневих вимог до програмної системи актуалізації та прогнозування роздрібних цін і їх узгодження з можливостями агентно-орієнтованого підходу проектування. Вимоги охоплюють регулярний моніторинг і завантаження вхідних даних, актуалізацію цін, періодичне прогнозування, фіксацію екстремальних значень у БД, підвищення швидкості оновлення без додаткових апаратних ресурсів, розширення функціональності та підтримку інтерактивної взаємодії оператора, автоматизовану координацію доступу до спільних ресурсів через протоколи повідомлень, а також динамічне створення й видалення агентів. Додатково визначено вимоги до перспективності рішення (готовність до інтеграції сучасних технологій, зокрема ШІ), до інтерфейсної частини (CLI та децентралізація UI-компонентів), до процесу розробки (еволюційне прототипування) і до формування вихідних даних (генерація CSV після актуалізації чи прогнозування). Показано, що сукупність властивостей платформ агентно-орієнтованого програмування задовольняє 6 із 12 первинних вимог, що стало підставою для прийняття рішення про використання АОПП на етапі реалізації з метою вдосконалення програмної архітектури та підвищення розширюваності й повторного використання компонентів.

На цій основі сформовано метод розробки програмної системи з підтримкою розширюваності та повторного використання з використанням АОПП та побудовано концептуальну модель предметної області, яка відображає ключові доменні об’єкти та їх взаємозв’язки. Загалом розділ створює методологічне й концептуальне підґрунтя для подальшого проєктування та реалізації системи, узгоджуючи агентну модель, вимоги до архітектури, механізми взаємодії та підхід еволюційного прототипування як найбільш придатний для поетапного розвитку функціональності.

# 3. ТЕХНОЛОГІЯ РЕАЛІЗАЦІЇ МЕТОДУ УДОСКОНАЛЕНОГО МОДУЛЬНОГО ПРОЄКТУВАННЯ АГЕНТНО-ОРІЄНТОВАНИХ ПРОГРАМНИХ СИСТЕМ З ПІДТРИМКОЮ РОЗШИРЮВАНОСТІ ТА ПОВТОРНОГО ВИКОРИСТАННЯ

## 3.1. Огляд засобів розробки програмно-орієнтованих агентів та агентно-орієнтованої програмної системи

Платформи та середовища розробки програмно-орієнтованих агентів (ПОА) відіграють критично важливу роль у створенні мультиагентних систем, оскільки вони не лише спрощують процес розроблення, а й значною мірою знижують технічну складність етапів розгортання та підтримки таких систем. Завдяки цим платформам розробник отримує попередньо реалізовану інфраструктуру для комунікації агентів, управління їхнім життєвим циклом, взаємодії зі службами та підтримки розподіленого виконання, що значно скорочує час розробки та кількість помилок.

За останні десятиліття було створено або ініційовано створення понад сотні агентних платформ, таких як ZEUS, JADE, agenTool, Cougaar, MadKit, JACK Intelligent Agents та багато інших; кожна з них орієнтована на певний набір функціональних вимог та типів застосувань. Значною мірою це пов’язано з експоненційним зростанням інтересу до розподілених систем, штучного інтелекту та автономних програмних сутностей на межі 1990-2000-х років.

Переважна більшість платформ базується на Java, оскільки ця мова забезпечує гарну переносність, підтримку багатопоточності та багату інфраструктуру бібліотек, а також дозволяє створювати розподілені застосунки за допомогою вбудованих мережевих API.

Однак значна частина раніших платформ, або застаріла, або повністю припинила розвиток через відсутність активного співтовариства, зміну вимог до систем, або заміну сучаснішими фреймворками, і нині лише обмежене коло інструментів підтримується та активно використовується у дослідницькій і

промисловій сферах.

JADE (Java Agent Development Framework) залишається однією з найвідоміших, найбільш усталених та найширше застосовуваних платформ для побудови мультиагентних систем протягом більш ніж двадцяти років. Популярність JADE пояснюється поєднанням зрілої архітектури, дотриманням стандартів FIPA і можливістю створювати повноцінні розподілені системи з опорою на надійний стек технологій Java.

У рамках цієї платформи агенти функціонують у взаємопов'язаних контейнерах, віртуальних середовищах, які поєднані між собою за допомогою внутрішніх транспортних механізмів. Це дозволяє масштабувати систему, переміщувати агентів між вузлами та підтримувати гнучку архітектуру. JADE надає потужний набір сервісів: Discovery Service, Directory Facilitator, Agent Communication Channel, поведінкові моделі, шаблони повідомлень, підтримку ACL-мовлення та інші механізми, які значно спрощують реалізацію розподіленої взаємодії.

Значна частина наукових публікацій, пов'язаних із реалізацією й експериментуванням у сфері MAS, включає JADE як основну платформу, що свідчить про її надійність і стабільність як інструменту досліджень і розробки. Саме завдяки цьому JADE зберігає статус де-факто стандарту в академічних і практичних проєктах, пов'язаних з агентними системами.

SPADE (Smart Python Agent Development Environment) це сучасна, відкрита платформа для створення мультиагентних систем, розроблена мовою Python і розповсюджувана за ліцензією MIT, що робить її доступним і гнучким інструментом як для дослідників, так і для промислових проєктів.

На відміну від Java-орієнтованих платформ, SPADE базується на протоколі миттєвих повідомлень XMPP, що забезпечує ефективну та стандартизовану систему комунікації між агентами, включаючи шифрування, маршрутизацію і підтримку розподілених серверів.

Однією з найбільших переваг SPADE є її інтеграційна сумісність із сучасною Python-екосистемою: розробники можуть безпосередньо використовувати

бібліотеки для машинного навчання (TensorFlow, PyTorch, Scikit-learn), засоби обробки даних (Pandas, NumPy), а також інтегрувати агентні системи у веб-додатки через Django або Flask. Це робить платформу особливо привабливою для побудови інтелектуальних агентів, що використовують алгоритми класифікації, прогнозування, reinforcement learning або поведінкове моделювання.

Простота синтаксису Python, кросплатформеність та активна спільнота сприяють швидкому розвитку SPADE і роблять її одним із найперспективніших інструментів для створення гнучких, адаптивних і навчальних агентних систем у сучасних умовах.

## 3.2 Аналіз підходів до проектування програмних систем з підтримкою розширюваності та повторного використання

Технологія проектування програмного забезпечення являє собою впорядковану сукупність методів, підходів, інструментів і організаційних засобів, які визначають та регламентують процес створення програмного продукту на всіх етапах його існування. Вона охоплює не лише безпосередню реалізацію програмного коду, а й аналіз вимог, проектування архітектури, тестування, впровадження, супровід і подальший розвиток програмної системи. Структура та склад процесів життєвого циклу програмного забезпечення формалізовані в міжнародному стандарті ISO/IEC/IEEE 12207:2017, який визначає загальноприйняту модель організації робіт у сфері інженерії програмного забезпечення.

Згідно з вимогами зазначеного стандарту, у практиці розроблення ПЗ виділяють низку базових технологій або моделей життєвого циклу, що відображають різні підходи до планування та виконання робіт. До найбільш поширених з них належать каскадна модель, спіральна модель, ітеративна модель, V-модель, гнучкі методології (Agile), а також модель прототипування.

Кожен із цих підходів має власні особливості організації процесу розробки, рівень формалізації, вимоги до документації та ступінь гнучкості щодо змін вимог.

Варто зазначити, що в науковій та практичній літературі окремі елементи наведеного переліку іноді розглядають як повноцінні моделі життєвого циклу, а інші, як технології або методології, що доповнюють базові моделі розроблення програмного забезпечення. Порівняльний аналіз переваг і недоліків зазначених підходів наведено в таблиці 3.1, що дозволяє обґрунтовано обрати оптимальну модель нашої проектованої системи  залежно від специфіки проєкту, складності системи та вимог замовника.

Таблиця 3.1 – Порівняльна таблиця підходів до розробки ПЗ

| Назва | Переваги | Недоліки |
|---|---|---|
| Каскадна | Чітко структуровані та послідовні етапи розробки, що полегшує планування, контроль виконання робіт і документування результатів. Завдяки цьому вона добре підходить для проєктів зі стабільними та детально визначеними вимогами. Високий рівень формалізації процесів дозволяє легко відстежувати прогрес розробки та відповідність результатів початковому технічному завданню. Це спрощує управління проєктом і взаємодію між замовником та розробниками. | Малогнучка до змін, оскільки повернення до попередніх етапів розробки потребує значних часових і ресурсних витрат. Це робить її неефективною для проєктів, у яких вимоги можуть змінюватись. Робоча версія програмного забезпечення з’являється лише на пізніх етапах життєвого циклу, що ускладнює раннє виявлення помилок і оцінку відповідності продукту очікуванням користувачів. |
| Спіральна | Поєднує ітеративний підхід із систематичним аналізом ризиків, що дозволяє своєчасно виявляти та зменшувати критичні проблеми проєкту. Це робить її ефективною для розробки складних і високоризикових програмних систем.<br>Наявність багаторазових ітерацій дає змогу поступово уточнювати вимоги та вдосконалювати архітектуру на основі зворотного зв’язку. Завдяки цьому підвищується якість кінцевого продукту та його відповідність потребам замовника. | Реалізація спіральної моделі потребує значних управлінських зусиль і високої кваліфікації команди, особливо у сфері аналізу та оцінювання ризиків. Це може ускладнювати її застосування в невеликих або обмежених за ресурсами проєктах. Високий рівень складності планування та контролю ітерацій може призводити до збільшення вартості й тривалості розробки. За відсутності чітко визначених критеріїв завершення ітерацій існує ризик затягування проєкту. |

| | | |
|---|---|---|
| Ітеративна | Дозволяє поетапно розробляти програмне забезпечення з регулярним отриманням робочих версій продукту. Це дає змогу швидко отримувати зворотний зв’язок від користувачів і вчасно коригувати вимоги. Завдяки поділу розробки на окремі ітерації спрощується виявлення та виправлення помилок на ранніх етапах. Такий підхід знижує ризики та підвищує загальну якість програмного продукту. | Вимагає чіткої організації процесу та постійної координації між учасниками команди, що може ускладнювати управління проєктом. За відсутності належного планування ітерації можуть стати неузгодженими. Часті зміни вимог і повторне переглядання рішень можуть призводити до збільшення витрат часу та ресурсів. Існує ризик фрагментації архітектури, якщо її цілісність не контролюється протягом усього циклу розробки. |
| V-модель | Забезпечує чіткий зв’язок між етапами розробки та відповідними етапами тестування, що дозволяє планувати перевірку якості ще на ранніх стадіях життєвого циклу. Це підвищує надійність програмного забезпечення та зменшує кількість критичних помилок на завершальних етапах. Високий рівень формалізації процесів сприяє кращій керованості проєкту та прозорості виконання робіт. Такий підхід особливо ефективний для систем із жорсткими вимогами до якості та безпеки. | є малогнучкою щодо змін вимог, оскільки коригування на пізніх етапах потребують повернення до попередніх фаз і додаткових витрат. Це ускладнює її застосування в динамічних проєктах із мінливими вимогами. Значний обсяг документації та сувора послідовність етапів можуть збільшувати тривалість і вартість розробки. Відсутність швидких робочих прототипів обмежує можливості ранньої оцінки функціональності системи користувачами. |
| | Забезпечує високу гнучкість розробки, оскільки дозволяє оперативно вносити зміни до вимог на будь-якому етапі проєкту. Завдяки коротким ітераціям і постійному зворотному зв’язку підвищується відповідність програмного продукту реальним потребам користувачів. Часте постачання робочих інкрементів ПЗ дає змогу рано виявляти помилки та поступово підвищувати якість системи. Крім того, тісна співпраця між замовником і командою розробки сприяє швидшому ухваленню рішень. | Потребує високої залученості замовника та добре скоординованої команди, що не завжди можливо у великих або розподілених проєктах. За відсутності активної комунікації ефективність підходу суттєво знижується. Менша формалізація документації може ускладнювати супровід і масштабування програмного забезпечення в довгостроковій перспективі. Крім того, складно точно прогнозувати терміни та вартість розробки на початкових етапах проєкту. |

| | | |
|---|---|---|
| Прототипування | Дозволяє швидко створити робочу версію програмного продукту для демонстрації основних функцій і перевірки вимог. Це дає змогу отримати ранній зворотний зв’язок від користувачів та уточнити їхні очікування . Завдяки поетапному вдосконаленню прототипу зменшується ризик неправильного розуміння вимог і підвищується відповідність кінцевого продукту потребам замовника. Такий підхід особливо ефективний для проєктів з нечітко сформульованими або змінними вимогами | Існує ризик, що прототип буде помилково сприйнятий як майже готовий продукт, що може призвести до зниження якості архітектурних рішень. У результаті тимчасові рішення можуть перейти у фінальну версію системи. Часті ітерації створення та переробки прототипів можуть збільшувати витрати часу й ресурсів. Крім того, за відсутності чітких критеріїв завершення процес прототипування може затягуватися |

Поряд з цими моделями проектування існують і інші моделі, зокрема можна виділити DevOps, Lean, Scrum, Kanban та RAD, Extreme Programming, FDD, SAFe тощо. Scrum є гнучкою методологією управління розробкою ПЗ, що базується на коротких ітераціях (спринтах), чітко визначених ролях та регулярних зустрічах. Вона орієнтована на швидку адаптацію до змін вимог і тісну взаємодію між командою та замовником. DevOps-підхід об’єднує процеси розробки та експлуатації ПЗ з метою прискорення постачання продукту та підвищення його стабільності. Він базується на автоматизації, безперервній інтеграції та постійному моніторингу систем. Kanban візуальний підхід до управління процесом розробки, який фокусується на безперервному потоці завдань і обмеженні кількості робіт у процесі. Він дозволяє підвищити прозорість виконання робіт і оптимізувати використання ресурсів без жорстких ітераційKanban — це візуальний підхід до управління процесом розробки, який фокусується на безперервному потоці завдань і обмеженні кількості робіт у процесі. Він дозволяє підвищити прозорість виконання робіт і оптимізувати використання ресурсів без жорстких ітерацій . Lean Software Development (Lean-підхід) ґрунтується на принципах ощадливого виробництва та спрямований на усунення втрат у процесі розробки ПЗ. Основна увага приділяється створенню цінності для користувача, скороченню зайвих дій і постійному вдосконаленню процесів. RAD (Rapid Application Development)

орієнтований на швидке створення програмних рішень за рахунок активного використання прототипування, повторного використання компонентів і мінімізації формальної документації. Цей підхід ефективний для проєктів з обмеженими термінами та відносно простою архітектурою. Extreme Programming (XP) є гнучкою методологією, що робить акцент на високій якості коду, частому тестуванні та тісній взаємодії з замовником. Серед ключових практик — парне програмування, безперервна інтеграція та розробка, керована тестами. Feature-Driven Development (FDD) орієнтований на розробку програмного забезпечення через реалізацію чітко визначених функціональних можливостей. Проєкт поділяється на невеликі, добре формалізовані функції, що полегшує планування та контроль виконання. SAFe (Scaled Agile Framework) SAFe призначений для масштабування Agile-підходів у великих організаціях та складних проєктах. Він поєднує принципи Agile, Lean та системного мислення для координації роботи багатьох команд.

Слід зазначити, що в багатьох наукових джерелах, які використовувались для формування таблиці 3.1, перелік недоліків моделі прототипування часто не наведено, тоді як у інших літературних джерелах вони подаються з різним рівнем деталізації та часто залежать від конкретного типу прототипу, який використовується в процесі розробки. Водночас переваги цього підходу є достатньо вагомими для обґрунтування його застосування та загалом відповідають вимогам і очікуванням замовника. Саме це зумовлює необхідність більш ґрунтовного й детального аналізу підходу прототипування в межах подальшого дослідження.

Отже, узагальнюючи проведений аналіз, можна дійти висновку, що каскадна модель проєктування не забезпечує необхідної гнучкості щодо внесення змін до програмних процесів на ранніх етапах розробки, що суперечить вимогам замовника. Ітеративний підхід, у свою чергу, є недостатньо ефективним для реалізації невеликих за обсягом проєктів. Застосування спіральної моделі, яка поєднує риси каскадної та ітеративної, передбачає обов’язковий аналіз ризиків, що може призводити до додаткових часових витрат у процесі розроблення. Аналогічно, V-модель, як розширення каскадної парадигми, характеризується

обмеженою адаптивністю та не відповідає вимогам створення програмного забезпечення в умовах постійних змін, а також не усуває проблеми невизначеності щодо термінів виконання та вартості проєкту. Набільш прийнятною для проектування програмних систем з підтримкою розширюваності та повторного використання в нашому випадку є модель протипування.

## 3.3 Проєктування архітектури програмних систем з підтримкою розширюваності та повторного використання на основі підходу прототипування

Використання прототипів є важливим елементом і водночас одним із найбільш поширених підходів до проєктування програмних систем, який передбачає часткову реалізацію майбутнього продукту та застосовується з метою уточнення системних вимог і покращення їх сприйняття всіма учасниками процесу розробки на ранніх етапах життєвого циклу програмного забезпечення. Класичним призначенням прототипу є його створення з орієнтацією на максимальне повторне використання програмного коду в остаточній версії системи, що прийде на зміну прототипу, при цьому стандартизованою основою для його побудови слугують специфікації та вимоги замовника.

Характерною особливістю прототипування є концентрація уваги розробників на візуальній складовій програмної системи, що дає змогу наочно продемонструвати майбутній вигляд і поведінку застосунку. У випадках, коли основні ризики пов’язані з упровадженням нових технологічних рішень, доцільно створювати архітектурні прототипи, тоді як за наявності ризиків, пов’язаних із користувацьким інтерфейсом, застосовуються прототипи вимог. Прототипування дозволяє розв’язати низку завдань, пов’язаних із виявленням, демонстрацією та оцінюванням ключових характеристик та вимог до програмної системи, зокрема структури й вимог до даних, функціональних і нефункціональних вимог, операційних характеристик та показників ефективності, організаційних вимог до майбутньої системи,  (Рисунок 3.2).

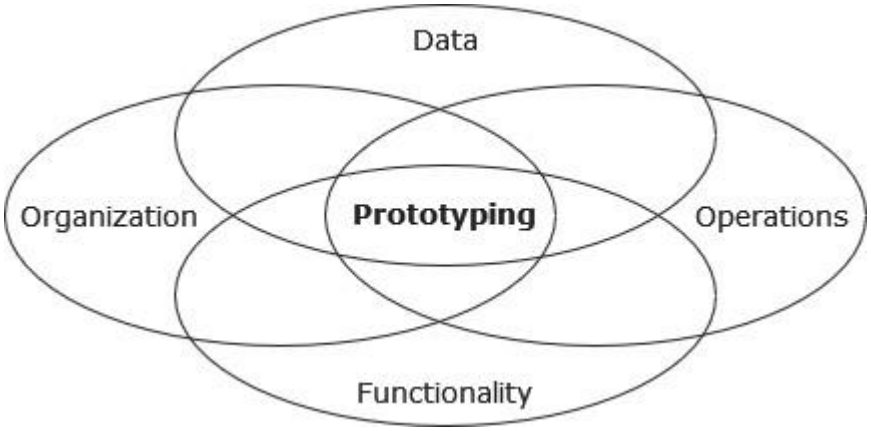



Рисунок 3.2 – Вимоги до ПС вирішувані в проектування методом прототипування

При застосуванні метода протипувування для програмних систем з підтримкою розширюваності та повторного використання на основі підходу прототипування слід дотримуватись ключових факторів та враховувати ризики підходу прототипування, як навдено узагальнено в таблиці 3.2 відповідно до проведеного аналізу наукових досліджень та джерел.

Таблиця 3.2 – Ключові фактори та ризики прототипування

| **Фактори** | **Ризики** |
|---|---|
| Висока ефективність під час розроблення програмних систем з підтримкою розширюваності | Існує ризик того, що прототип може бути трансформований у некоректно спроєктовану програмну систему внаслідок недостатнього попереднього аналізу та надмірної зосередженості розробників на самому прототипі |
| Зменшення витрат часу та фінансових ресурсів | Існує ризик виникнення непорозумінь із замовником через відсутність у нього чіткого розуміння відмінностей між прототипом, що відображає переважно користувацький інтерфейс, та повноцінним кінцевим програмним продуктом. |
| Участь кінцевих користувачів у формуванні та уточненні вимог | Існує ризик порушення встановлених термінів розробки у випадку нераціонального використання часу, оскільки процес створення програмного забезпечення має відбуватися у стислі строки. |
| Отримання більш гнучкого та з більшим функціоналом прототипу | Існує ймовірність реалізації помилкових архітектурних рішень у разі надмірної зосередженості процесу розробки на прототипі. |

Ризики подано в умовній формі, тому за зміни вихідних умов вони підлягають усуненню. Перший ризик, наведений у таблиці 3.2, пропонується нейтралізувати шляхом формування повного та однозначного опису вимог,

орієнтованих на кінцевий програмний продукт, а не на його прототип. Усунення другого ризику доцільно здійснити за рахунок чіткого розмежування між узагальненими вимогами до остаточної версії системи та вимогами, що висуваються до прототипів на різних етапах проєктування й розроблення. Третій ризик пропонується мінімізувати шляхом застосування підходу, який відповідає очікуванням замовника щодо темпів розробки, зокрема забезпечує можливість оперативного створення користувацького інтерфейсу, що сприятиме підвищенню швидкості розробки як прототипів різних версій, так і системи на наступних етапах її еволюції. Останній ризик частково суперечить ідеї використання еволюційного прототипування, що передбачає послідовну модифікацію прототипів, тому його усунення пропонується досягти шляхом концентрації уваги на вимогах до кінцевого продукту, а не до проміжних прототипних рішень.

У науковій літературі виділяють кілька основних різновидів прототипів, зокрема одноразові, еволюційні та операційні, а також горизонтальні й вертикальні. Одноразові прототипи використовуються виключно для початкової перевірки ідей і не залучаються до подальшої розробки програмної системи, тоді як еволюційні прототипи слугують основою для поетапного розвитку та вдосконалення ПС. Горизонтальні прототипи призначені для демонстрації загальної структури системи або її окремих підсистем і відображають переважно інтерфейсні та концептуальні характеристики без повної реалізації функціоналу. Натомість вертикальні прототипи забезпечують глибшу реалізацію окремих функціональних компонентів, залишаючи нереалізованими другорядні властивості. Горизонтальні одноразові прототипи, на відміну від вертикальних еволюційних, які зазвичай максимально наближені до фінальної версії програмної системи, можуть відповідати лише мінімальному набору вимог, зосереджуючись передусім на зручності використання. Водночас створення вертикальних еволюційних прототипів потребує ґрунтовного аналізу вимог і застосування повноцінних підходів до проєктування, характерних для розробки завершених програмних систем.

Характерні риси швидкого прототипування значною мірою збігаються з

особливостями застосування одноразових прототипів і не повною мірою відповідають вимогам замовника, оскільки не передбачають можливості подальшого використання створеного прототипу в наступних версіях програмного продукту. Інкрементне та екстремальне прототипування мають подібні принципи організації та можуть застосовуватися для створення завершених систем, однак у контексті вимог до програмної системи, що розробляється, вони фактично орієнтовані на окрему підсистему та не враховують потреби подальшого поділу на незалежні підсистеми з можливістю їх окремої реалізації. Горизонтальне прототипування також не задовольняє поставлені вимоги, оскільки зосереджується переважно на демонстрації інтерфейсних аспектів і не забезпечує повноцінного відображення функціональних можливостей системи. З огляду на це, як основний підхід до проєктування та розроблення програмної ситеми було обрано саме вертикальне еволюційне прототипування, яке широко застосовується компаніями в межах підходів до створення програмних систем із підтримкою розширюваності та повторного використання компонентів. Такий вибір обумовлений тим, що зазначений підхід найбільш повно відповідає початковим вимогам проєкту. Його застосування дозволяє поетапно формувати повноцінну програмну систему з мінімальними часовими витратами, отримуючи на кожному етапі функціональну робочу версію продукту.

Водночас використання агентно-орієнтованого підходу проектування опосередковано сприяє вдосконаленню процесу прототипування шляхом усунення умов виникнення одного з ключових недоліків, пов’язаного з дотриманням строків розробки. Зокрема, це досягається за рахунок скорочення витрат часу, а отже й трудомісткості, на створення та модифікацію версій прототипу завдяки менш складному процесу реалізації користувацького інтерфейсу порівняно з традиційними веб-додатками. Крім того, застосування АОПП узгоджується з вимогами вертикального еволюційного прототипування, яке передбачає використання тих самих підходів до проєктування, що й при розробленні повноцінного програмного забезпечення. Таким чином, зменшення складності процесу створення інтерфейсу, зокрема, та впровадження агентно-орієнтованого

підходу загалом дозволяє підвищити співвідношення показників корисності до витрат у процесі еволюції прототипу до мінімально життєздатного продукту програмної системи та його подальших повнофункціональних версій.

Підвищення швидкості розробки ПС з підтримкою розширюваності та повторного використання та розширення функціоналу під час застосування прототипування до удосконалення та після нього подано на рисунку 3.3.

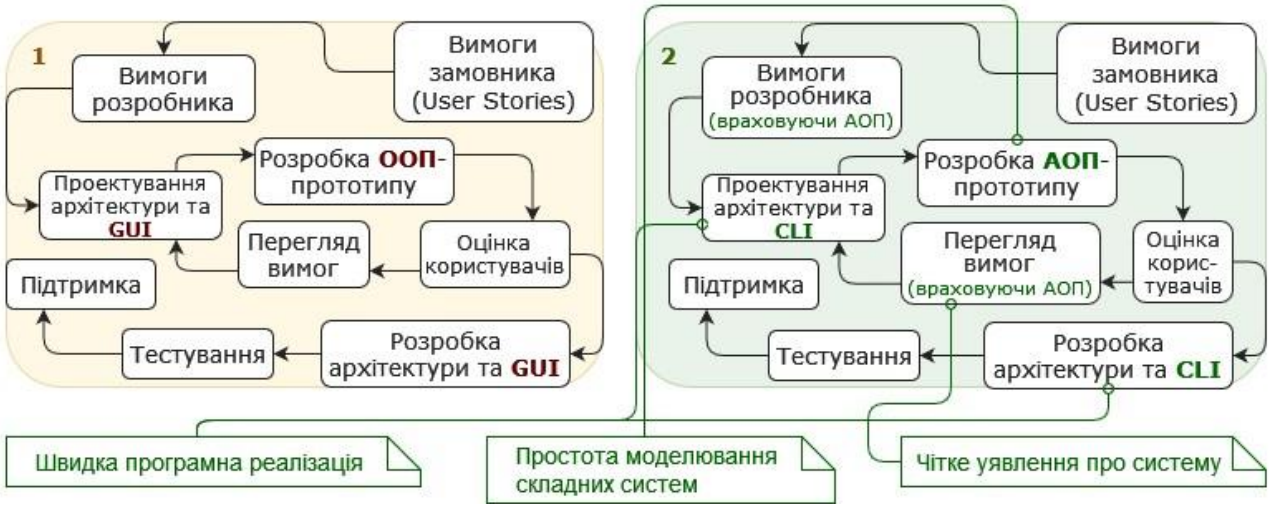



Рисунок 3.3 – Порівняння класичної методики проектування та методики з підтримкою розширюваності та повторного використання

## 3.4Проектування модулів та структури програмної системи

У результаті виконання етапу детального проєктування окремих модулів і програмної системи в цілому було сформовано статичну модель її структури, яку представлено на рисунку 3.4 у вигляді діаграми класів. Зазначена діаграма відображає основні структурні елементи системи та їхні взаємозв’язки, забезпечуючи наочне уявлення про внутрішню організацію програмного забезпечення. У межах діаграми зафіксовано всі ключові статичні відношення між класами, зокрема відношення узагальнення, асоціації та залежності, що дозволяє проаналізувати ієрархію, зв’язність і рівень декомпозиції системи.

На діаграмі, для кожного класу на діаграмі наведено перелік його властивостей (атрибутів), які характеризують стан відповідних об’єктів, а також

методів (операцій), що визначають доступну функціональність і поведінку класів у межах системи. Такий підхід дає змогу чітко простежити розподіл відповідальностей між компонентами та оцінити повноту їх функціонального наповнення. Абстрактні класи разом із притаманними їм властивостями і методами спеціально виокремлено курсивним шрифтом, що підкреслює їхню роль як базових елементів для подальшого наслідування та розширення. Це, у свою чергу, сприяє кращому розумінню механізмів повторного використання, розширюваності та еволюції архітектури програмної системи.

З метою підвищення зручності супроводу програмної моделі, а також створення передумов для її подальшого розвитку і повторного використання окремих програмних компонентів, у проєкті було застосовано принцип функціонального групування класів. Таке групування реалізовано шляхом об’єднання класів у логічно пов’язані пакети, кожен з яких відповідає за окремий аспект функціонування програмної системи, що наочно продемонстровано на рисунку 3.5.

Використання пакетної структури дозволяє чітко відокремити різні функціональні підсистеми, зменшити зв’язність між класами, а також підвищити читабельність і зрозумілість архітектури програмного забезпечення. Крім того, такий підхід спрощує модифікацію й розширення системи, оскільки зміни в межах одного пакета мінімально впливають на інші частини моделі. Угрупування класів за функціональною ознакою також створює сприятливі умови для повторного використання окремих пакетів або їх складових у майбутніх проєктах, що відповідає принципам модульності, масштабованості та підтримуваності програмних систем. Додатково пакетна організація сприяє ефективнішому розподілу відповідальності між компонентами та полегшує командну розробку, оскільки різні пакети можуть розвиватися паралельно. Така структуризація також підвищує керованість життєвого циклу програмного забезпечення, спрощуючи тестування, супровід і подальшу інтеграцію нових функціональних можливостей.

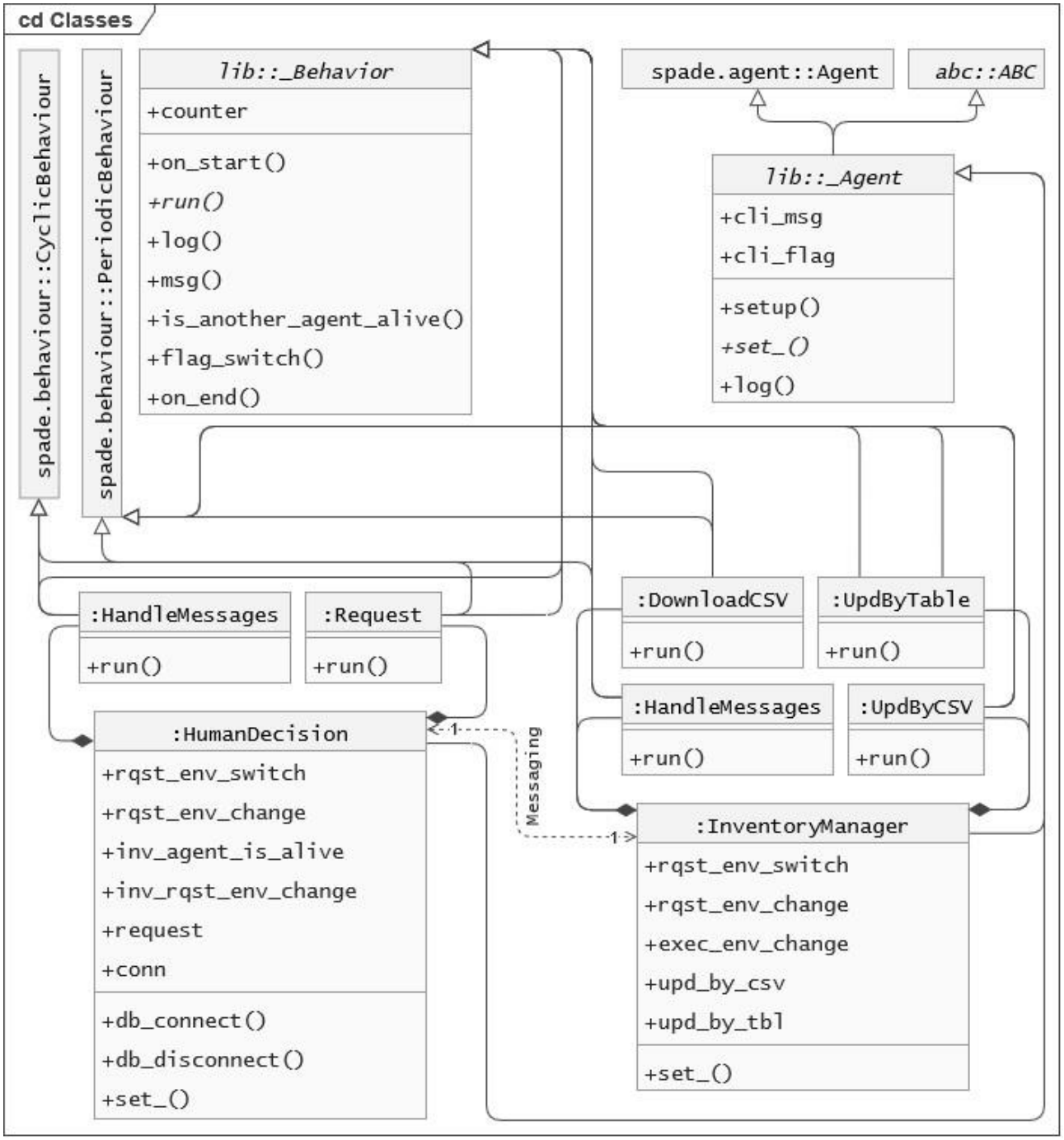



Рисунок 3.4 – Діаграма основних класів агентно-орієнтованої програмної системи з підтримкою розширюваності та повторного використання

З метою визначення оптимальних способів розгортання програмних пакетів і окремих класів у середовищі операційної системи, а також для формування цілісного уявлення про архітектуру розроблюваної програмної системи, було виконано об’єктно-орієнтовану декомпозицію. У процесі цієї декомпозиції система була поетапно розкладена на сукупність логічно завершених підсистем, кожна з яких виконує визначений набір функцій і відповідає за окремі аспекти загальної

поведінки програмного забезпечення.

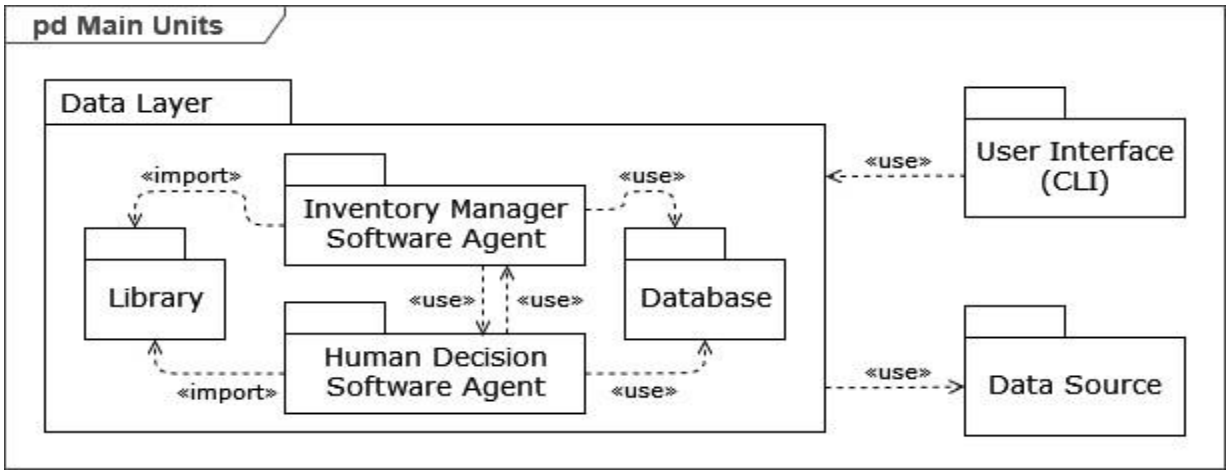



Рисунок 3.5 – Діаграма пакетів програмної системи

Результатом такого підходу стало формування переліку підсистем і встановлення зв’язків між ними, що дозволяє чітко окреслити межі відповідальності окремих компонентів та їхню взаємодію. Для наочного відображення структури системи, взаємозалежностей між підсистемами та принципів їх інтеграції було використано структурну діаграму пакетів компонентів. Застосування цієї діаграми сприяє кращому розумінню архітектурних рішень, полегшує аналіз можливих змін у системі та створює основу для подальшого масштабування, супроводу і повторного використання компонентів у межах як поточного, так і майбутніх програмних проєктів .

## 3.5Висновки

У межах даного розділу було виконано детальне проєктування агентно-орієнтованої програмної системи, що дозволило сформувати цілісне уявлення про її статичну структуру, внутрішні зв’язки та принципи організації. Побудова діаграми класів надала можливість формалізувати архітектуру програмного забезпечення, визначити ключові структурні елементи та відобразити основні типи статичних відношень між ними, зокрема узагальнення, асоціації та залежності. Це,

у свою чергу, забезпечило прозорість і логічну впорядкованість архітектурних рішень, а також створило основу для аналізу рівня зв’язності та декомпозиції системи. Чітке визначення атрибутів і методів кожного класу дало змогу простежити розподіл відповідальностей між компонентами, оцінити повноту реалізації функціональних вимог і переконатися у відповідності моделі принципам об’єктно-орієнтованого проєктування.

Особливу роль у сформованій статичній моделі відіграють абстрактні класи, які визначають базові концепції предметної області та слугують фундаментом для подальшого розширення й наслідування. Їх виокремлення підкреслює орієнтацію архітектури на повторне використання та еволюційний розвиток програмної системи без суттєвих змін її ядра. Такий підхід дозволяє зменшити вартість подальших модифікацій, спростити адаптацію системи до нових вимог і підвищити гнучкість архітектури в довгостроковій перспективі. У результаті детального проєктування було досягнуто балансу між абстракцією та конкретною реалізацією, що є важливою передумовою створення масштабованих і підтримуваних програмних рішень.

Застосування принципу функціонального групування класів у пакети стало важливим кроком у напрямі підвищення зрозумілості та керованості програмної моделі. Пакетна структура дозволила логічно відокремити окремі підсистеми, зменшити міжмодульну зв’язність і локалізувати зміни в межах конкретних функціональних областей. Це істотно спрощує супровід програмного забезпечення, полегшує пошук і усунення помилок, а також створює сприятливі умови для командної розробки, коли різні розробники або групи можуть працювати над окремими пакетами незалежно один від одного. Крім того, така організація коду відповідає сучасним вимогам до модульності та підтримуваності програмних систем.

Проведена об’єктно-орієнтована декомпозиція програмної системи до рівня підсистем і їхніх взаємозв’язків дозволила сформувати чітке уявлення про архітектуру розгортання та принципи інтеграції компонентів у середовищі операційної системи. Використання структурної діаграми пакетів компонентів

забезпечило наочне відображення меж відповідальності підсистем, а також механізмів їхньої взаємодії. Це є особливо важливим для подальших етапів реалізації та тестування, оскільки спрощує аналіз впливу змін на систему в цілому та знижує ризик порушення її цілісності.

Узагальнюючи результати розділу, можна стверджувати, що виконане детальне проєктування створило надійну архітектурну основу для подальшої реалізації агентно-орієнтованої програмної системи з підтримкою розширюваності та повторного використання. Сформована статична модель, пакетна структура та декомпозиція на підсистеми забезпечують відповідність сучасним принципам програмної інженерії, таким як модульність, масштабованість, повторне використання та супроводжуваність. Отримані результати можуть бути ефективно використані не лише в межах даного проєкту, а й як основа для розробки подібних програмних систем у майбутньому, що підтверджує доцільність і практичну значущість обраних архітектурних рішень.

# 4. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

## 4.1. Програмна реалізація агентно-орієнтованої програмної системи з підтримкою розширюваності та повторного використання

Хід розробки програмного застосунку доцільно розпочати з проєктування бази даних, оскільки саме вона визначає структуру зберігання та подальшої обробки інформації. Концептуальну модель бази даних було сформовано з використанням UML-діаграми, що дало змогу наочно відобразити основні сутності предметної області та їх взаємозв’язки. Центральною незалежною сутністю моделі є Alloy, яка містить два базові атрибути та може необов’язково включати три залежні сутності. Кожна з цих залежних сутностей розширює основну додатковими відомостями щодо відповідних модулів і містить від дев’яти до десяти атрибутів. Необов’язковий характер включення залежних сутностей до незалежної відображено на діаграмі за допомогою незабарвлених стрілок, що вказує на можливість їх відсутності в конкретному екземплярі даних.

На основі побудованої концептуальної моделі була сформована логічна структура даних, які використовуються програмно-орієнтованими агентами. Її реалізовано у вигляді фізичної реляційної ER-моделі в середовищі MySQL, що забезпечує практичну придатність моделі до використання у реальній інформаційній системі. Відповідна фізична модель бази даних наведена на рисунку 4.1 і відображає таблиці, їхні ключові поля та зв’язки між ними.

Ординальність і кардинальність зв’язків, тобто мінімальна та максимальна кількість можливих відповідностей між записами таблиць, позначені сполучними лініями між ключовими полями id. Для основної таблиці alloy поле id виступає первинним ключем, тоді як для пов’язаних таблиць (зокрема balance) воно використовується як зовнішній ключ. Характер зв’язків між основною та залежними таблицями є однаковим: записи таблиці balance можуть бути пов’язані з таблицею alloy лише в одному екземплярі, тоді як для кожного запису таблиці alloy у відповідних залежних таблицях допускається наявність нуля або одного

пов’язаного запису (кардинальність 0..1). Така структура забезпечує цілісність даних і відповідає логіці предметної області, що моделюється.

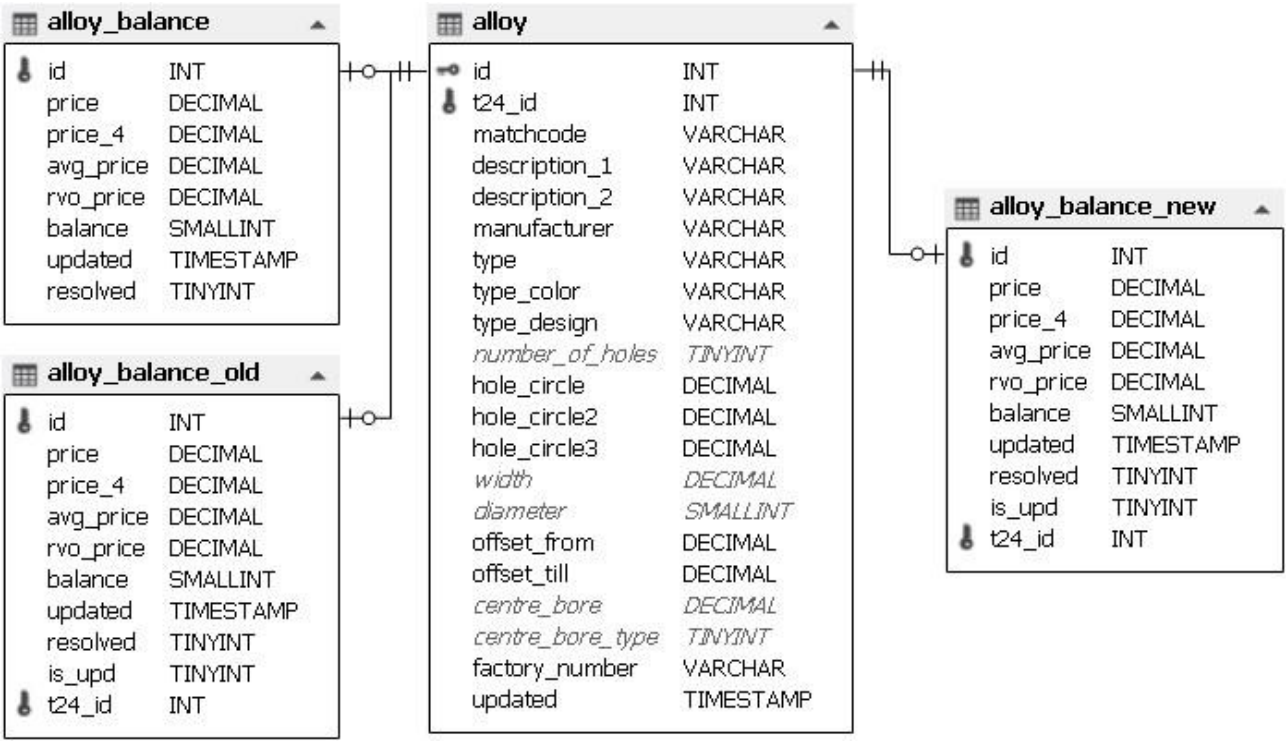



Рисунок 4.1 – Фізична структура бази даних

Фрагмент БД відображає фізичну ER-модель бази даних, розроблену для програмної системи актуалізації та прогнозування роздрібних цін у межах агентно-орієнтованого підходу, що був обґрунтований у попередніх розділах роботи. Дана модель реалізує структуру зберігання основних доменних даних, з якими взаємодіють програмно-орієнтовані агенти під час виконання процесів моніторингу, актуалізації, прогнозування та аналізу цінових показників.

Центральною таблицею моделі є таблиця alloy, яка відповідає незалежній сутності предметної області та описує базовий об’єкт — товар (сплав / продукт), для якого здійснюється обробка цінових даних. Вона містить первинний ключ id, а також набір атрибутів, що характеризують товар з точки зору ідентифікації, класифікації та виробничих характеристик. До таких атрибутів належать ідентифікатор зовнішньої системи t24_id, код матеріалу matcode, текстові описи (description_1, description_2), виробник (manufacturer), типові ознаки (type, type_color, type_design), а також геометричні та технологічні параметри (кількість

отворів, їх діаметри, ширина, офсети, параметри центрального отвору тощо). Наявність поля updated дозволяє відстежувати момент останнього оновлення запису, що є важливим для агентних сценаріїв контролю актуальності даних.

Для зберігання даних про модулі використано окремі залежні таблиці: alloy_balance, alloy_balance_old та alloy_balance_new. Таке рішення відповідає вимогам до розширюваності, історизації та аналізу динаміки даних, сформульованим у попередніх розділах. Усі ці таблиці пов’язані з основною таблицею alloy через зовнішній ключ id, який водночас є первинним ключем у відповідних таблицях балансу. Це реалізує зв’язок типу 1 : 0..1, за якого для кожного товару може існувати не більше одного актуального запису балансових даних у кожній з таблиць.

Таблиця alloy_balance призначена для зберігання поточних розрахункових значень, які безпосередньо використовуються програмно-орієнтованими агентами в процесах актуалізації. Вона містить атрибути price, price_4, avg_price, rvo_price, що відображають різні моделі розрахунку ціни, поле balance, яке характеризує фінансовий або товарний баланс, а також службові поля updated і resolved, які застосовуються агентами для контролю стану обробки та прийняття рішень.

Таблиця alloy_balance_old виконує роль історичного сховища, у якому зберігаються попередні значення попередніх рішень. Додаткове поле is_upd дозволяє фіксувати факт участі запису в процесах оновлення або міграції даних. Наявність історичних даних є критично важливою для реалізації агентних алгоритмів прогнозування, які використовують часові ряди та аналіз попередніх станів системи.

Таблиця alloy_balance_new використовується для тимчасового зберігання результатів прогнозування або попередніх розрахунків, отриманих програмно-орієнтованими агентами. Атрибут is_upd у цій таблиці сигналізує про готовність даних до подальшої інтеграції в основну модель або до підтвердження оператором через інтерфейс взаємодії, що відповідає вимогам до інтерактивної участі людини у критичних сценаріях.

Таким чином, представлена ER-модель відображає агентно-орієнтовану

логіку обробки даних, де основна сутність зберігає стабільні доменні характеристики, а змінні, часово-залежні показники винесені в окремі залежні таблиці. Такий підхід знижує зв’язність, спрощує масштабування, забезпечує підтримку еволюційного прототипування та створює передумови для повторного використання модулів обробки даних у майбутніх версіях програмної системи на основі підходу АОПП.

На підставі отриманих результатів порівняльного аналізу, виконаного з використанням метрики LOC (Lines of Code), було проведено оцінювання обсягу програмного коду, створеного із застосуванням традиційних підходів до розробки програмного забезпечення та з використанням платформи агентно-орієнтованого програмування. Аналіз показав суттєву різницю між цими підходами, що дозволило сформулювати обґрунтований висновок про переваги запропонованого нами методу.

Зокрема встановлено, що програмний код, реалізований за допомогою агентно-орієнтованої платформи, має менший обсяг у порівнянні з кодом, створеним традиційними засобами. Це означає, що запропонований підхід потребує менших витрат часу та обчислювальних ресурсів на етапах реалізації, тестування та супроводу програмної системи. Скорочення кількості рядків коду досягається, зокрема, за рахунок вбудованих механізмів автоматичної реєстрації агентів, стандартизованих засобів обміну повідомленнями, повторного використання готових шаблонів поведінки та інфраструктурних компонентів, які не потребують ручної реалізації.

Таким чином, застосування платформи агентно-орієнтованого програмування сприяє підвищенню продуктивності розробки, зменшенню складності програмної реалізації та зниженню ризику помилок, пов’язаних із надмірною кількістю коду. Це підтверджує доцільність використання запропонованого підходу для створення масштабованих, розширюваних і підтримуваних програмних систем у межах досліджуваної предметної області Рисунок 3.2).

```python
import spade
from spade.agent import Agent
from spade.behaviour import OneShotBehaviour
from spade.message import Message

class SenderAgent(Agent):
    class InformBehav(OneShotBehaviour):
        async def run(self):
            msg = Message(to='receiver@xmpp_server')     # Створення екземпляру повідомлення
            msg.body = 'Hello World'                     # Контент повідомлення
            try:
                await self.send(msg) # Відправка повідомлення
                print('Message sent successfully!')
            except Exception as e:
                print(f'Failed to send message: {e}')
...
class ReceiverAgent(Agent): # Отримувач
    class RecvBehav(OneShotBehaviour):
    async def run(self):
        msg = await self.receive(timeout=10) # Очікування 10 с на повідомлення
            if msg:
                print(f'Message received with content: {msg.body}')
            else:
                print('Did not received any message after 10 seconds.')
```

Рисунок 4.2 – Програмна реалізація комунікації ПОА ПС за допомогою підходу АОПП.

Користуацький інтерфейс представлено на рисунках 4.3–4.4. Зокрема, на рисунку 4.31 відображено запуск ПОА «Подійний менеджер» та його поведінку з урахуванням даних з таблиць бази даних.

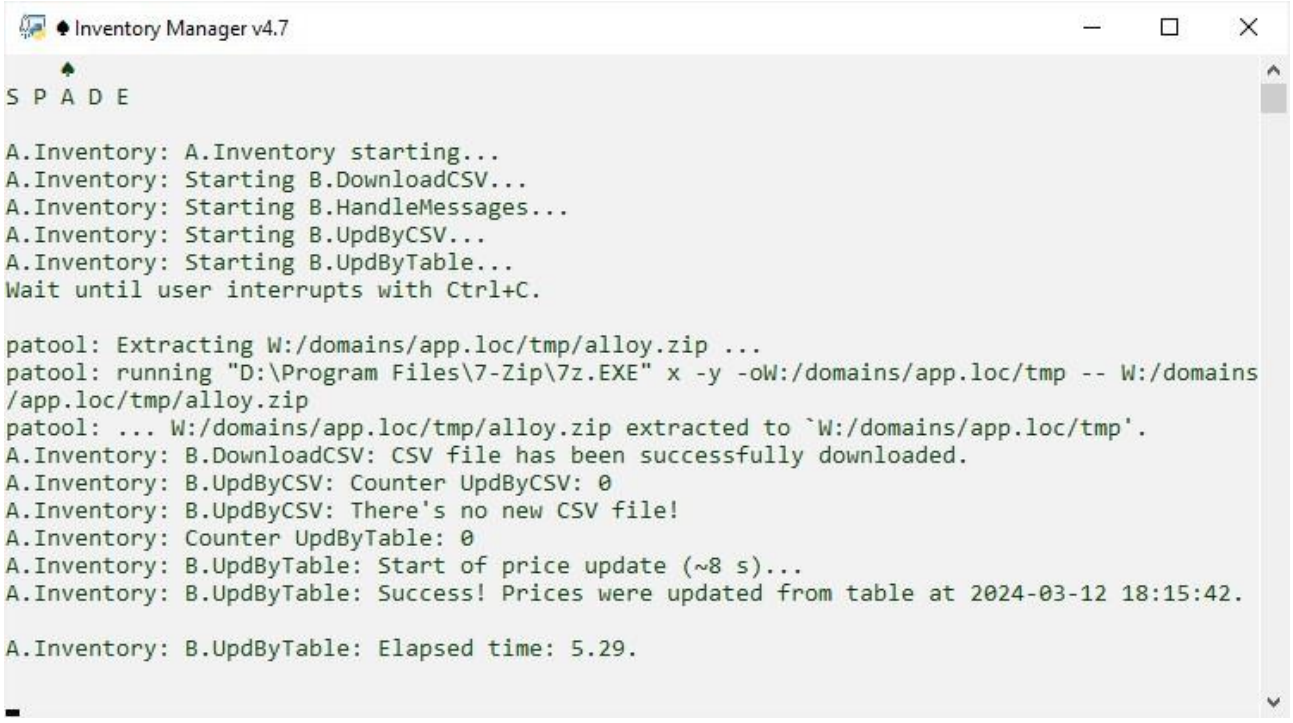



Рисунок 4.3 – Подійний менеджер ПОА

Перед виконанням даної поведінки програмна система ініціює окремий процес, що відповідає за завантаження та розархівування CSV-файлу з вхідними даними, отриманого із зовнішнього джерела. Після успішного завершення цього етапу автоматично запускається процедура актуалізації відповідних модулів програмної системи на основі проаналізованої та підготовленої інформації.

Водночас зазначена поведінка була коректно завершена в автоматичному режимі у зв’язку з тим, що в процесі перевірки не було виявлено нового CSV-файлу для обробки. Даний факт встановлено шляхом порівняння часових міток наявних файлів, що дозволило системі визначити відсутність оновлень і уникнути зайвого виконання обчислювальних операцій.

На рисунку 4.4 подакно користувацький інтерфейс запуску ПОА «Рішення оператора», його процеси зміну станів та обміну повідомленнями з ПАО «Подійний менеджер». Зокрема показано запит стану активності ПОА « Подійний менеджер» та запит на зміну відповідного  середовища.

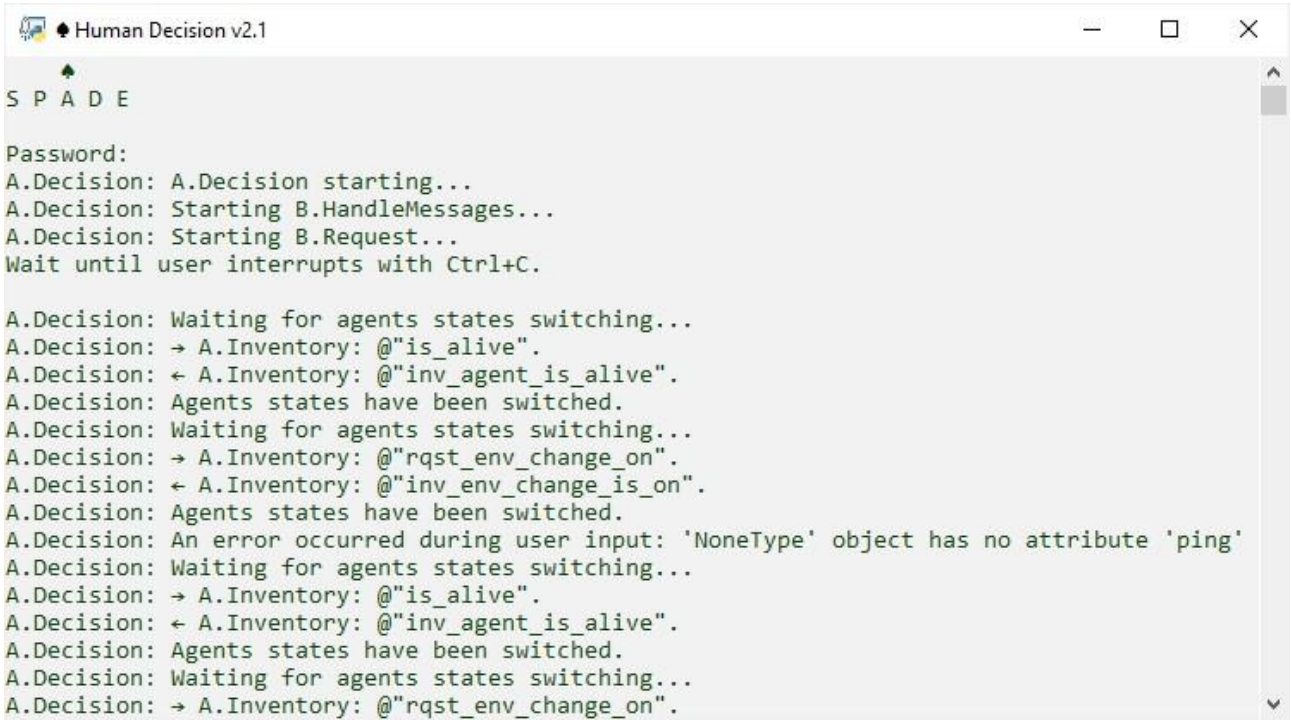



Рисунок 4.4 –ПОА «Рішення оператора»

На рисунку 4.5 подано систему логування результатів поведінок ПОА «Подійний менеджер» та «Рішення оператора», у тому числі також їх обмін повідомленнями.

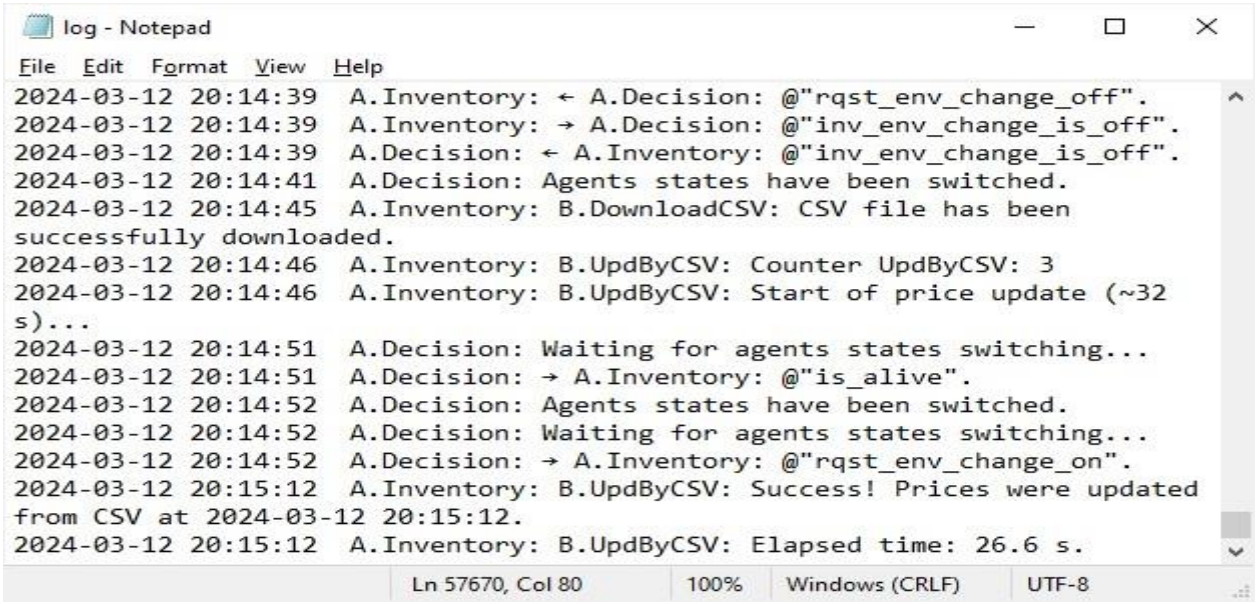



Рисунок 4.5 – Логування поведінок ПА

З метою повного відображення фізичної реалізації програмного застосунку та наочного представлення його апаратної організації, розподіл елементів програмної системи між обчислювальними вузлами подано у вигляді структурної діаграми розгортання (діаграми розміщення). На цій діаграмі показано вкладені виконувані компоненти, а також відношення та канали взаємодії між окремими вузлами системи (Рисунок 4.6).

У процесі аналізу архітектурної структури та механізмів взаємодії між компонентами (модулями) програмної системи було визначено рівень їх зв’язності, що відображено на рисунку у вигляді залежностей між окремими елементами. Подальше застосування агентно-орієнтованого підходу дозволило суттєво зменшити ці залежності, що наочно продемонстровано на рисунку. Абстрактна модель, представлена на цьому рисунку, ілюструє можливість автономного функціонування програмних агентів на різних пристроях та їхню гнучку взаємодію через серверне середовище без залучення додаткових допоміжних процесів. У

результаті досягається зниження рівня зв’язності між компонентами системи та, відповідно, підвищення її модульності, гнучкості й масштабованості (Рисунок 4.6)

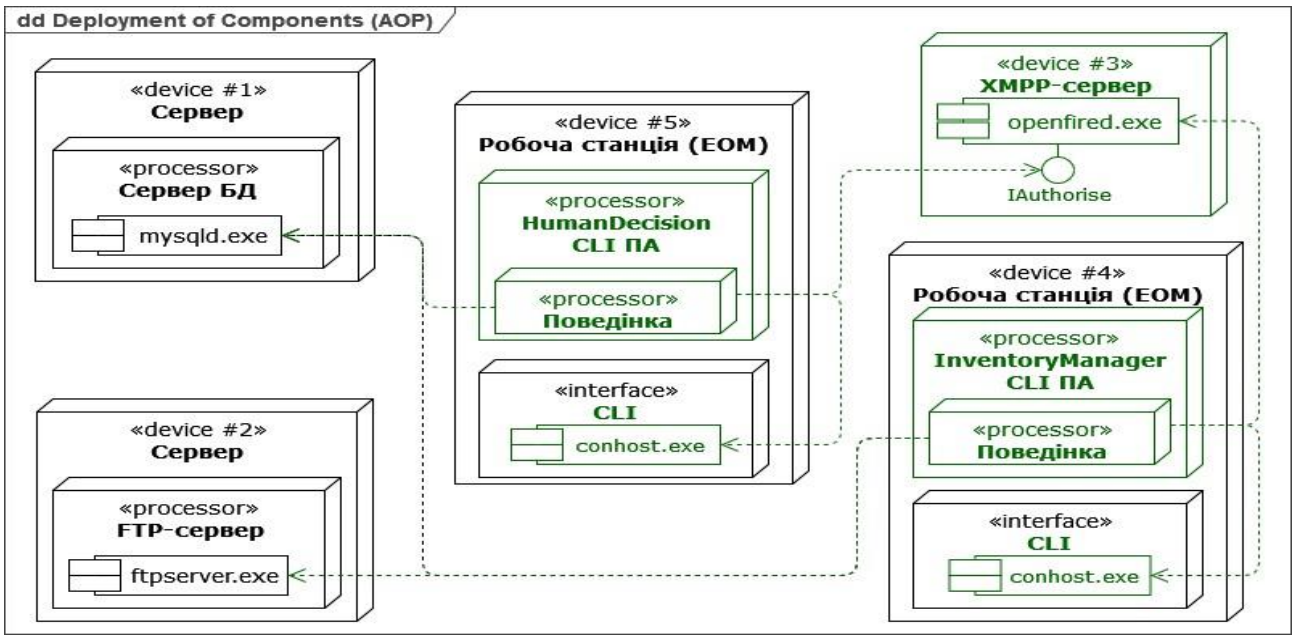



Рисунок 4.6– Діаграма розгортання програмної системи

Підсумовуючи, можемо зробити висновок, що у ході розробки програмного застосунку було обґрунтовано доцільність початку проєктування з побудови бази даних як фундаментального елемента, що визначає структуру зберігання, обробки та аналізу інформації у предметній області. Побудована концептуальна UML-модель та її подальша реалізація у вигляді фізичної реляційної ER-моделі в середовищі MySQL забезпечили логічну узгодженість даних, підтримку цілісності та відповідність агентно-орієнтованій логіці системи. Запропонована структура з центральною незалежною сутністю та набором залежних таблиць дозволила чітко розмежувати стабільні доменні характеристики та змінні, часово-залежні показники, що є критично важливим для процесів актуалізації, прогнозування та історичного аналізу цін. Такий підхід сприяє зменшенню зв’язності між компонентами, підвищенню розширюваності моделі даних і створює передумови для еволюційного розвитку системи.

Проведений порівняльний аналіз за метрикою LOC підтвердив ефективність застосування агентно-орієнтованого підходу до програмування, який дозволив

суттєво скоротити обсяг програмного коду та зменшити витрати ресурсів на реалізацію, тестування й супровід програмної системи. Реалізація взаємодії між програмно-орієнтованими агентами, наявність подійного менеджера, механізмів прийняття рішень оператором, системи логування та підтримка автоматизованого обміну повідомленнями забезпечили гнучку, модульну й масштабовану архітектуру застосунку. Побудована діаграма розгортання додатково підтвердила зниження зв’язності між компонентами та можливість автономної роботи агентів у розподіленому середовищі. Загалом отримані результати доводять, що поєднання агентно-орієнтованого підходу з еволюційним прототипуванням є доцільним і ефективним рішенням для створення складних інформаційних систем у сфері проектування програмних систем за підходом АОПП.

## 4.2 Валідація, тестування та експериментальні результати

У процесі вибору інструментів для оцінювання продуктивності програмного забезпечення було проведено детальний порівняльний аналіз існуючих рішень та засобів конкурентного тестування. За результатами цього аналізу як найбільш ефективний і доцільний інструмент для виконання завдань тестування продуктивності було обрано Apache JMeter. Даний вибір зумовлений широкими функціональними можливостями інструменту, його зрілістю, активною підтримкою спільноти, а також здатністю інтегруватися з іншими засобами автоматизації тестування.

Apache JMeter є програмним забезпеченням з відкритим вихідним кодом, реалізованим мовою програмування Java, і широко використовується у практиці тестування як універсальний автоматизований інструмент для виконання навантажувального, стресового та продуктивнісного тестування різних типів програмних систем. Він дозволяє моделювати одночасну роботу великої кількості користувачів, аналізувати реакцію системи під різними рівнями навантаження та виявляти вузькі місця в архітектурі програмного забезпечення. У поєднанні з платформою BlazeMeter Apache JMeter забезпечує повноцінну підтримку

автоматизації тестування продуктивності, включаючи розподілене тестування, генерацію звітів та інтеграцію з CI/CD-пайплайнами.

Оскільки навантажувальне та стрес-тестування є складовими методами в межах загального підходу до оцінювання продуктивності програмного забезпечення, Apache JMeter застосовувався для реалізації всіх зазначених типів тестів. Крім того, даний інструмент використовувався для виконання завдань моніторингу продуктивності застосунків у рамках підходу Application Performance Monitoring (APM). Основною метою використання APM-рішень є своєчасне виявлення, аналіз і локалізація потенційних проблем продуктивності програмного забезпечення ще до того, як вони почнуть негативно впливати на досвід кінцевих користувачів або стабільність роботи системи.

Для збору та аналізу даних щодо використання апаратних ресурсів серверного середовища, зокрема показників завантаження центрального процесора та обсягу споживаної оперативної пам’яті, окрім Apache JMeter було залучено спеціалізований програмно-орієнтований агент PerfMon Server Agent, також відомий як JP@GC агент (JMeter Plugins at Google Code). Зазначений агент реалізований мовою програмування Java та базується на бібліотеці SIGAR (System Information Gatherer And Reporter), що забезпечує доступ до системної інформації на рівні операційної системи. PerfMon Server Agent дозволяє отримувати серверні метрики в режимі реального часу, що є критично важливим для комплексної оцінки продуктивності та коректної інтерпретації результатів навантажувального тестування.

Перед початком виконання основних тестових сценаріїв було проведено димове тестування, метою якого була перевірка працездатності базового функціоналу програмної системи, коректності розгортання середовища та готовності інфраструктури до проведення більш інтенсивних випробувань. Проведення димового тестування дозволило знизити ризик отримання некоректних результатів на наступних етапах експериментального дослідження. Узагальнений перелік типів тестування, використаних інструментів, а також їх основні характеристики та призначення наведено в таблиці 4.1.

Таблиця 4.1 – План для тестування ПС

| № | Тип тесту | Засоби для тестування | Опис |
|---|---|---|---|
| | Продуктивності ПОА | | Різниця між умовами спрацювання ПОА (так/ні). |
| | Продуктивності ПС | | Порівняння CLI та WUI в аспекті CAP. |
| | Навантаження | | З’єднання XMPP (підключення ПА до серверу). |
| | Продуктивності(Навантаження) | | Тестуання для порівнняння навантаження на сервер середовища XMPP із застосуванням різних типів поведінки ПОА (поза процесом оновлення модулів ПС та під оновлення). |
| | Стрес-тест | | Перевірка навантаження при підключенні великої кількості ПОА. |

Ключовим моментом в нашому тестуванні є тестування продуктивності програмної системи.

Метою цього тесту (Таблиця 4.1, пункт 1) є отримання детальних експериментальних даних за ключовими метриками продуктивності XMPP-сервера та програмного агента, який у межах дослідження розглядається як окремий системний процес. Збір відповідних показників здійснювався із застосуванням плагіна Dummy Sampler, що використовується у випадках, коли відсутня потреба у виконанні реальних команд або формуванні тестових запитів до системи. Такий підхід дозволяє зосередитися виключно на вимірюванні споживання ресурсів і поведінки системи в стабільному режимі без впливу додаткових операцій.

Параметри «Primary Metrics» плагіна JPAGC для центрального процесора та оперативної пам’яті (Рисунок 4.7) було залишено у значеннях за замовчуванням – в процентах для центрального процесора та  резидентна для оперативної памяті відповідно. Вибір цих параметрів обумовлений їх універсальністю та відповідністю завданню оцінювання фактичного навантаження на апаратні ресурси. Зазначені

метрики відображають частку використання обчислювальних ресурсів програмним забезпеченням у разі застосування механізму ідентифікації конкретного процесу, що дозволяє отримати більш точні та інтерпретовані результати.

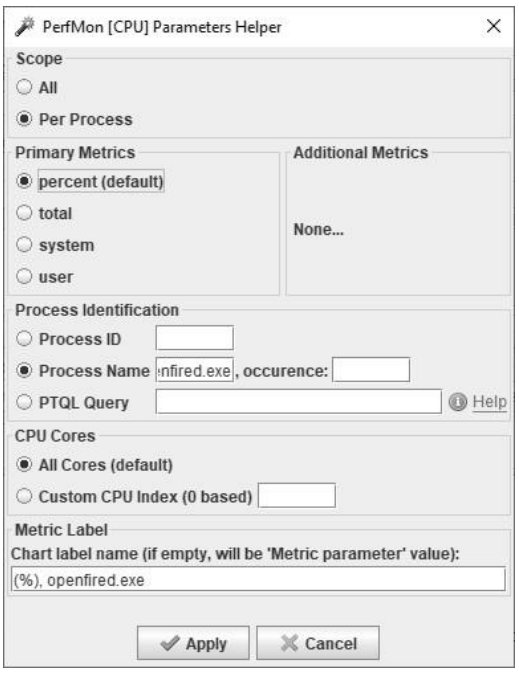



Рисунок 4.7 – Вибір методу моніторингу процесу

У налаштуваннях плагіна JPAGC кожній окремій метриці в розділі Servers to Monitor – Parameters Helper, доступ до якого здійснюється через елемент керування у кінці рядка відповідної метрики (Рисунок 4.8), передбачено вибір області збору статистичних даних. Зокрема, користувач може обрати режим моніторингу всіх процесів, що використовують відповідний апаратний компонент (далі — «юніт», наприклад CPU), активувавши перемикач «All», або обмежити збір показників окремим процесом за допомогою режиму «Per Process». З урахуванням необхідності визначення впливу конкретного програмного процесу на відповідний апаратний юніт було прийнято рішення використовувати саме другий варіант, який забезпечує більш точну локалізацію навантаження.

У цьому ж вікні налаштувань, у полі Process Identification, реалізовано можливість вибору способу ідентифікації процесу – за унікальним ідентифікатором процесу «Process ID», або за його назвою «Process Name» (Рисунок 4.7). Варто

зазначити, що окремі типи процесів, зокрема програмні компоненти, реалізовані мовою Python, у середовищі операційної системи відображаються під узагальненою назвою (наприклад, Py.exe). Така особливість ускладнює коректний моніторинг у випадках, коли одночасно запущено декілька ідентичних процесів, оскільки назва процесу не дозволяє однозначно їх розрізняти. У зв'язку з цим для ідентифікації було обрано використання PID, що забезпечує точність моніторингу та є більш доцільним з практичної точки зору, оскільки усуває необхідність додаткових дій із призначення або уточнення імен процесів.

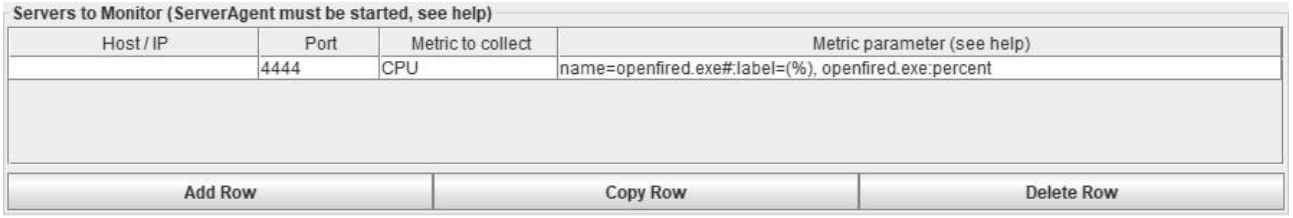



Рисунок 4.8– Налаштування JPAGC для метрик

Ідентифікацію системного процесу доцільніше здійснювати за його назвою у випадках, коли вона є унікальною, оскільки, на відміну від ідентифікатора процесу (PID), який змінюється при кожному запуску програмного забезпечення, назва процесу залишається сталою. У параметрах налаштування плагіна JP@GC відповідні значення необхідно вказувати на основі даних зі стовпців «Name» та «PID» вкладки Details диспетчера завдань операційної системи (рисунок 4.9, 4.10). Зокрема, процес Openfire Launcher відповідає виконуваному файлу openfire, тоді як Openfire (XMPP) Server представлений процесом openfired. Оскільки агенти SPADE здійснюють підключення безпосередньо до XMPP-сервера, збір метрик продуктивності виконувався саме для цього процесу.

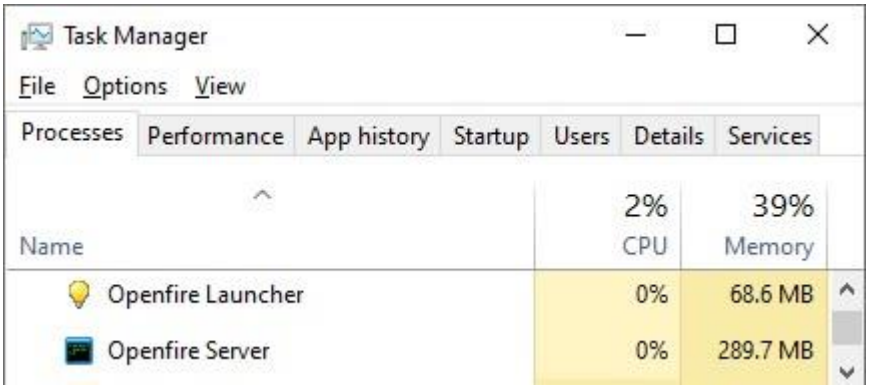



Рисунок 4.9 – Завантаженість процесів програми Диспетчер завдань

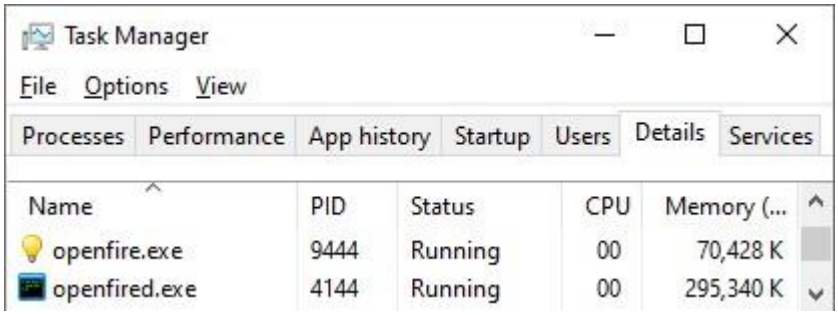



Рисунок 4.10 – Датальний показ процесів програми Диспетчер завадань

За результатами виконання функції оновлення модулів (рис. 4.11) за допомогою плагіна Debug Sampler визначено середнє значення дорівнює приблизно 91,56. Враховуючи, що мінімально необхідний час для цієї операції становить більше 2 с, було вирішено збирати дані в межах цього інтервалу часу (завжди з початку процесу оновлення; таким чином забезпечуючи нормалізацію даних для їх подальшого аналізу).

Проведено порівняння (за середніми значеннями швидкості процесу оновлення модулів) трьох варіантів архітектури ПС, що відрізняються розташуванням АПМ: на стороні веб-сервера з викликом через cron (тестовий запуск через браузер показано на рисунку 4.11); з тим же розташуванням, але з викликом через ПА SPADE (рис. 4.12, 4.13; дану архітектуру використано для більшості тестів); АПМ агрегований ПОА CLIPS (рис. 4.11). Швидкість виконання програмного коду в другому випадку приблизно дорівнює швидкості в першому, що очікувано, враховуючи незмінність розташування АПМ. Даний варіант архітектури дозволяє швидко інтегрувати ПОА CLIPS до веб-проекту (протягом ~ 1 хв). Швидкість виконання програмного коду в третьому випадку становить $S_T$=2,17 с.

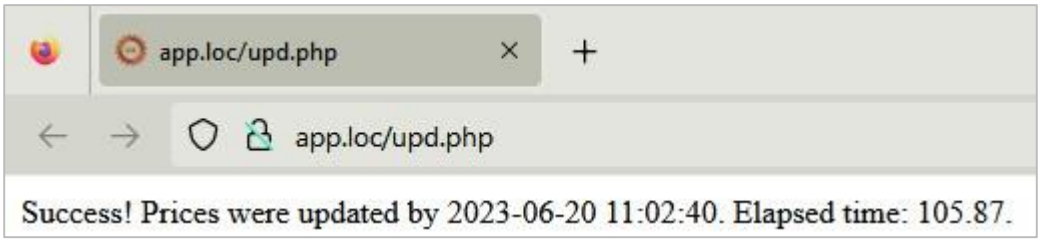



Рисунок 4.11 – Інформаційне повідомлення модулів, яке відображається в результаті запиту через веб-браузер


Рисунок 4.12 – Запуск програмно орієнтованого агента

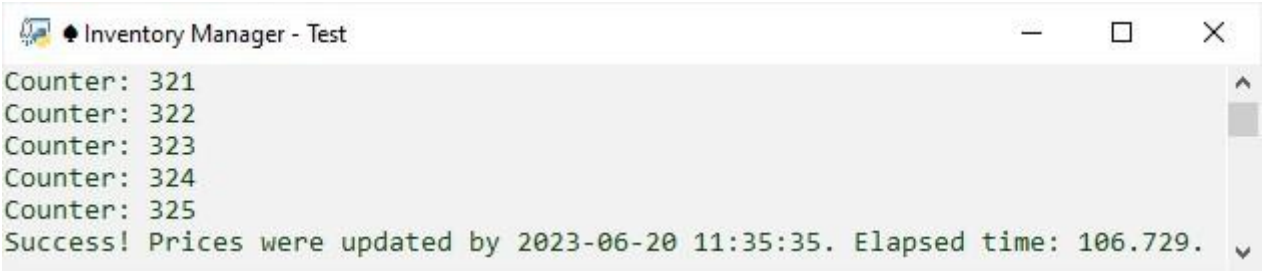



Рисунок 4.13 – Успішне оновлення модулів  за допомогою ПОА

На основі зібраних даних (Рисунок  4.11) розраховано середнє значення використовуваних ЦП ресурсів дорівнює 0,18 %, для ОЗП — 78,23 МБ. Пскільки XMPP не використовує певні апаратні ресурси персонального комп’ютера конкретного користувача, то  для визначення ефективності нами враховувалися лише та ресурси, які використовує наш ПОА (в режимі командної стрічки: центральний процесор  – 0,6 %,  оперативна пам'ять  – 62,42 Mb (Рисунок 4.11).

Нами розраховано ефективність кількості споживання апартних ресурсів при роботі ПОА CLIPS в режимі командної стрічки порівняно з веб-інтерфесом: для цетрального процесора  це  (0,18 – 0,42) / 0,18 × 100 % = –13 % (+0,04 %), для оперативної пам’яті — (78,23 – 62,43) / 98,22 × 100 % = 11,58 % (–11,43 Mb).

Слід зазначити, що в наведеному прикладі застосовувався інтерфейс AWUI, який не містив підключеного програмного коду бібліотек. У випадку використання більш складного графічного інтерфейсу, реалізованого із залученням сучасних фреймворків, можна очікувати зростання ефективності застосування програмно-орієнтованих агентів CLIPS у режимі командного рядка порівняно з їх використанням через графічний інтерфейс користувача

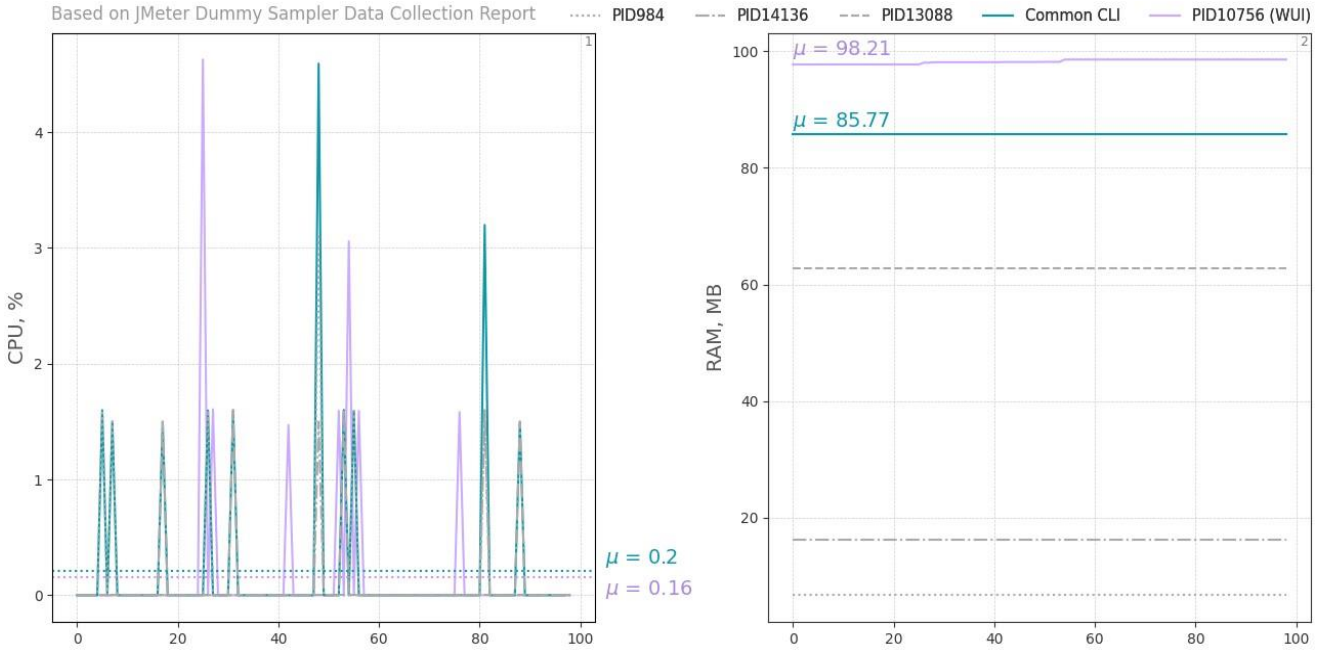



Рисунок 4.10 – Графік порівняння усіх даних ПОА за метриками центрального прооцесора та оперативної памяті

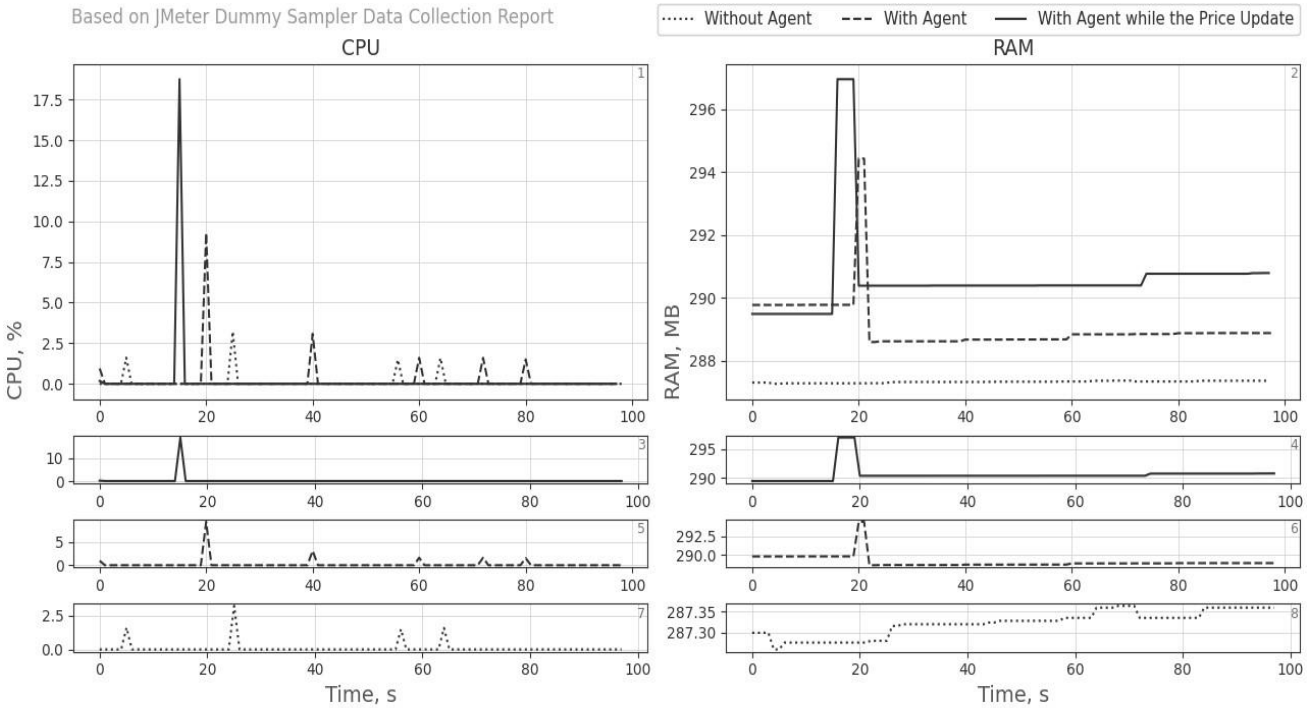



Рисунок 4.11 – Показники споживання апаратних засобів процесу для сервера XMPP зібраних Debug Sampler

## 4.3 Стрес-тестування сервера

Стрес-тестування для запуску та самої роботи сервера XMPP, а також хначної кількості програмно-орієнтованих агентів, з відправленням різноманітних запитів сервера проводилося 7 раз, тривалістю 60 с для кожної ітерації, з двома визначеними налаштуваннями: обмеження частоти запуску програмно-орієнтованих агентів ($F_s$ = 2 Гц, де F –частота процесора, а 2 – запуски а також без нього. Результати тестування сервера наведено в таблиці 4.3 та на рисунку 4.12 і рисунку 4.13.

Таблиця 4.3 – Результати тестування сервера

| № | Кількість запусків додатку | | | Результат |
|---|---|---|---|---|
| | Всього | Успішні | Неуспішні | |
| | | | | Спад на тестувальнику |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | Спад тестувальника сервері |
| | | | | Стабільна робота ПС. |

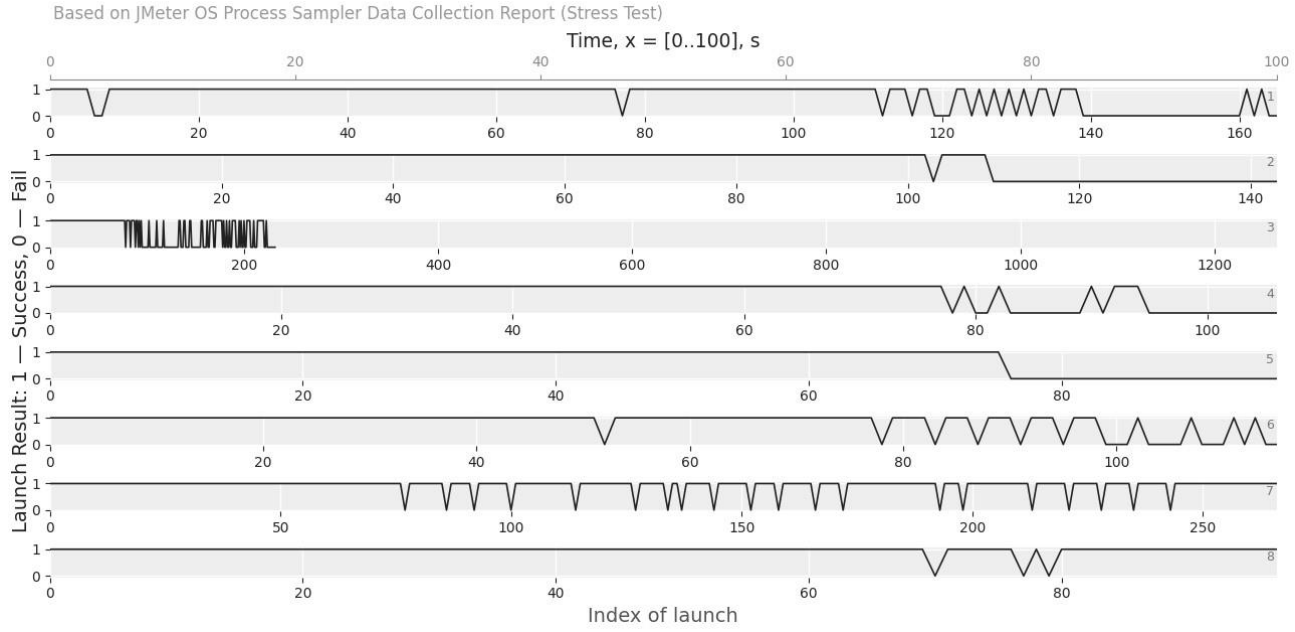



Рисунок 4.12– Результати тестування сервера

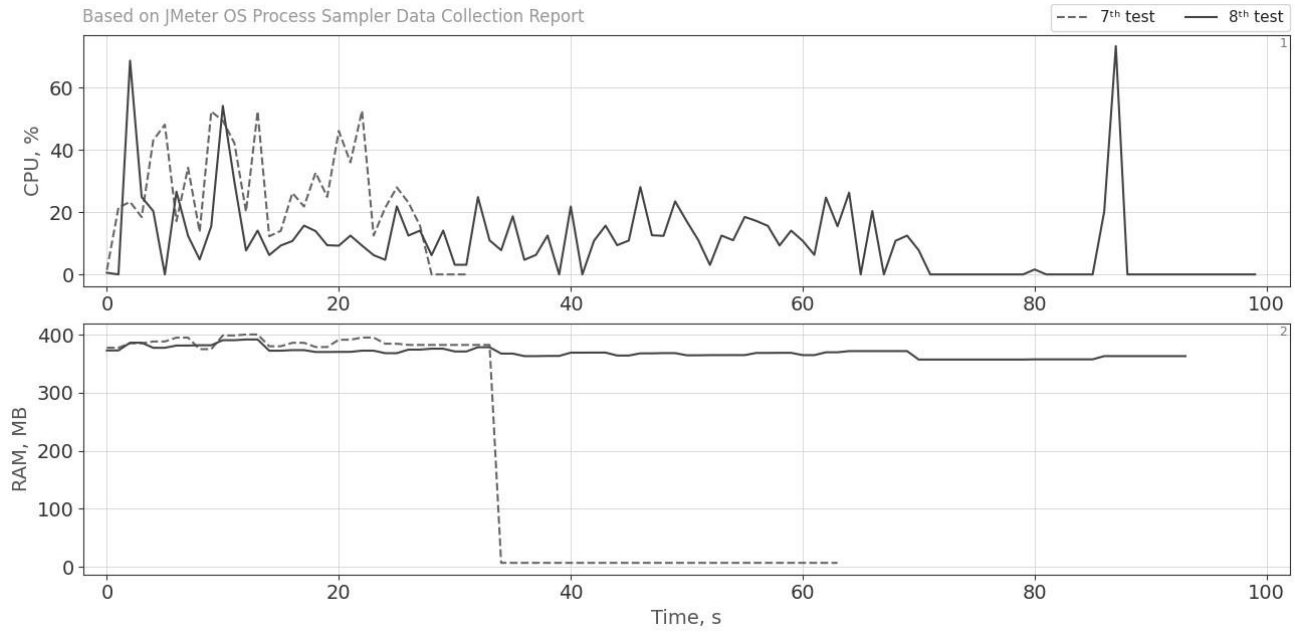



Рисунок 4.13 – Метрики для двох вибіркових тестів

Протягом усього експериментального дослідження сервер здебільшого зберігав працездатний стан, що можна пояснити відносно невисокою частотою звернень кожного програмного агента до серверного середовища, яка становила Fs = 2,2 Гц. Зазначене значення є щонайменше у шість разів меншим за частоту запуску агентів, що суттєво зменшує пікове навантаження на сервер. Водночас слід враховувати, що навантаження на сервер формується не лише в момент ініціалізації програмного агента, але й у процесі виконання ним інших операцій, зокрема реалізації поведінок та обробки внутрішніх подій, що додатково впливає на загальний стан системи.

Аналіз метрики навантаження оперативної пам’яті, представленої на рисунку 4.12, зокрема відсутність значень показника середнього навантаження апаратних засобів після тридцять шостої секунди під час сьомого тесту, дозволяє припустити виникнення збою або часткового падіння сервера. Водночас, незважаючи на припинення збору даних за метрикою навантаження центрального процесора на шістдесят другій секунді експерименту, тестувальник продовжував коректно виконувати операцію запуску програмно-орієнтованих агентів. Це свідчить про те, що відмова окремих механізмів моніторингу не обов’язково супроводжується повною втратою функціональності тестованої системи.

Подальший аналіз показав, що більшість помилок під час запуску програмно-орієнтованих агентів спостерігається у випадках, коли кількість одночасно активних ПОА перебуває в діапазоні приблизно 40–220 одиниць. За таких умов фіксувалися зависання процесів операційної системи, що згодом призводило до аварійного завершення роботи серверних застосунків як тестувальника, так і самого сервера. Встановлення обмеженої частоти запуску агентів на рівні 2,2 Гц дало змогу істотно зменшити кількість помилок, стабілізувати роботу середовища та уникнути критичних збоїв у функціонуванні програмного забезпечення.

Тенденція стійкого зростання показників, представлена на рисунку 4.13, суттєво відрізняється від тенденції часткового спадання, яка спостерігається у випадках послідовного запуску програмних агентів із незначною різницею у частотах (2,2 Гц для першого випадку та 2,31 Гц для другого). Узагальнюючи результати поточного та попереднього аналізу, можна зробити висновок, що зменшення значення частоти запуску агентів сприяє підвищенню стабільності роботи програмних застосунків, а також покращує точність і достовірність зібраних експериментальних даних.

За результатами проведеного експериментального дослідження встановлено, що використання Apache JMeter у поєднанні з додатковими плагінами та агентами моніторингу (зокрема JPAGC та PerfMon Server Agent) забезпечує комплексний і достовірний підхід до оцінювання продуктивності програмних систем і програмно-орієнтованих агентів. Запропонована методика тестування, яка включає димове тестування, навантажувальні, продуктивнісні та стрес-тести, дозволила детально проаналізувати поведінку системи за різних умов експлуатації. Отримані результати підтвердили доцільність використання Dummy Sampler для ізольованого вимірювання впливу агентів на апаратні ресурси без виконання реальних запитів, а також ефективність застосування APM-підходу для раннього виявлення потенційних проблем продуктивності.

Порівняльний аналіз архітектурних рішень продемонстрував, що інтеграція програмно-орієнтованих агентів у режимі командного рядка є більш ресурсоефективною порівняно з використанням графічних веб-інтерфейсів,

особливо з урахуванням масштабування та зростання складності інтерфейсних компонентів. Результати стрес-тестування підтвердили критичну роль контролю частоти запуску агентів, оскільки її зменшення безпосередньо впливає на стабільність серверного середовища, знижує кількість помилок і підвищує надійність зібраних даних. Таким чином, запропонований підхід до тестування та організації взаємодії програмно-орієнтованих агентів із XMPP-сервером довів свою ефективність і може бути рекомендований для застосування при розробленні та експлуатації масштабованих агентно-орієнтованих програмних систем.

## 4.4. Висновки четвертого розділу

У межах розділу було обґрунтовано логіку та послідовність розроблення програмного застосунку, де стартовою точкою визначено проєктування бази даних як системоутворювального елемента, що задає правила зберігання, узгодження та подальшої обробки інформації предметної області. Побудова концептуальної моделі БД засобами UML дозволила формалізувати ключові сутності та відношення між ними ще до реалізації, зменшивши ризики помилок на наступних етапах проєктування. Центральну роль у моделі відіграє незалежна сутність Alloy, яка містить базові атрибути і виступає «ядром» доменних даних, тоді як три залежні сутності реалізують модульне розширення, зберігаючи додаткові набори параметрів (9–10 атрибутів) і допускаючи необов’язкову присутність у конкретних екземплярах даних. Така побудова є важливою для підтримки реальних сценаріїв електронної комерції, коли не всі модулі або типи даних можуть бути доступні постійно, а система має коректно працювати за неповної інформації. Перехід від концептуальної моделі до фізичної ER-структури в MySQL підтвердив практичну придатність запропонованих рішень: було визначено первинні та зовнішні ключі, зафіксовано кардинальності та ординальності зв’язків, що забезпечило цілісність і керованість даних. Зв’язок типу 1 : 0..1 між alloy та залежними таблицями означає, що доменний об’єкт має стабільний «паспорт» у центральній таблиці й може містити не більше одного запису певного типу у відповідній залежній таблиці. Це узгоджується з логікою предметної області (одна актуальна конфігурація

балансу/стану модуля на одиницю товару), спрощує перевірку консистентності й зменшує кількість конфліктів при паралельній обробці даних агентами.

Окремим результатом є те, що структура БД була адаптована до агентно-орієнтованої логіки виконання процесів моніторингу, актуалізації та прогнозування цін. Розмежування таблиць alloy_balance, alloy_balance_old та alloy_balance_new дало змогу одночасно задовольнити вимоги оперативної роботи з поточними значеннями, збереження історії для часових рядів та контрольованого введення «нових» розрахунків/прогнозів, які можуть потребувати підтвердження або додаткових перевірок. Зокрема, alloy_balance виступає джерелом актуальних розрахункових параметрів для агентів під час оновлення цін, тоді як alloy_balance_old забезпечує історизацію для порівняння станів і підтримки прогнозних алгоритмів. Таблиця alloy_balance_new виконує роль проміжного буфера, що дозволяє відокремити стадію формування прогнозу від стадії його інтеграції в робочі дані, підвищуючи надійність та керованість переходів між станами системи. Наявність службових полів (updated, resolved, is_upd) забезпечує механізми простежуваності, дозволяє агентам маркувати результати, синхронізувати етапи обробки та уникати повторної обробки однакових даних. Таким чином, БД не лише зберігає інформацію, а й підтримує сценарії керування станами та прийняття рішень, що є принципово важливим для мультикомпонентної агентної системи в умовах змінності вхідних даних.

Порівняльний аналіз за метрикою LOC підтвердив доцільність використання платформи агентно-орієнтованого програмування для реалізації запропонованого методу. Менший обсяг коду, отриманий у межах АОПП, свідчить про зниження трудомісткості розробки та витрат на супровід, а також про зменшення ризику дефектів, які зазвичай корелюють із розміром кодової бази та складністю ручної реалізації інфраструктурних функцій. Скорочення LOC пояснюється тим, що значна частина типових для розподілених систем задач (реєстрація агентів, стандартизований обмін повідомленнями, шаблони поведінок, підтримка протоколів взаємодії) реалізована платформою та використовується повторно без необхідності дублювання коду. Це, у свою чергу, підсилює принципи модульності

та повторного використання, які були заявлені як ключові вимоги до програмної системи. Представлена програмна реалізація комунікації між ПОА, наявність подійного менеджера і агента «Рішення оператора» демонструють, що взаємодія агентів організована як керований процес з обміном повідомленнями та відстеженням станів, а отже система може підтримувати як автоматичні сценарії (оновлення, обробка), так і напівавтоматичні (втручання людини у критичних випадках). Логування поведінок агентів додатково підвищує спостережуваність системи, забезпечує аудит виконаних операцій і створює основу для подальшої діагностики та оптимізації.

Важливим практичним висновком є підтвердження ефективності механізму керування потоком обробки вхідних даних через контроль наявності нового CSV-файлу. Описаний сценарій, де система спочатку виконує завантаження та розархівування CSV, а потім ініціює актуалізацію модулів лише за умови виявлення оновлення (порівняння часових міток), демонструє раціональне використання ресурсів і запобігає зайвим обчисленням. Це узгоджується з нефункціональними вимогами щодо продуктивності та економії апаратних ресурсів, оскільки зменшується частота непотрібних циклів обробки. Такий підхід також підвищує стабільність роботи агентів у розподіленому середовищі, бо зменшує кількість «порожніх» запусків процесів і ймовірність конкурентних конфліктів за ресурси (зокрема за доступ до БД).

Окремо встановлено, що для коректної оцінки продуктивності та валідації архітектурних рішень доцільним є комплексний підхід до тестування, де використовується Apache JMeter як основний інструмент навантажувального та стрес-тестування, доповнений засобами моніторингу серверних метрик (PerfMon Server Agent / JPAGC). Така комбінація дозволила одночасно оцінювати поведінку системи під навантаженням та її реальне споживання апаратних ресурсів (CPU/ОЗП), що є критично важливим для агентних систем, у яких навантаження залежить не лише від зовнішніх запитів, а й від внутрішніх поведінок агентів. Проведення димового тестування перед основними експериментами забезпечило коректність середовища та зменшило ризик некоректних вимірювань.

Використання Dummy Sampler дало змогу ізолювати вимірювання ресурсного впливу без виконання реальних команд, що підвищує чистоту експерименту та дозволяє порівнювати архітектури на однаковій основі. Важливо й те, що розмежування тестів за типами (продуктивність ПОА, продуктивність ПС, навантаження XMPP, порівняння поведінок агентів, стрес-тест) формує відтворюваний план дослідження, який може бути використаний для подальших ітерацій оптимізації системи.

Порівняння архітектурних варіантів розміщення компонентів показало, що інтеграція агентів у систему може бути реалізована швидко, без суттєвого впливу на швидкодію у випадках, коли розташування основних модулів не змінюється. Окремо підкреслено, що ефективність CLI-підходу зростає зі збільшенням складності GUI, особливо якщо графічний інтерфейс використовує сучасні фреймворки та підключає значні бібліотечні залежності. Отже, агентно-орієнтовані механізми керування через командний рядок або легкі інтерфейси можуть бути ресурсно вигіднішими для операцій, що не потребують складної візуалізації, і саме така стратегія є доцільною для фонового виконання актуалізації/прогнозування. Результати стрес-тестування також продемонстрували критичну роль контролю частоти запуску агентів: стабільність сервера та точність даних підвищуються зі зменшенням інтенсивності запусків, оскільки знижується пікове навантаження і зменшується кількість збоїв. Виявлений діапазон (приблизно 40–220 активних агентів), у якому помилки виникають частіше, може бути використаний як практичний орієнтир для налаштування політик масштабування, обмеження паралельності та планування запусків агентних процесів.

Завершальним узагальненням є те, що результати розділу підтверджують методологічну узгодженість обраного підходу: база даних спроєктована як стабільне ядро домену, поверх якого реалізовано агентні механізми взаємодії, а вся система верифікована через відтворюваний план тестування продуктивності та стресостійкості. Фізична діаграма розгортання підсилила цей висновок, оскільки продемонструвала можливість розподілу компонентів між обчислювальними вузлами та зниження зв’язності при застосуванні АОПП. Це означає, що система

може еволюціонувати до більш децентралізованих сценаріїв, зберігаючи керованість та цілісність даних завдяки центральному серверному середовищу та протоколам взаємодії. У підсумку поєднання продуманої ER-структури, агентно-орієнтованого підходу до реалізації та комплексного тестування створює надійну основу для побудови масштабованого, розширюваного і придатного до повторного використання програмного рішення

# ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне науково-прикладне завдання розробки методу удосконалення процесу модульного проєктування агентно-орієнтованих програмних систем із забезпеченням розширюваності та повторного використання компонентів, а також підтверджено практичну доцільність застосування агентних технологій для побудови гнучких програмних рішень у динамічних доменах електронної комерції та обробки даних. [ОВ]

У межах досягнення мети дослідження виконано системний аналіз предметної області та обґрунтовано перехід від традиційних об’єктно-орієнтованих підходів до агентно-орієнтованих, оскільки останні забезпечують вищий рівень автономності компонентів, зниження зв’язності між підсистемами, кращу адаптивність до змін вимог і середовища, а також перспективність інтеграції інтелектуальних механізмів. У ході огляду наукових джерел узагальнено еволюцію поняття програмно-орієнтованого агента, ключові властивості агентної поведінки (реактивність/проактивність, соціальність, автономність) та формальні підходи, що лежать в основі AOSE. Це дозволило сформувати концептуальний базис для подальшого проєктування методу удосконаленого модульного проєктування.

Основним результатом роботи є запропонований метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем, у якому агент розглядається як композиція формалізованих модулів поведінки з чітко визначеними інтерфейсами та контрактами взаємодії. Такий підхід забезпечує, керовану розширюваність — можливість додавання нових функцій без перебудови всієї системи та повторне використання — перенесення перевірених модулів/патернів у нові проєкти, спрощення супроводу — локалізацію змін у межах окремих модулів; передумови автоматизованої перевірки сумісності модулів, а також узгодженості їхньої взаємодії.

У другому розділі обґрунтовано вибір типу агентів і архітектурних характеристик цільового рішення, зокрема доцільність використання статичних інтернет-мікроагентів як достатніх для реалізації операцій моніторингу,

актуалізації та прогнозування даних у межах визначеного класу задач. Сформовано перелік низькорівневих вимог до програмної системи (обробка вхідних даних, підтримка БД, продуктивність без додаткових апаратних ресурсів, динамічне створення/видалення агентів, координація доступу до спільних ресурсів, CLI-взаємодія тощо). На підставі співставлення вимог і можливостей агентних платформ показано, що застосування АОПП дозволяє закрити суттєву частину первинних вимог та покращити розширюваність/масштабованість майбутнього програмного засобу.

У третьому розділі визначено технологію реалізації запропонованого методу із застосуванням еволюційного (вертикального) прототипування як найбільш придатної моделі життєвого циклу для умов змінних вимог і потреби швидкого отримання робочих інкрементів. Проведено порівняльний аналіз поширених моделей ЖЦ (каскадна, V-модель, спіральна, ітеративна, Agile, прототипування) та аргументовано вибір прототипування, включно з аналізом ризиків (помилкове сприйняття прототипу як готового продукту, можливе «закріплення» тимчасових архітектурних рішень, розмиття критеріїв завершення ітерацій). Запропоновано практичні кроки нейтралізації ризиків за рахунок розмежування вимог до прототипів і кінцевого продукту та концентрації на архітектурній цілісності.

У четвертому розділі виконано програмну реалізацію та експериментальне оцінювання розробленого рішення. Показано, що раціонально починати реалізацію зі схеми даних: сформовано концептуальну UML-модель та побудовано фізичну ER-модель у MySQL, де центральна сутність відокремлює стабільні доменні атрибути від змінних часово-залежних показників, винесених у залежні таблиці. Така організація даних забезпечує підтримку актуальних значень, історизацію та контрольоване введення «нових» розрахунків/прогнозів, що узгоджується з агентною логікою виконання сценаріїв і підсилює модульність системи.

Комунікацію між агентами реалізовано на основі повідомлень у серверному середовищі (XMPP-підхід), забезпечено сценарії подійного керування, взаємодії з оператором та логування поведінок агентів. Продемонстровано механізм уникнення зайвих обчислень через контроль наявності нового вхідного CSV-файлу

(порівняння часових міток), що підвищує ресурсну ефективність та стабільність виконання процесів.

Для валідації запропонованих рішень застосовано комплексний підхід до тестування продуктивності та стресостійкості із використанням Apache JMeter і серверних агентів моніторингу. Побудовано відтворюваний план експериментів (димове тестування, продуктивнісні/навантажувальні/стрес-тести, моніторинг CPU та RAM), що дозволило проаналізувати поведінку системи під різними умовами. Експериментально підтверджено, корисність контролю частоти запуску агентів для підвищення стабільності сервера, перспективність більш «легких» інтерфейсних рішень (зокрема CLI) в аспекті ресурсоощадності порівняно з важкими веб-інтерфейсами, практичну доцільність агентного підходу для зменшення зв'язності та підтримки розподіленого розгортання компонентів.

Наукова новизна роботи полягає в удосконаленні методичного підходу до проєктування агентно-орієнтованих систем через модульну композицію поведінкових компонентів, що забезпечує стандартизовану розширюваність і повторне використання, а також у практичному поєднанні цього підходу з еволюційним прототипуванням та контрольованими механізмами комунікації агентів. Практична значимість полягає у можливості використання запропонованого методу як основи для створення та розвитку масштабованих програмних систем, де критичними є модульність, автономність компонентів, підтримуваність і готовність до інтеграції сучасних технологій (включно з ШІ-підходами).

Перспективи подальших досліджень доцільно пов'язати з розширенням каталогу повторно використовуваних агентних модулів і шаблонів протоколів взаємодії, автоматизацією перевірки сумісності модулів на рівні контрактів і сценаріїв комунікації, інтеграцією інтелектуальних механізмів (класифікації, прогнозування, адаптації поведінок) у вигляді окремих підключуваних модулів, масштабуванням на більші навантаження через політики оркестрації агентів та обмеження паралельності, підвищенням інформаційної безпеки комунікацій та контролю доступу в розподіленому середовищі.

Загалом, поставлену мету кваліфікаційної роботи досягнуто: розроблено й обґрунтовано метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем, виконано проєктування та реалізацію програмного засобу, а також проведено експериментальну перевірку ефективності прийнятих архітектурних і технологічних рішень.

Перелік джерел посилання

1. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. Comput. Sci. 2015. p.38–39.

2. Peng, J.; Fan, B.; Liu, W. Voltage-Based Distributed Optimal Control for Generation Cost Minimization and Bounded Bus Voltage Regulation in DC Microgrids. IEEE Trans. Smart Grid 2021. p.106–116.

3. Dimarogonas, D.V.; Frazzoli, E.; Johansson, K.H. Distributed Event-Triggered Control for Multi-Agent Systems. IEEE Trans. Autom. Control. 2022. p. 291–297.

4. Dietrich, D., Bruckner, D., Zucker, G. and Palensky, P. Communication and computation in buildings: A short introduction and overview. IEEE Trans. Ind. Electronics 57.2020. p. 577-584.

5. HuginExpert (2020) HUGIN Graphical User Interface Documentation, Release 8.9. 2020. URL: https://download.hugin.com/webdocs/manuals/GUI/.

6. Klinc, R., Turk, Z. Construction 4.0 – digital transformation of one of the oldest industries. Economic and Business Review 21 (3).2020. pp. 393–410.

7. Nota, G., Peluso, D., Toro Lazo, A. The contribution of Industry 4.0 technologies to facility management. International Journal of Engineering business management, Volume 13. 2021. p. 1–14.

8. Pasetti Monizza, G., Bendetti, C., Matt, D.T. Parametric and generative design techniques in massproduction environments as effective enablers of Industry 4.0 approaches in the Building Industry, Automation in Construction.2020. p. 270–285.

9. Alzayed A. Evaluating the Role of Requirements Engineering. Practices in the Sustainability of Electronic Government Solutions. Sustainability 16, no. 1. 2024. p. 433.

10. Ikram A., Jalil M. A., Ngah A. B. Project Assessment in Offshore Software Maintenance Outsourcing Using Deep Eltreme Learning Machines. Computers, Materials and Continua. 2023. p.871-886.

11. Yaseen M., Mustapha A., Ibrahim N. Prioritization of Software Functional Requirements From Developers Perspective. International Journal of Advanced Trends

in Computer Science and Engineering 11, no. 9. 2020. p.183

12. Bessenrodt C.. Maximal Multiplicative Properties of Partitions/ Annals of Combinatorics 20, no. 2020. p. 59 -64.

13. Chern S., Fu S., Tang D. Some Inequalities for k-Colored Partition Functions. The Ramanujan Journal 46, no. 3. 2018. p. 713-725.

14. Yaseen M., Mustapha A., Zaman S. U., Khan S. U., Rahman A. U. Functional Requirements Prioritization Using Graph and Sorting Algorithm. International Journal of Scientific. Research in Computer Science and Engineering 8, no. 4. 2020.p. 14-22

15. Win T. Z., Mohamed R., Sallim J.. Requirement Prioritization Based on Non-Functional Requirement Classification Using Hierarchy AHP,"IOP Conference Series: Materials Science and Engineering. 2020. p.120-160.

16. Jaya S. M. A. An Integrated Approach Towards Automated Software Requirements Elicitation From Unstructured Documents," Journal of Ambient Intelligence and Humanized Computing. 2020. p. 763–773.

17. Chiosa, R., Piscitelli, M. S., Pritoni, M., Capozzoli, A. A portable application framework for energy management and information systems (EMIS) solutions using Brick semantic schema, Energy &Buildings.2024. p.114-122.

18. Cai, B., Liu, Y., Hu, J., Liu, Z., Wu, S., and Ji, R. (2018) Byesian Networks in Fault Diagnosis: Practice and Application. World Scientific. 2025. URL: https://www.worldscientific.com.

19. Chen Z. A review of data-driven fault detection and diagnostics for building HVAC systems, Applied Energy. 2023. p.121-130.

20. Yang Deng, Lizi Liao, Zhonghua Zheng, Grace Hui Yang, Tat-Seng Chua.. Towards humancentered proactive conversational agents. In Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2024. p. 807–818.

21. Chongming Gao, Wenqiang Lei, Xiangnan He, Maarten de Rijke, Tat-Seng Chua. Advances and challenges in conversational recommender systems. AI Open, 2021. p.100–126.

22. Dietmar Jannach, Ahtsham Manzoor, Wanling Cai, Li Chen.. A survey on conversational recommender systems. ACM Computing Surveys (CSUR). 2021. p.1–36.

23. Ivica Kostric, Krisztian Balog, and Filip Radlinski. Soliciting user preferences in conversational recommender systems via usage-related questions. In Proceedings of the 15th acm conference on recommender systems. 2021. p. 724–729

24. Chuang Li, Yang Deng, Hengchang Hu, Min-Yen Kan, Haizhou Li. Chatcrs: Incorporating external knowledge and goal guidance for llm-based conversational recommender systems. In Findings of the Association for Computational Linguistics: NAACL. 2025, pages 295–312.

25. Allen Lin, Ziwei Zhu, Jianling Wang, and James Cave.lee. 2023a. Enhancing user personalization in conversational recommenders. In Proceedings of the ACM Web Conference. 2023, p. 770–778.

26. Kwangwook Seo, Donguk Kwon, and Dongha Lee. MT-RAIG: Novel benchmark and evaluation framework for retrieval-augmented insight generation over multiple tables. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2025. p. 142–172

27. Yueming Sun and Yi Zhang. Conversational recommender system. In The 41st international acm sigir conference on research & development in information retrieval.2020. p.235–244.

28. Xiaolei Wang, Kun Zhou, Ji-Rong Wen, and Wayne Xin Zhao.. Towards unified conversational recommender systems via knowledge-enhanced prompt learning. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2022. p.1929–1937.

29. An Zhang, Yuxin Chen, Leheng Sheng, Xiang Wang, and Tat-Seng Chua. 2024. On generative agents in recommendation. In Proceedings of the 47th international ACM SIGIR conference on research and development in Information Retrieval.2024. p. 807–817.\

30. Shuo Zhang and Krisztian Balog.. Evaluating conversational recommender systems via user simulation. In Proceedings of the 26th acm sigkdd international conference on knowledge discovery & data mining.2020. p. 512–520.

31. Lixi Zhu, Xiaowen Huang, and Jitao Sang. 2024. How reliable is your simulator? analysis on the limitations of current llm-based user simulators for conversational recommendation. In Companion Proceedings of the ACM on Web Conference 2024, p. 726–732.

32. Y. Kehan and L. Peng, “Digital identity management for ai agent communication protocols,” Internet Engineering Task Force, InternetDraft draft-yl-agent-id-requirements-00. 2025. URL: https://datatracker.ietf.org/doc/draft-yl-agent-id-requirements/

33. R. Jonathan and W. Pat, “Aauth - agentic authorization oauth 2.1 extension,” Internet Engineering Task Force, Internet-Draft draft-rosenberg-oauth-aauth-00. 2025. URL: https: //datatracker.ietf.org/doc/draft-rosenberg-oauth-aauth/

34. H. Ken, S. N. Vineeth, H. Idan, and S. Akram, “Agent name service (ans): A universal directory for secure ai agent discovery and interoperability,” Internet Engineering Task Force, InternetDraft draft-narajala-ans-00, May 2025. URL: https: //datatracker.ietf.org/doc/draft-narajala-ans/

35. D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, M. Sabadello, and J. Holt, “Decentralized identifiers (dids) v1. 0,” Draft Community Group Report, 2020.

36. G. Kellogg, P.-A. Champin, and D. Longley, “Json-ld 1.1–a json-based serialization for linked data,” Ph.D. dissertation, W3C, 2020.

37. Hugo Alcaraz-Herrera, John Cartlidge, Zoi Toumpakari, Max Western, and Iván Palomares. 2022. EvoRecSys: Evolutionary framework for health and well-being recommender systems. User Modeling and User-Adapted Interaction 32, 5 (2022), 883--921.

38. Ashmi Banerjee, Tunar Mahmudov, and Wolfgang Wörndl. 2024. Green Destination Recommender: A Web Application to Encourage Responsible City Trip Recommendations. In Adjunct Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization. 486--490.

39. Alexander Felfernig, Damian Garber, Viet-Man Le, Sebastian Lubos, and Thi Ngoc Trang Tran. 2025. Sustainability Evaluation Metrics for Recommender Systems. arXiv preprint arXiv:2507.22520 (2025).

40. Sebastian Lubos, Thi Ngoc Trang Tran, Alexander Felfernig, Seda Polat Erdeniz, and Viet-Man Le. 2024. LLM-generated explanations for recommender systems. In Adjunct Proceedings of the 32nd ACM Conference on User Modeling, Adaptation and Personalization. 276--285.

41. Pavel Merinov. 2023. Sustainability-oriented recommender systems. In Proceedings of the 31st ACM Conference on User Modeling, Adaptation and Personalization. 296--300.

42. Pavel Merinov. 2023. Sustainability-oriented recommender systems. In Proceedings of the 31st ACM Conference on User Modeling, Adaptation and Personalization. 296--300.

43. Mehrdad Rostami, Ali Vardasbi, Mohammad Aliannejadi, and Mourad Oussalah. 2024. Emotional Insights for Food Recommendations. In European Conference on Information Retrieval. Springer, 238--253.

44. Gayane Sedrakyan, Anand Gavai, and Jos van Hillegersberg. 2023. Design implications towards human-centric semantic recommenders for sustainable food consumption. In International Conference on Conceptual Modeling. Springer. 2023.p.312--328.

45. Giuseppe Spillo, Allegra De Filippo, Cataldo Musto, Michela Milano, and Giovanni Semeraro. Towards sustainability-aware recommender systems: analyzing the trade-off between algorithms performance and carbon footprint. In Proceedings of the 17th ACM Conference on Recommender Systems.2023. p. 856--862.

46. Giuseppe Spillo, Allegra De Filippo, Cataldo Musto, Michela Milano, and Giovanni Semeraro. Comparing data reduction strategies for energy-efficient green recommender systems. Journal of Intelligent Information Systems. 2025. p.1-27.

47. ShoujinWang, Xiuzhen Zhang, YanWang, and Francesco Ricci. Trustworthy recommender systems. ACM Transactions on Intelligent Systems and Technology 15. 2024. p.1-20.

ДОДАТОК А

(Обов’язковий)

Копії наукових публікацій

Міністерство освіти і науки України
Хмельницький національний університет


ЗБІРНИК НАУКОВИХ ПРАЦЬ
за матеріалами XVII Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп’ютерних наук АПКН-2025»

***14-15 листопада 2025***

Хмельницький 2025

УДК 004.41

Стецюк П.П., Форкун Ю.В.

*Хмельницький національний університет*

# МЕТОД УДОСКОНАЛЕНОГО МОДУЛЬНОГО ПРОЄКТУВАННЯ АГЕНТНО-ОРІЄНТОВАНИХ ПРОГРАМНИХ СИСТЕМ З ПІДТРИМКОЮ РОЗШИРЮВАНОСТІ ТА ПОВТОРНОГО ВИКОРИСТАННЯ

*У роботі представлено онтологічний підхід до модельного проектування мультиагентних систем. Розроблено метамодель та інструмент для автоматичної генерації коду. Запропонований підхід підвищує узгодженість моделей, скорочує час розробки й забезпечує формальну верифікацію систем.*

*The paper presents an ontological approach to model-based design of multi-agent systems. A metamodel and a tool for automatic code generation are developed. The proposed approach increases model consistency, reduces development time, and provides formal verification of systems.*

Розглянуто підхід до вдосконалення процесів розробки мультиагентних систем (MAS) шляхом поєднання методів модельно-орієнтованої інженерії (MDE) та онтологій. Запропонована методика забезпечує уніфіковану модель, що охоплює три ключові виміри MAS — агентний, середовищний та організаційний — з метою інтеграції проектування, кодування та верифікації систем у єдиному семантичному просторі.

Мета роботи полягає у проведенні комплексного дослідження різних методів, для розробки онтологічних підходів у галузі моделювання MAS, зокрема для підтримки розробників MAS та дослідження його використання, а також збору та підготовці матеріалу для виконання наукової роботи.

Проведено аналіз існуючих методологій AOSE (агентно-орієнтованої інженерії програмного забезпечення), таких як наприклад Prometheus, які демонструють високий потенціал у створенні гнучких інтелектуальних систем, проте не забезпечують повної інтеграції різних рівнів моделювання. Запропонований підхід ґрунтується на онтологічній метамоделі, яка дозволяє здійснювати семантичне міркування, автоматичну генерацію коду та перевірку узгодженості між моделями та реалізацією.

У межах дослідження розроблено вдосконалений інструмент, що забезпечує перетворення моделей у вихідний код для платформи MAS. Цей підхід дозволяє скоротити час розробки, підвищити точність відтворення моделей у коді та уникнути розбіжностей між проектною специфікацією та реалізацією. Використання онтології як метамоделі забезпечує формальну специфікацію знань і

створює передумови для багаторазового використання компонентів MAS у різних проектах.

Ключові результати дослідження полягають у:

– формалізації методології онтологічного моделювання мультиагентних систем;

– створенні інтегрованої моделі, що поєднує агентів, середовище та організацію;

– розробці механізмів автоматичної генерації коду з онтологічних моделей;

– розробці інструментального середовища;

– оцінюванні ефективності підходу на різних прикладах проектів MAS.

Результати експериментів засвідчують, що використання онтологій на етапах проектування та програмування підвищує узгодженість компонентів, зменшує ризики помилок та сприяє підвищенню продуктивності розробки. Семантичне моделювання також забезпечує можливість автоматизованої перевірки властивостей системи на логічну несуперечність, що є важливим кроком до створення надійних та масштабованих інтелектуальних систем.

Запропонована методологія може бути застосована для розробки широкого спектра багатокомпонентних систем — від систем підтримки прийняття рішень до розподілених сервісних платформ. Отримані результати відкривають перспективи подальших досліджень у напрямі автоматизації агентного програмування, розвитку стандартів представлення знань та інтеграції онтологій у процеси інженерії програмного забезпечення.

**Перелік посилань**

1. Muhammad Yaseen, Esraa Ali , Nadeem Sarwar, Leila Jamel, Irfanud Din, Farrukh Yuldashev, Foongli Law. Design of Minimal Spanning Tree and Analytic Hierarchical Process (SAHP) Based Hybrid Technique for Software Requirements Prioritization. WileyIET SoftwareVolume 2025. Article ID 8819735, 23. URL: https://doi.org/10.1049/sfw2/
2. Harder D.W. A practical introduction to real-time systems for undergraduate engineering. University of Waterloo, 2023. 744p.
3. Ikram A., Jalil M. A., Ngah A. B. Project Assessment inOffshore Software Maintenance Outsourcing Using DeepEltreme Learning Machines, Computers, Materials and Continua 74.2023. 1871–1886
4. Alzayed A. Evaluating the Role of Requirements EngineeringPractices in the Sustainability of Electronic GovernmentSolutions, Sustainability 16, no. 1.2024
5. Hassan M. A., Zhang D. A Review on Industrial Robot Control and Applications. IEEE Access. 2021. Vol. 9. pp. 112–130.
6. Левченко Л. О. Архітектура системного програмного забезпечення: підручник. Київ.КПІ ім. І. Сікорського. 2022. 497 с.
7. Копиленко О. Внутрішній пристрій MySQL для фахівців з оптимізації. Літура. 2023. 420с.

ДОДАТОК Б

(Обов’язковий)

Презентаційні матеріали

1

**ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# Метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання


**Виконав:**
**студент групи ІПЗм-24-1**
**Стецюк Павло Петрович**

**Керівник:**
**кандидат технічних наук, доцент**
**Форкун Юрій Вікторович**


2

## Актуальність теми дослідження

### Науковий контекст

Спостерігається стійке зростання публікацій у сфері AOSE (Agent-Oriented Software Engineering). Наукова спільнота активно досліджує методи побудови автономних систем, визнаючи MAS ключовою технологією для майбутнього розподілених обчислень.

### Прогалини досліджень

Традиційні ООП-підходи мають високу зв'язність компонентів, що ускладнює масштабування. Існуючим агентним методам часто бракує стандартизованих механізмів для ефективного повторного використання модулів поведінки.

### Сучасні тенденції

Зростання складності систем електронної комерції та індустрії  вимагає переходу від монолітних до децентралізованих архітектур. Критичними стають вимоги до автономності та адаптивності без зупинки системи.

### Практична значущість

Запропонований метод поєднує гнучкість агентів зі швидкістю прототипування. Це дозволяє скоротити час розробки, зменшити споживання ресурсів  та забезпечити реальну розширюваність систем.

# Мета і завдання дослідження

**Мета:** Розробка методу удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання із застосуванням прототипування для підвищення швидкості розробки.

1. Аналіз предметної області (ООПП vs АОПП)

2. Розробка методу проєктування

3. Технологія реалізації методу

4. Програмна реалізація та тестування

# Об’єкт і предмет дослідження

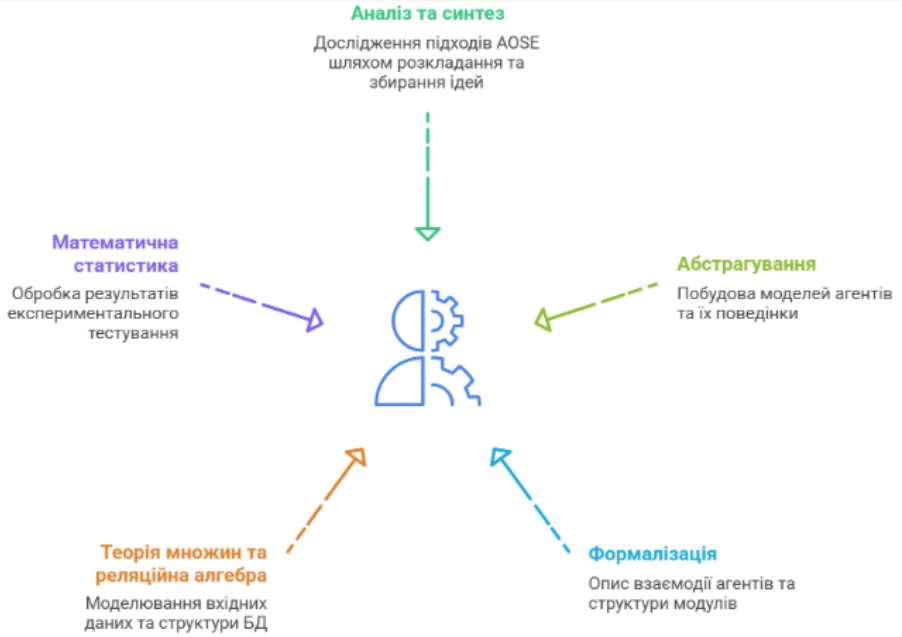
## Об’єкт дослідження

Процес моделювання та проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання.

## Предмет дослідження

Методи, архітектурні принципи агентно-орієнтованих ПС та програмні засоби, що забезпечують удосконалення проєктування та підвищення ефективності розробки.

# Теоретичні методи дослідження






# Емпіричні та технологічні методи дослідження

# Наукова новизна

### Наукова новизна

- **Удосконалення методичного підходу:** Проєктування агентно-орієнтованих систем здійснюється через модульну композицію поведінкових компонентів.
- **Стандартизація:** Забезпечення стандартизованої розширюваності та повторного використання коду.
- **Інноваційне поєднання:** Інтеграція підходу з еволюційним прототипуванням та контрольованими механізмами комунікації агентів.

### Практична значимість

- **Основа для масштабування:** Використання методу для створення складних програмних систем.
- **Критичні характеристики:** Забезпечення модульності, автономності компонентів та їх підтримуваності.
- **Готовність до технологій:** Архітектура готова до інтеграції сучасних рішень, включно з підходами штучного інтелекту (ШІ).



# Розділ 1. Аналіз предметної області

## Недоліки ООП

Ускладнена підтримка розподілених обчислень, залежність від планувальників. утруднений подальший розвиток

## Переваги АОПП

- BDI-архітектура
- Децентралізація

## Вибір типу агента

Обрано рефлексивні статичні інтернет-мікроагенти через відсутність потреби у складній мобільності коду, акцент на обробці даних.

# Розділ 1. Порівняння підходів

### Швидкість розробки

Використання готових патернів поведінки прискорює створення системи.

### Автономність

Зниження залежностей від зовнішнього керування та між компонентами спрощує зміни

### Реактивність

Миттєва реакція на зміни середовища

**Висновок аналізу: АОПП дозволяє зменшити зв'язність системи та спростити масштабування.**



# Розділ 2. Розробка методу

### Суть методу

Поєднання **АОПП** з **вертикальним еволюційним прототипуванням** для мінімізації ризиків та поетапного нарощування функціоналу.

### Вимоги до системи:

- **Функціональні:** Регулярний моніторинг CSV-файлів, актуалізація цін, фіксація екстремальних значень.
- **Нефункціональні:** Робота без додаткових апаратних ресурсів, підтримка CLI інтерфейсу.
- **Архітектурні:** Динамічне створення агентів, асинхронні протоколи повідомлень.

# Розділ 2. Концептуальна модель системи

### Агентна складова

- **InventoryManager:** Відповідає за моніторинг файлів, розпакування архівів та оновлення БД.
- **HumanDecision:** Забезпечує інтерактивну взаємодію з оператором для прийняття складних рішень.

### Інфраструктура

- **XMPP-сервер:** Середовище обміну повідомленнями (Messaging).
- **База даних:** Зберігання станів та історії (таблиці балансу).
- **Взаємодія**: Асинхронний обмін замість прямих викликів.



# Розділ 3. Технологія та Методологія

### Засоби розробки

**Вибір платформи: SPADE (Python)**

- На відміну від застарілого JADE (Java), SPADE базується на протоколі миттєвих повідомлень **XMPP**.
- Забезпечує інтеграцію з сучасними бібліотеками ML (TensorFlow, PyTorch).
- Підтримує асинхронну модель, критичну для мікроагентів.

### Підхід до проєктування

**Вертикальне еволюційне прототипування**

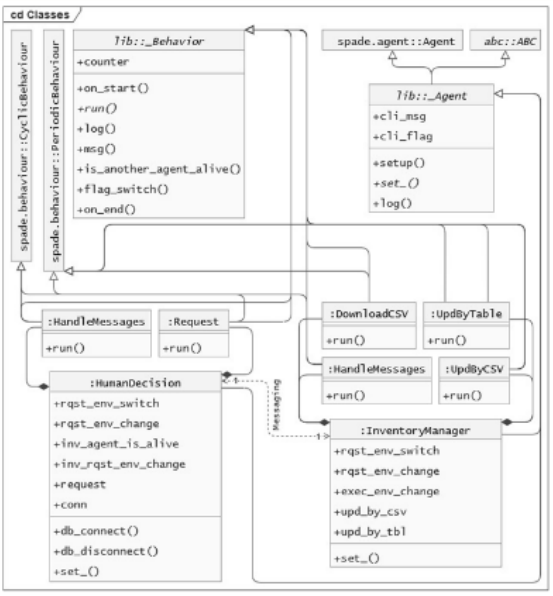
- Обрано на основі порівняльного аналізу
- Дозволяє поетапно реалізовувати функціонал (від "скелету" до повної системи).
- Найкраще відповідає вимогам швидкої розробки інтерфейсу та мінімізації ризиків.

# Розділ 3. Проєктування архітектури

### Структурна організація

- **Функціональне групування:** Класи об'єднано у логічні пакети (Data Layer, Agents, UI) для зменшення зв'язності.
- **Об'єктно-орієнтована декомпозиція:** Система розкладена на незалежні підсистеми, що дозволяє паралельну розробку.

*Розроблено діаграми класів та пакетів, які фіксують статичні зв'язки (узагальнення, асоціації) та забезпечують основу для повторного використання компонентів.*




# Розділ 4. Програмна реалізація

Реалізація базується на фізичній **ER-моделі MySQL**, яка підтримує логіку агентів:

| Таблиця | Тип сутності | Роль у системі |
|---|---|---|
| alloy | Незалежна | Зберігає стабільні характеристики товару (ID, опис). |
| alloy_balance | Залежна | Динамічні дані: поточні ціни, стан. Зв'язок **1 : 0..1**. |
| alloy_balance_old | Архівна | Історичні дані для прогнозування трендів. |

**</>Ефективність реалізації (LOC):**
Аналіз метрики *Lines of Code* показав суттєве скорочення обсягу коду порівняно з традиційним підходом, завдяки використанню вбудованих механізмів платформи SPADE (реєстрація, ACL-повідомлення).

# Розділ 4. Тестування

## Економія ресурсів (CLI vs Web GUI)

Використання агентів у режимі командного рядка суттєво зменшує навантаження на систему порівняно з традиційним веб-інтерфейсом.


## Стрес-тестування:

- Критичний поріг збоїв: 40–220 одночасних агентів.
- **Стабілізація:** Обмеження частоти запуску до 2.2 Гц повністю усунуло збої.



# Висновки

### Мета досягнута

Розроблено та програмно реалізовано метод модульного проєктування АОПС.

### Loose Coupling

Знижено рівень зв'язності системи та забезпечено повторне використання модулів.

### Доведення

Експериментально доведено ефективність прототипування та CLI-реалізації

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання»
Автор: Стецюк Павло Петрович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Форкун Юрій Вікторович, кандидат технічних наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

| № | Висновок | Позначка про відповідність |
|---|---|---|
| 1 | Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту. | **відповідає** |
| 2 | Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи | |
| 3 | Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат. | |
| 4 | Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту. | |
| 5 | Інше: | |

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті дипломної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, назвах розділів/підрозділів тощо та в назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланя на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Максимальний обсяг запозичень, визначений системою Anti-Plagitarism, складає 2 %. За системою StrickerPlagitarism коефіцієнт подібності складає 1,5 %, коефіцієнт подібності 2 складає 2 %.

Дата 15.12.2025

Керівник _____ Ю.В. Форкун

Гарант ОП _____ О.М. Яшина

Завідувач кафедри _____ Л.П. Бедратюк

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Стецюка П.П.
Прізвище, ініціали

факультет ІТ, 2 курс, група ІПЗм-24-1

**ЗАЯВА**

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

08.12.2025
дата

підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

**РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ**
освітнього ступеня «Магістр»

Дипломник Стецюк Павло Петрович

Тема Метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання

Спеціальність 121 – Інженерія програмного забезпечення

**Обсяг дипломного проекту:**

Кількість листів креслень _____; кількість сторінок записки 94

1.Короткий зміст пояснювальної записки та прийнятих рішень У магістерській роботі розроблено метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання. Запропонований підхід поєднує агентно- та об'єктно-орієнтовані технології, що забезпечує підвищення ефективності проєктування, прискорення розробки та спрощення повторного використання програмних компонентів.

2. Висновок про відповідність проекту поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У першому розділі виконано аналіз предметної області, сучасного стану агентно-орієнтованого проєктування та обґрунтовано доцільність використання агентних підходів з позицій сучасних досягнень науки і техніки. У другому розділі розроблено метод удосконаленого модульного проєктування, що базується на композиції агентних модулів з чіткими інтерфейсами, із застосуванням передових архітектурних принципів повторного використання та розширюваності. Третій розділ присвячено вибору технологій і моделі життєвого циклу з використанням еволюційного прототипування та сучасних інструментів розробки. У четвертому розділі реалізовано програмний застосунок, проведено тестування й валідацію з використанням актуальних засобів моніторингу та навантажувального тестування, що підтвердило ефективність і практичну цінність запропонованих рішень.вимог, в результаті було підтверджено коректну роботу програмного комплексу.

4. Позитивні сторони проекту Тематика дипломного проекту є актуальною, оскільки на робота з проєктування агентно-орієнтованих програмних систем є досить затребуваною.

5. Негативні сторони проекту У проекті опис готових рішень не є повним, доцільно додати розширений аналіз прийнятих рішень.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконано відповідно до теми дипломної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів

7. Відгук про дипломний проект в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики дипломної роботи. Графічний матеріал дає можливість наочно побачити деталі проектування програмного засобу.

8. Інші зауваження

9. Оцінка дипломного проекту Кваліфікаційна робота виконана на достатньому рівні, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Пилипчук Олександра Степанівна, д. т. н., професор, професор кафедри КІС

"15" грудня 2025 р. _____
(підпис)

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗм-24-1
Стецюка П.П.

**ЗАЯВА**

Прошу закріпити за мною тему дипломної роботи освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення»: «Метод удосконаленого модульного проєктування агентно-орієнтованих програмних систем з підтримкою розширюваності та повторного використання»

(керівник дипломного проекту – к.т.н., доцент Форкун Юрій Вікторович)

01.09.2025
Дата

Підпис студента