МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Кваліфікаційна наукова
праця на правах рукопису

ЗІНОВ'ЄВ ДМИТРО ВОЛОДИМИРОВИЧ

УДК 004.273:004.05:004.89

ДИСЕРТАЦІЯ

ІНТЕЛЕКТУАЛЬНІ МОДЕЛІ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ ДЛЯ АДАПТИВНОГО УПРАВЛІННЯ КОНФІГУРАЦІЯМИ ПРОГРАМНИХ МІКРОСЕРВІСІВ

Спеціальність 122 – Комп’ютерні науки
Галузь знань 12 – Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ **Д.В. Зінов'єв**

Науковий керівник: **Ткачук Микола Вячеславович**
доктор технічних наук, професор

Харків – 2026

АНОТАЦІЯ

Зінов'єв Д. В. Інтелектуальні моделі та інструментальні засоби для адаптивного управління конфігураціями програмних мікросервісів - Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 122 Комп'ютерні науки (Галузь знань 12 Інформаційні технології), Харківський національний університет імені В. Н. Каразіна, Міністерство освіти і науки України, Харків, 2026 р.

Дисертація присвячена вирішенню актуальної науково-технічної задачі підвищення якості процесів розробки та супроводу програмних систем (ПС) з мікросервісною архітектурою (МСА) шляхом адаптивного управління їх конфігураціями із використанням знання-орієнтованої багатовимірної інформаційної моделі для опису інфраструктури операційного середовища функціонування ПС з МСА у поєднанні із методами логічного висновку на основі аналізу прецедентів.

У *вступі* обґрунтовано актуальність обраної тематики та показано її зв'язок з науковими розробками кафедри, де виконувалася дисертаційна робота. Сформульовано основну мету, визначено об'єкт, предмет і методи дослідження, до яких відносяться: принципи прикладного системного аналізу, математичний апарат загальної теорії множин; побудова доменних онтологій, методи логічного висновку на основі аналізу прецедентів; об'єктно-орієнтований аналіз та синтез ПЗ з використанням уніфікованої мови системного моделювання UML (Unified Modeling Language) для формалізації процедур проектування інструментальних програмних засобів, методи обробки статистичних даних для аналізу проведених експериментів.

Описано наукову новизну та практичне значення отриманих результатів дисертаційного дослідження, наведено інформацію про особистий внесок здобувача у публікаціях, які зроблені у співавторстві, про апробацію результатів дослідження на конференціях та семінарах, а також відомості щодо структури та обсягу дисертаційної роботи.

У першому розділі проведено аналітичний огляд сучасних розробок у галузі конфігурування МСА, зокрема, таких технологій контейнеризації та оркестрації МСА як Docker і Kubernetes, засобів моніторингу їх стану Prometheus, Grafana, Nagios та інших, показано, що недостатньо опрацьованими залишаються питання підвищення ефективності таких процесів за рахунок адаптованого управління конфігураціями у реальному масштабі часу та без необхідності перезавантаження системи в цілому, що зумовлює актуальність цього дисертаційного дослідження. Розділ закінчується визначенням мети та основної задачі дослідження, що полягає в розробці модельно-технологічного інструментарію, який дозволить автоматизувати та адаптувати управління конфігураціями ПС з МСА на різних етапах їх ЖЦ.

У другому розділі дисертації напрацьовані методологічні основи розробки модельно-технологічного інструментарію (МТІ) для забезпечення адаптивного управління програмними мікросервісами. Запропоновано розглядати доменне моделювання як концептуальну основу для подальшого аналізу процесів конфігурування МСА, виконано порівняльний огляд моделей та методів адаптивного управління, оглянуто інструментальні засоби для підтримки інфраструктури функціонування МСА та запропоновано інтелектуальний підхід до вирішення задачі адаптивного управління на основі методу аналізу прецедентів.

Розглянуто й мотивовано обрано можливість застосування інтелектуального підходу до управління функціонуванням мікросервісів з використання методу аналізу прецедентів (Case-Based Reasoning - CBR), який дозволяє використовувати минулий досвід розробників ПС з МСА для вирішення нових задач шляхом пошуку та налаштування інфраструктурних параметрів їх функціонування, які вже були застосовані у схожих ситуаціях, забезпечуючи таким чином адаптивне управління конфігураціями МСА в реальному масштабі часу.

У третьому розділі дисертації розроблено алгоритмічну модель (АМ) адаптивного управління конфігураціями програмних мікросервісів на основі методу аналізу прецедентів, яка забезпечує автоматизований вибір та коригування параметрів конфігурації МСА залежно від поточного стану середовища їх виконання.

Запропонована АМ передбачає формалізацію процесу ідентифікації типових ситуацій функціонування мікросервісів на основі багатовимірного опису контексту, що включає показники навантаження, використання ресурсів, параметри продуктивності та стабільності. Для кожної ситуації здійснюється пошук релевантних прецедентів у базі кейсів, адаптація знайдених рішень та формування рекомендацій щодо зміни конфігураційних параметрів. При цьому у запропонованій АМ передбачено застосування, програмна реалізація та в подальшому порівняльний аналіз результатів застосування 5 різних методів CBR, зокрема: метод k найближчих сусідів, метод зважених k найближчих сусідів, пошук на основі ознак, пошук на основі кластеризації, а також пошук потрібних прецедентів із індексацією та хешуванням кейсів. Це дозволило оцінити компроміси між швидкодією, масштабованістю та ступенем пояснювальності прийнятих рішень.

Для підтримки АМ розроблено концептуальну модель багатовимірного інформаційного базису, яка забезпечує уніфікований опис станів мікросервісної системи, конфігураційних параметрів та результатів їх застосування. На основі цієї моделі спроектовано архітектуру інструментального засобу адаптивного управління конфігураціями, що інтегрується з існуючими платформами контейнеризації та оркестрації. Запропоновані модельні рішення забезпечують можливість оперативної адаптації конфігурацій МСА без перезавантаження системи, підвищують стабільність роботи мікросервісів у змінному середовищі їх виконання та створюють основу для подальшого можливого розвитку інтелектуальних механізмів управління МСА.

Четвертий розділ дисертації присвячено програмній реалізації розроблених моделей та інструментальних засобів, а також експериментальному дослідженню показників якості процесів адаптивного управління конфігураціями програмних мікросервісів із використанням CBR-підходу.

Для проведення експериментів розроблено тестовий програмний застосунок (полігон) з мікросервісною архітектурою у предметній області електронної комерції, який складається з 3 функціонально незалежних мікросервісів, що взаємодіють між собою через стандартизовані API з забезпеченням гнучкості та масштабованості системи. Застосунок розгорнуто у контейнеризованому середовищі з використанням сучасних засобів оркестрації та моніторингу, таких як Docker та Kubenetes. Інструментальний засіб адаптивного управління конфігураціями включає модуль збору та аналізу телеметричних даних, модуль CBR-прийняття рішень та модуль автоматичного застосування конфігурацій. Збір показників продуктивності, використання ресурсів та стабільності здійснювався за допомогою систем моніторингу, а результати експериментів агрегуються та зберігаються у вигляді структурованих записів.

Модуль збору та аналізу даних реалізований на основі інструментів (Prometheus) для обчислення метрик оцінки стану MCA та збирає дані про продуктивність, доступність та інші показники якості функціонування кожного мікросервісу. Модуль управління конфігураціями використовує вищезазначені методи CBR для адаптивного налаштування конфігурацій MCA з механізмом для автоматичного застосування цих змін у конфігураціях тестового полігону. Модуль візуалізації та моніторингу реалізований за допомогою інструментального засобу Grafana для надання зручного інтерфейсу для моніторингу стану системи та відображення ключових метрик.

Розроблена методика експериментального дослідження передбачала порівняння роботи мікросервісної системи у режимі статичних конфігурацій та у режимі адаптивного CBR-керування при різних сценаріях навантаження

та змін середовища виконання. Методика включала: визначення та вибір тестових конфігурацій для кожного мікросервісу, включаючи варіанти з різними параметрами налаштувань та ресурсів; налаштування середовища тестування, яке імітувало реальні умови експлуатації, проведення тестів на продуктивність, стійкість до збоїв, масштабованість та адаптивність.

Результати експериментів підтвердили доцільність застосування CBR-підходу для адаптивного управління конфігураціями мікросервісів і показали, що запропоновані алгоритмічні та інструментальні рішення забезпечують покращення продуктивності й стійкості системи у порівнянні з традиційними статичними підходами до конфігурування та можливість підбору конфігурації за час $\leq 0{,}5$ с навіть при зростанні бази прецедентів до 1000 записів.

В роботі реалізовано та експериментально досліджено п’ять варіантів CBR-методів. Оцінено часові витрати цих методів у діапазоні розмірів БД прецедентів у діапазоні [50-1000] записів. Встановлено, що найвищу швидкодію продемонстрував Indexing and Hashing (орієнтовно 27,6–50,3 мс), KNN показав лінійне зростання часу з обсягом кейс-бази, а Weighted KNN забезпечив більшу керованість за рахунок вагових коефіцієнтів для окремих параметрів опису прецедентів. Застосування CBR дозволяє зменшити середній час відгуку системи на 10–22% порівняно зі статичними конфігураціями.

Таким чином у дисертаційній роботі поставлена та вирішена актуальна науково-прикладна задача підвищення якості процесів розробки та супроводу ПС з МСА шляхом адаптивного управління їх конфігураціями за допомогою запропонованої алгоритмічної моделі, в якій використані методи логічного висновку на основі аналізу прецедентів.

Наукова новизна отриманих результатів полягає у наступному:

– *вперше*: запропонована алгоритмічна модель, яка на відміну від існуючих підходів, забезпечує адаптивне управління конфігураціями програмних мікросервісів на основі використання методів аналізу прецедентів та багатовимірного інформаційного базису, що дозволяє підвищити якість

процесів проектування, розгортання та супроводу програмних систем з мікросервісною архітектурою;

– *удосконалено:* інтелектуальну інформаційну технологію управління програмними мікросервісами за рахунок інтеграції вже існуючих інструментальних засобів із застосуванням блоку адаптивного управління їх конфігураціями, що дозволяє зменшити час, необхідний на отримання нових конфігурацій та дає можливість динамічного керування мікросервісами;

– *отримали подальший розвиток*: методи визначення якості процесу управління конфігураціями мікросервісів шляхом врахування як статичних так і динамічних показників якості їх функціонування із застосуванням набору визначених для цього кількісних метрик.

Тема дослідження безпосередньо пов’язана з виконанням ініціативної НДР МОН України «Концептуальні моделі, методи та технології створення адаптивних інформаційних систем на основі знання-орієнтованих підходів та засобів розробки програмного забезпечення» (№ ДР: 0121U110310). На кафедрі інтелектуальних програмних систем і технологій ННІ комп’ютерних наук та штучного інтелекту Харківського національного університету імені В. Н. Каразіна. Також ці результати використовуються для оновлення методичного забезпечення для лабораторних та практичних робіт студентів за спеціальністю F2 - Комп’ютерні науки у дисциплінах «Розробка сервіс-орієнтованих програмних систем», «Проектування розподілених інформаційних систем» та ін.

Ключові слова: *інтелектуальний підхід, програмне забезпечення, системна архітектура, алгоритмічна модель, мікросервіс, управління конфігураціями, адаптація, метод аналізу прецедентів (CBR), інструментальний засіб, метрика, якість, UML.*

ABSTRACT

Zinov'ev D. V. Intelligent Models and Instrumental Tools for Adaptive Configuration Management of Software Microservices - Qualification scientific work as a manuscript.

Dissertation submitted for the degree of Doctor of Philosophy in specialty 122 Computer Science (Field of Study 12 Information Technologies), V. N. Karazin Kharkiv National University, Ministry of Education and Science of Ukraine, Kharkiv, 2026.

The dissertation is devoted to solving a relevant scientific and technical problem of improving the quality of development and maintenance processes of software systems (SS) with a microservice architecture (MSA) through adaptive configuration management using a knowledge-oriented multidimensional information model to describe the infrastructure of the operational environment of MSA-based software systems, combined with logical inference methods based on case-based reasoning.

The introduction substantiates the relevance of the selected research topic and demonstrates its connection with the scientific activities of the department where the dissertation was carried out. The main goal is formulated, and the object, subject, and research methods are defined. The methods include principles of applied systems analysis, mathematical apparatus of general set theory, domain ontology construction, logical inference methods based on case-based reasoning, object-oriented analysis and software synthesis using the Unified Modeling Language (UML) for formalizing the design procedures of software tools, as well as statistical data processing methods for analyzing experimental results.

The scientific novelty and practical significance of the obtained results are described, along with information about the author's personal contribution to co-authored publications, the approbation of research results at conferences and seminars, and details regarding the structure and scope of the dissertation.

The first chapter provides an analytical review of modern developments in the field of MSA configuration, including containerization and orchestration

technologies such as Docker and Kubernetes, as well as monitoring tools such as Prometheus, Grafana, and Nagios. It is shown that issues related to improving the efficiency of these processes through adaptive configuration management in real time, without requiring a full system restart, remain insufficiently addressed. This determines the relevance of the dissertation. The chapter concludes with the formulation of the research goal and the main objective, which is to develop a model-based technological toolkit that enables automated and adaptive configuration management of MSA-based software systems at different stages of their lifecycle.

The second chapter develops the methodological foundations for designing a model-based technological toolkit (MTT) to support adaptive management of software microservices. Domain modeling is proposed as a conceptual basis for further analysis of MSA configuration processes. A comparative review of adaptive management models and methods is conducted, existing infrastructure support tools are analyzed, and an intelligent approach to solving the adaptive management problem based on case-based reasoning is proposed.

The feasibility of applying an intelligent approach to microservice operation management using case-based reasoning (CBR) is justified. This approach enables the use of past experience from MSA software developers to solve new problems by retrieving and adapting infrastructure configuration parameters that were successfully applied in similar situations, thereby ensuring real-time adaptive configuration management.

In the third chapter, an algorithmic model (AM) for adaptive configuration management of software microservices based on the CBR method is developed. This model provides automated selection and adjustment of configuration parameters depending on the current state of the execution environment.

The proposed model formalizes the process of identifying typical operational situations of microservices based on a multidimensional context description, including workload indicators, resource utilization, performance, and stability parameters. For each situation, relevant cases are retrieved from the case base, adapted, and used to generate recommendations for modifying configuration

parameters. The model incorporates the implementation and comparative analysis of five CBR methods: k-nearest neighbors (KNN), weighted KNN, feature-based retrieval, cluster-based retrieval, and indexing and hashing-based retrieval. This enables evaluation of trade-offs between performance, scalability, and explainability of decision-making.

To support the algorithmic model, a conceptual model of a multidimensional information basis is developed, providing a unified representation of microservice system states, configuration parameters, and their outcomes. Based on this model, the architecture of a configuration management tool is designed, which integrates with existing containerization and orchestration platforms. The proposed solutions enable dynamic configuration adaptation without system restart, improve system stability under changing execution conditions, and establish a foundation for further development of intelligent management mechanisms.

The fourth chapter is devoted to the software implementation of the developed models and tools, as well as to the experimental evaluation of adaptive configuration management processes using the CBR approach.

A test application with a microservice architecture in the e-commerce domain was developed, consisting of three functionally independent microservices interacting via standardized APIs, ensuring flexibility and scalability. The application is deployed in a containerized environment using Docker and Kubernetes. The adaptive configuration management tool includes modules for telemetry data collection and analysis, CBR-based decision-making, and automatic configuration application. Performance, resource usage, and stability metrics are collected using monitoring systems and stored as structured records.

The data collection and analysis module is implemented using Prometheus for computing system state metrics. The configuration management module applies CBR methods for adaptive configuration tuning and includes mechanisms for automatic deployment of configuration changes. The visualization and monitoring module is implemented using Grafana to provide an intuitive interface for system monitoring.

The experimental methodology involves comparing system performance under static configurations and adaptive CBR-based management across various workload scenarios and environmental changes. The methodology includes selecting test configurations, setting up a realistic test environment, and conducting experiments on performance, fault tolerance, scalability, and adaptability.

Experimental results confirm the effectiveness of the CBR-based approach, demonstrating improved system performance and stability compared to traditional static configuration methods. The system is capable of selecting configurations within ≤ 0.5 seconds even with a case base of up to 1000 entries.

Five CBR methods were implemented and experimentally evaluated. Their computational performance was analyzed across case base sizes ranging from 50 to 1000 entries. The indexing and hashing method showed the highest performance (approximately 27.6–50.3 ms), while KNN exhibited linear growth in computation time with increasing case base size. Weighted KNN provided improved controllability due to the use of feature weighting. The application of CBR reduced average system response time by 10–22% compared to static configurations.

Thus, the dissertation solves an important scientific and applied problem of improving the quality of development and maintenance processes of MSA-based software systems through adaptive configuration management using the proposed algorithmic model based on case-based reasoning.

The scientific novelty of the obtained results is as follows:

– *For the first time***:** an algorithmic model has been proposed which, unlike existing approaches, provides adaptive configuration management of software microservices based on case-based reasoning methods and a multidimensional information basis, thereby improving the quality of design, deployment, and maintenance processes of software systems with a microservices architecture;

– *Improved:* an intelligent information technology for managing software microservices through the integration of existing instrumental tools with an adaptive configuration management module, which reduces the time required to obtain new configurations and enables dynamic control of microservices;

– *Further developed:* methods for assessing the quality of microservice configuration management processes by taking into account both static and dynamic performance indicators through the application of a defined set of quantitative metrics.

The research is directly related to the implementation of the initiative research project of the Ministry of Education and Science of Ukraine “Conceptual models, methods, and technologies for creating adaptive information systems based on knowledge-oriented approaches and software development tools” (state registration number: 0121U110310), carried out at the Department of Intelligent Software Systems and Technologies of the Educational and Scientific Institute of Computer Science and Artificial Intelligence of V. N. Karazin Kharkiv National University. The results are also used to update teaching materials for laboratory and practical classes in Computer Science programs.

Keywords: *intelligent approach, software, system architecture, algorithmic model, microservice, configuration management, adaptation, case-based reasoning (CBR), software tool, metric, quality, UML.*

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у наукових фахових виданнях України:

1. Зінов'єв Д.В., Ткачук М.В. Аналіз, класифікація та тестування інструментальних засобів для управління конфігураціями програмних мікросервісів. *Вісник Харківського національного університету імені В. Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління».* 2023. Вип. 57. С.33-42.

DOI: https://doi.org/10.26565/2304-6201-2023-57-03

Ключові слова: програмні мікросервіси, управління конфігураціями, класифікація, тестування, адаптація, якість, метрика, інформаційна технологія, UML, діаграма компонентів, модель, засіб.

URL: https://periodicals.karazin.ua/mia/article/view/23251/21305

(Особистий внесок здобувача: запропоновано інформаційну технологію адаптивного управління процесом конфігурування МСА у вигляді компонентної діаграми, зроблена класифікація інструментальних засобів управління конфігураціями мікросервісів, розроблено тестовий мікросервісний застосунок, проведене тестування засобу Microconfig.io, показано принципову можливість створення конфігураційних файлів для управління конфігураціями МСА.

Особистий внесок Миколи Ткачука: концептуалізація основних проблем дисертаційного дослідження, постановка задачі аналізу існуючих підходів до управління конфігураціями МСА, структурування та редагування тексту, формулювання напрямків подальших досліджень.)

2. Ткачук М. В., Зінов'єв Д.В. Розробка та дослідження алгоритмічної моделі для адаптивного управління конфігураціями програмних мікросервісів. *Системи обробки інформації*. 2024. № 2(177).С. 107–111.

DOI: https://doi.org/10.30748/soi.2024.177.12

Ключові слова: алгоритм, модель, програмний мікросервіс, управління конфігураціями, адаптація, метод аналізу прецедентів, метрика, продуктивність, хмарні сервіси.

URL: https://journal-hnups.com.ua/index.php/soi/article/view/1663/1530

(Особистий внесок здобувача: розроблена алгоритмічна модель адаптивного управління мікросервісами із застосуванням методу CBR, зроблений формальний опис цієї моделі з використанням апарату теорії множин, спроектована структурно-функціональна схема інструментального засобу, виконана реалізація програмного прототипу засобу із застосуванням стеку технологій JavaScript (JS), Node.js і Serverless Framework, а також хмарних сервісів Amazon Web Services., проведені обчислювальні експерименти з тестовою базою прецедентів, зроблене порівняння результатів з альтернативними програмними рішеннями, такими як myCBR і jCOLIBRI.

Особистий внесок Миколи Ткачука: рекомендації по вибору типу моделі управляння МСА, оцінка коректності запропонованої методики обчислювальних експериментів, перевірка та редагування тексту статті.)

3. Зінов'єв Д.В., Ткачук М. В. Архітектура, програмна реалізація та аналіз результатів застосування інтелектуального інструментального засобу для конфігурування мікросервісних застосунків. *Вісник Харківського національного університету імені В.Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління».* 2025. Вип. 67. С.56-66.

DOI: https://doi.org/10.26565/2304-6201-2025-67-05

Ключові слова: програмний мікросервіс, архітектура, управління конфігураціями, інтелектуальний підхід, метод аналізу прецедентів, CBR, інтелектуальний інструментальний засіб, тестування, якість, метрика, модель.

URL: https://periodicals.karazin.ua/mia/article/view/28383/24799

(Особистий внесок здобувача: розроблена архітектура інтелектуального інструментального засобу для адаптивного управління

конфігураціями МСА з модулем прийняття рішень на основі Case-Based Reasoning (CBR), побудована ER-модель БД прецедентів, реалізовано веб-інтерфейс для моніторингу ручного та автоматичного конфігурування МСА, спроектована архітектура і реалізовано програмний тестовий полігон для проведення експериментальних досліджень, порівняна ефективність використання 5 CBR-методів.

Особистий внесок Миколи Ткачука: участь у розробці методики проведення обчислювальних експериментів, перевірка та редагування тексту статті, корекція формулювання напрямків подальших досліджень.)

Наукові праці, які засвідчують апробацію матеріалів дисертації:

1. Зінов'єв Д.В., Ткачук М.В.. Тріщенко І.В. Моделі та технології забезпечення якості сервіс-орієнтованих програмних систем: сучасний стан та перспективні напрямки досліджень. *Матеріали міжн. науков.-техн. конф. КМНТ-2021 (м. Харків, 23-25 квітня 2021 року).* Х.: ХНУ імені В.Н. Каразіна, 2021. С. 166-169.
2. Зінов'єв Д.В., Ткачук М.В. Розробка інструментального засобу для автоматизованої оцінки показників якості мікросервісних застосунків. *Стан, досягнення та перспективи інформаційних систем і технологій», XXIII Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів*. Одеса, 20-21 квітня 2023 р. Одеса, ОНТУ, 2023 р. С. 239-240.
3. Tkachuk M.V., Zinoviev D.V. A Case-based Reasoning Approach to Quality Assurance in Microservice Software Systems. *Інформаційні технології: наука, техніка, технологія, освіта, здоров'я: Тези доповідей XXXI міжнародної науково-практичної конференції MicroCAD-2023, 17-20 травня 2023 р.,* / за ред. проф. Сокола Є.І. Харків, НТУ «ХПІ». С.1034.
4. Зінов'єв Д.В., Ткачук М.В. До питання побудови адаптивного механізму управління програмними мікросервісами із застосуванням методу аналізу прецедентів. *Матеріали міжн. науков.-техн. конференції КМНТ-2023*

(м. Харків, 25-27 жовтня 2023 року). Х.: ХНУ імені В.Н. Каразіна, 2023. С. 72-74.

5. Зінов'єв Д.В. Розробка концептуальної моделі багатовимірного інформаційного простору для управління конфігураціями мікросервісних застосунків. *«Стан, досягнення та перспективи інформаційних систем і технологій», XXIV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів.* Одеса, 18-19 квітня 2024 р. Одеса, ОНТУ, 2024 р. С. 235-236.

6. Зінов'єв Д. В., Ткачук М. В. Порівняльний аналіз особливостей застосування методів аналізу прецедентів для управління конфігураціями програмних мікросервісів. *Матеріали міжн. науков.-техн. конференції ІТМД-2025. (м. Харків, 12-14 листопада 2025 року).* Х.: ХНУ імені В.Н. Каразіна, 2025. С. 123-126.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Скорочення українською мовою

АУКМ - Адаптивне управління конфігураціями мікросервісів

БД - База даних

ІТ - Інформаційні технології

ІТЗ - Інструментальний засіб

КС - Конфігураційна система

МСА - Мікросервісна архітектура

ПЗ - Програмне забезпечення

РПС - Розподілена програмна система

Скорочення англійською мовою

API - Application Programming Interface

AWS - Amazon Web Services

CBR - Case-Based Reasoning

CPU - Central Processing Unit

DevOps - Development And Operations

ER - Entity-Relationship

FBR - Feature-Based Retrieval

IDEF0 - Integration Definition For Function Modeling

JSON - JavaScript Object Notation

KNN - K-Nearest Neighbors

OC - Operational Costs

QoS - Quality Of Service

RAM - Random Access Memory

REST - Representational State Transfer

UML - Unified Modeling Language

ВСТУП

Обґрунтування вибору теми дослідження та актуальність роботи. Сучасний етап розвитку інформаційних технологій характеризується широким впровадженням розподілених програмних систем (РПС), побудованих на основі мікросервісної архітектури (МСА) [2, 14, 15, 18]. Такий підхід забезпечує високу масштабованість, гнучкість та відмовостійкість програмних застосунків [3, 15, 17, 24], що є критично важливим для сучасних інформаційних систем, зокрема у сферах електронної торгівлі, фінансових технологій, телекомунікацій та хмарних сервісів [41-45].

Разом із тим, використання МСА суттєво ускладнює процеси конфігурування, розгортання та супроводу програмних систем [1, 19, 37]. Збільшення кількості мікросервісів, їх динамічна взаємодія, необхідність обробки великої кількості запитів та зміна умов виконання призводять до зростання складності управління конфігураціями таких систем [34, 39]. Наявні інструментальні засоби, зокрема засоби контейнеризації, оркестрації та централізованого керування конфігураціями, забезпечують лише часткову автоматизацію цих процесів і не враховують повною мірою багатовимірний характер показників якості функціонування систем [5, 8, 36, 38-40].

Аналіз існуючих підходів показав, що більшість з них орієнтовані на статичні або напівавтоматичні механізми конфігурування і не забезпечують адаптивного прийняття рішень на основі поточного стану системи [26-28, 72, 73]. У зв’язку з цим виникає необхідність розробки інтелектуальних методів і засобів, здатних забезпечити автоматичний вибір оптимальних конфігурацій з урахуванням змін умов функціонування, ресурсних обмежень та показників якості обслуговування [1, 2, 70, 72, 76].

Перспективним підходом до розв’язання цієї задачі є застосування методів аналізу прецедентів (Case-Based Reasoning, CBR), які дозволяють використовувати накопичений досвід функціонування системи для прийняття рішень у нових ситуаціях. Використання CBR у поєднанні з багатовимірним

інформаційним базисом створює передумови для побудови адаптивних систем управління конфігураціями мікросервісів [6, 8, 10, 13, 69].

Таким чином, задача розробки інтелектуальних моделей і інструментальних засобів для адаптивного управління конфігураціями програмних мікросервісів є актуальною науково-прикладною задачею, що має важливе значення для підвищення ефективності функціонування сучасних розподілених програмних систем.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконана відповідно до наукового напряму кафедри інтелектуальних програмних систем і технологій Харківського національного університету імені В. Н. Каразіна та пов'язана з виконанням прикладної науково-дослідної роботи Міністерства освіти і науки України «Концептуальні моделі, методи та технології створення адаптивних інформаційних систем на основі знання-орієнтованих підходів та засобів розробки програмного забезпечення» (№ ДР: 0121U110310).

Мета і задачі дослідження. Метою дисертаційної роботи є підвищення якості процесів конфігурування, розміщення та супроводу розподілених програмних систем з мікросервісною архітектурою шляхом побудови модельно-технологічного інструментарію для адаптивного управління конфігураціями компонентів таких систем.

Для досягнення поставленої мети у роботі сформульовано та вирішено такі задачі:

1. Виконати аналіз сучасних проблем розробки та супроводу розподілених програмних систем з мікросервісною архітектурою.
2. Провести огляд існуючих інструментальних засобів управління конфігураціями мікросервісів та побудувати їх класифікацію.
3. Розробити концептуальну модель багатовимірного інформаційного простору для підтримки процесів адаптивного конфігурування.
4. Запропонувати алгоритмічну модель адаптивного управління конфігураціями МСА на основі методу аналізу прецедентів.

5. Дослідити та порівняти різні методи вилучення прецедентів.
6. Спроєктувати архітектуру інтелектуального інструментального засобу управління конфігураціями мікросервісів.
7. Реалізувати програмний прототип системи адаптивного управління конфігураціями.
8. Провести експериментальні дослідження ефективності запропонованого підходу.

Об'єкт дослідження. Процеси розробки та супроводу розподілених програмних систем з мікросервісною архітектурою.

Предмет дослідження. Інтелектуальні моделі та інструментальні засоби адаптивного управління конфігураціями програмних мікросервісів з урахуванням показників якості їх функціонування.

Методи дослідження. У роботі використано принципи системного аналізу, методи теорії автоматичного управління, математичний апарат теорії множин, методи аналізу прецедентів, а також методи сучасної програмної інженерії, включаючи доменне моделювання, об'єктно-орієнтований аналіз і синтез програмного забезпечення. Для оцінювання ефективності запропонованого підходу застосовано методи статистичного аналізу експериментальних даних. Для моделювання програмних рішень використано мову UML та нотацію IDEF0.

Наукова новизна отриманих результатів.

Наукова новизна одержаних результатів полягає в наступному:

вперше: запропонована алгоритмічна модель, яка на відміну від існуючих підходів, забезпечує адаптивне управління конфігураціями програмних мікросервісів на основі використання методів аналізу прецедентів та багатовимірного інформаційного базису, що дозволяє підвищити якість процесів проектування, розгортання та супроводу програмних систем з мікросервісною архітектурою;

удосконалено: інтелектуальну інформаційну технологію управління програмними мікросервісами за рахунок інтеграції вже існуючих

інструментальних засобів із застосуванням блоку адаптивного управління їх конфігураціями, що дозволяє зменшити час, необхідний на отримання нових конфігурацій та дає можливість динамічного керування мікросервісами;

отримали подальший розвиток: методи визначення якості процесу управління конфігураціями мікросервісів шляхом врахування як статичних так і динамічних показників якості їх функціонування із застосуванням набору визначених для цього кількісних метрик.

Особистий внесок здобувача. Дисертаційне дослідження виконано здобувачем самостійно, при цьому всі основні наукові положення, моделі, методи та практичні результати, що виносяться на захист, отримані особисто автором. У наукових працях, опублікованих у співавторстві, здобувачу належить визначальний внесок, який полягає у постановці задач дослідження, розробці теоретичних положень, програмній реалізації запропонованих рішень та аналізі отриманих результатів.

Зокрема, особистий внесок здобувача включає такі ключові результати:

1) У межах дослідження сучасного стану проблеми управління конфігураціями мікросервісів:

– виконано системний аналіз та класифікацію існуючих інструментальних засобів управління конфігураціями МСА ;

– запропоновано інформаційну технологію адаптивного управління конфігураціями, представлену у вигляді компонентної моделі ;

– розроблено тестовий мікросервісний застосунок для експериментального дослідження;

– проведено експериментальне тестування інструментального засобу Microconfig.io та показано можливість автоматизованого формування конфігураційних рішень .

2) У частині розробки математичного та алгоритмічного забезпечення:

– розроблено алгоритмічну модель адаптивного управління конфігураціями мікросервісів на основі методу Case-Based Reasoning (CBR) [3, 20–23];

– виконано формалізацію моделі з використанням апарату теорії множин, що забезпечило її строгий теоретичний опис;

– спроєктовано структурно-функціональну модель інтелектуального інструментального засобу, яка реалізує повний цикл адаптивного управління .

3) У частині програмної реалізації:

– розроблено програмний прототип системи адаптивного управління конфігураціями мікросервісів із використанням стеку технологій JavaScript (Node.js, Serverless Framework) та хмарної інфраструктури Amazon Web Services;

– реалізовано модуль прийняття рішень на основі CBR, що забезпечує автоматичний підбір конфігурацій ;

– побудовано базу прецедентів та ER-модель її структури, що підтримує зберігання багатовимірних характеристик стану системи;

– створено веб-інтерфейс для моніторингу та керування конфігураціями як у ручному, так і в автоматичному режимах;

– спроєктовано та реалізовано тестовий полігон на основі мікросервісної архітектури, який використовується для проведення експериментальних досліджень.

4) У частині експериментальних досліджень:

– проведено обчислювальні експерименти з використанням тестової бази прецедентів;

– реалізовано та досліджено п’ять методів вилучення прецедентів (KNN, Weighted KNN, Feature-Based Retrieval, Cluster-Based Retrieval, Indexing & Hashing);

– виконано порівняльний аналіз ефективності запропонованого підходу з існуючими рішеннями (зокрема myCBR та jCOLIBRI);

– отримано кількісні результати, що підтверджують ефективність запропонованих моделей і методів .

5) У частині узагальнення результатів:

– виконано аналіз отриманих експериментальних даних [29-31];

– сформульовано висновки та практичні рекомендації щодо використання різних CBR-методів у задачах адаптивного управління конфігураціями мікросервісів;

– забезпечено узгодження теоретичних моделей із результатами програмної реалізації та експериментальної перевірки .

Усі наведені результати є логічно пов’язаними між собою та утворюють цілісну наукову роботу, що охоплює повний цикл дослідження — від аналізу предметної області та побудови моделей до створення програмного засобу та його експериментальної перевірки.

Практичне значення отриманих результатів. Практичне значення роботи полягає у створенні інструментального засобу адаптивного управління конфігураціями мікросервісів, який може бути використаний при розробці та супроводі сучасних розподілених програмних систем .

Результати дисертаційного дослідження можуть бути безпосередньо застосовані у сфері електронної комерції, де мікросервісні архітектури є стандартом для побудови високонавантажених систем обробки користувацьких запитів. Зокрема, розроблений підхід до адаптивного управління конфігураціями може бути використаний для динамічного налаштування параметрів сервісів автентифікації, каталогів товарів і обробки замовлень залежно від змін інтенсивності навантаження. Це дозволяє забезпечити стабільну роботу системи під час пікових періодів (наприклад, акцій чи розпродажів), зменшити затримки відповіді та підвищити доступність сервісів без надлишкового використання обчислювальних ресурсів.

Важливою прикладною областю є також хмарні обчислювальні платформи та DevOps-інфраструктури, де задачі автоматизованого конфігурування та масштабування мікросервісів є критичними. Запропонований інструментальний засіб може бути інтегрований у системи оркестрації (наприклад, Kubernetes-подібні середовища) як модуль інтелектуального прийняття рішень, що дозволяє не лише виконувати

масштабування, а й адаптивно змінювати конфігураційні параметри сервісів на основі аналізу метрик і накопиченого досвіду. Це особливо актуально для хмарних сервісів із змінним навантаженням, де ефективне використання ресурсів безпосередньо впливає на вартість експлуатації.

Крім того, результати можуть бути використані у фінансових інформаційних системах, телекомунікаційних платформах та системах реального часу, де висуваються підвищені вимоги до надійності, швидкодії та безперервності обслуговування. У таких системах адаптивне управління конфігураціями дозволяє оперативно реагувати на зміну умов функціонування, забезпечуючи оптимальний баланс між продуктивністю, витратами ресурсів і якістю обслуговування . Запропонований підхід також може бути використаний як основа для побудови інтелектуальних систем підтримки прийняття рішень у задачах супроводу складних розподілених систем.

Результати дослідження впроваджені в межах прикладної науково-дослідної роботи МОН України, а також використовуються у навчальному процесі при викладанні дисциплін, пов’язаних із розробкою сервіс-орієнтованих програмних систем.

Апробація результатів дисертації. Основні положення дисертаційної роботи доповідалися та обговорювалися на засіданнях кафедри інтелектуальних програмних систем і технологій та наукових семінарах ННІ комп'ютерних наук та штучного інтелекту Харківського національного університету імені В.Н. Каразіна. Ключові положення дослідження оприлюднені у доповідях на міжнародних і всеукраїнських науково-технічних конференціях протягом 2021 – 2025 років, зокрема :

– взяв участь у Міжнародній науково-технічній конференції «Комп'ютерне моделювання в наукоємних технологіях -2021», підготував тези доповіді, в яких висвітлюються деякі результати дослідження : Зінов’єв Д.В., Ткачук М.В.. Тріщенко І.В. Моделі та технології забезпечення якості сервіс-орієнтованих програмних систем: сучасний стан та перспективні напрямки

досліджень // Матеріали міжн. науков.-техн. конф. КМНТ-2021 (м. Харків, 23-25 квітня 2021 року) – Х.: ХНУ імені В.Н. Каразіна, 2021. – С. 166-169.

– взяв участь у Всеукраїнській науково-технічній конференції молодих вчених, аспірантів та студентів «Стан, досягнення та перспективи інформаційних систем і технологій», підготував тези доповіді, в яких висвітлюються деякі результати дослідження : Зінов'єв Д.В., Ткачук М.В. Розробка інструментального засобу для автоматизованої оцінки показників якості мікросервісних застосунків // «Стан, досягнення та перспективи інформаційних систем і технологій», XXIII Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів. Одеса, 20-21 квітня 2023 р. - Одеса, ОНТУ, 2023 р. – с. 239-240.

– взяв участь у Міжнародній науково-практичній конференції «MicroCAD-2023», підготував тези доповіді, в яких висвітлюються деякі результати дослідження : Tkachuk M.V., Zinoviev D.V. A Case-based Reasoning Approach to Quality Assurance in Microservice Software Systems // Інформаційні технології: наука, техніка, технологія, освіта, здоров'я: Тези доповідей XXXI міжнародної науково-практичної конференції MicroCAD-2023, 17-20 травня 2023р., / за ред. проф. Сокола Є.І. – Харків, НТУ «ХПІ». – С.1034.

– Взяв участь у Міжнародній науково-технічній конференції «Комп'ютерне моделювання в наукоємних технологіях - 2023», підготував тези доповіді, в яких висвітлюються деякі результати дослідження : Зінов'єв Д.В., Ткачук М.В. До питання побудови адаптивного механізму управління програмними мікросервісами із застосуванням методу аналізу прецедентів //Матеріали міжн. науков.-техн. конференції КМНТ-2023 (м. Харків, 25-27 жовтня 2023 року) – Х.: ХНУ імені В.Н. Каразіна, 2023. – С. 72-74.

– взяв участь у XXIV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів, підготував тези доповіді, в яких висвітлюються деякі результати дослідження : Зінов'єв Д.В. Розробка концептуальної моделі багатовимірного інформаційного простору для управління конфігураціями мікросервісних застосунків // «Стан, досягнення

та перспективи інформаційних систем і технологій», XXIV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів. Одеса, 18-19 квітня 2024 р. - Одеса, ОНТУ, 2024 р. – с. 235-236.

– взяв участь у Міжнародній науково-технічній конференції «ІТМД-2025», підготував тези доповіді, в яких висвітлюються деякі результати дослідження : Зінов'єв Д.В., Ткачук М. В. Порівняльний аналіз особливостей застосування методів аналізу прецедентів для управління конфігураціями програмних мікросервісів // Матеріали міжн. науков.-техн. конференції ІТМД-2025. (м. Харків, 12-14 листопада 2025 року) – Х.: ХНУ імені В.Н. Каразіна, 2025. – С. 123-126.

Публікації. Основні результати дисертаційного дослідження опубліковано у 9 наукових працях, серед яких 3 статті у фахових наукових виданнях категорії В та 6 публікацій у матеріалах міжнародних та всеукраїнських науково-практичних конференціях.

Структура та обсяг дисертації.

Дисертаційна робота складається з вступу, чотирьох розділів, висновків, списку використаних джерел і трьох додатків. Загальний обсяг дисертації становить 170 сторінок, у тому числі: анотації на 11 сторінках, зміст на 2 сторінках, основний текст на 134 сторінках, список використаних джерел із 98 найменувань на 9 сторінках та три додатки на 7 сторінках. Робота містить 21 таблицю та 53 рисунки.

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ПРОБЛЕМ РОЗРОБКИ ТА СУПРОВОДУ ПРОГРАМНИХ СИСТЕМ НА ОСНОВІ МІКРОСЕРВІСІВ

1.1 Актуальність проблем розробки та застосування мікросервісної архітектури

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням складності програмних систем, широким впровадженням хмарних обчислень, розвитком розподілених середовищ та необхідністю обробки великих обсягів даних у реальному часі [41, 65, 80, 84, 85]. У таких умовах традиційні підходи до проєктування програмного забезпечення виявляються недостатньо ефективними, що обумовлює необхідність використання нових архітектурних рішень [16, 18, 66].

Як показано на рисунку 1.1, розвиток архітектур програмних систем відбувався у напрямку підвищення автономності компонентів та зменшення їх залежності від монолітної моделі до сервіс-орієнтованої архітектури, далі до мікросервісної архітектури та cloud-native підходу. Схема відображає послідовне зростання автономності компонентів, зменшення централізації управління та посилення ролі контейнеризації, автоматизованого розгортання й оркестрації [23, 34, 36].

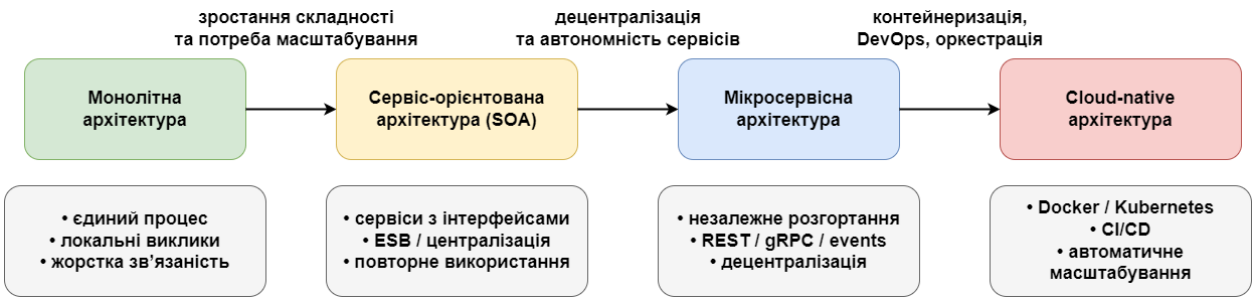


Рисунок 1.1 – Еволюція архітектур програмних систем

Мікросервісна архітектура на сьогоднішній день стала де-факто стандартом для побудови масштабованих і гнучких програмних систем [14, 15,

18, 66]. Її популярність пояснюється здатністю забезпечувати незалежний розвиток компонентів, ефективне використання ресурсів та швидке впровадження змін у програмних продуктах [18, 24]. Разом із тим, перехід до мікросервісної архітектури супроводжується появою нових проблем, пов'язаних із управлінням складними розподіленими системами [19, 67, 73].

Як показано на рисунку 1.2, ключові проблеми мікросервісної архітектури пов'язані з багаторівневою складністю системи, яка включає рівні сервісів, інфраструктури та взаємодії [18, 19, 67].

Рисунок 1.2 – Основні проблеми мікросервісної архітектури

Однією з ключових проблем є *складність управління конфігураціями* [1-3, 37, 39, 40]. У мікросервісних системах кількість конфігураційних параметрів значно зростає, оскільки кожен сервіс має власні налаштування, конфігурації залежать від середовища виконання, параметри можуть змінюватися під час роботи системи [40, 63]. Це призводить до явища *configuration drift*, коли фактичні параметри системи відхиляються від початково заданих [62-64]. У результаті система може працювати нестабільно або не відповідати очікуваним характеристикам. Крім того, значною проблемою є *залежність конфігурацій від контексту виконання*, зокрема: рівня навантаження; доступних ресурсів; характеристик мережі; стану інших сервісів [34, 57, 60].

Традиційні підходи до управління конфігураціями, які базуються на статичних правилах або ручному налаштуванні, не здатні ефективно враховувати ці фактори. Це створює передумови для пошуку нових підходів до управління конфігураціями, орієнтованих на адаптивність [26-28, 70-73].

Обґрунтування необхідності адаптивного управління конфігураціями

Аналіз переваг і обмежень мікросервісної архітектури показує, що конфігурація є одним із ключових елементів, який визначає ефективність функціонування програмної системи [37-40, 46-48]. В умовах розподіленого середовища конфігураційні параметри впливають на:

– продуктивність системи;
– використання ресурсів;
– надійність і доступність сервісів;
– швидкість обробки запитів.

Як показано на рисунку 1.3, необхідність адаптивного управління конфігураціями зумовлена поєднанням кількох факторів.

Рисунок 1.3 – Чинники необхідності адаптивного управління конфігураціями

По-перше, сучасні програмні системи функціонують у *динамічних середовищах*, де параметри навантаження можуть змінюватися в широких

межах. Статичні конфігурації не здатні забезпечити оптимальну роботу системи за таких умов [25, 71, 77].

По-друге, управління конфігураціями має *багатокритеріальний характер*, оскільки необхідно одночасно враховувати декілька показників, таких як продуктивність, вартість використання ресурсів та доступність системи. Це ускладнює процес прийняття рішень [43, 46].

По-третє, у системах накопичується значний обсяг *історичних даних про їх функціонування*, який може бути використаний для підвищення ефективності управління. Однак традиційні підходи не використовують цей потенціал у повній мірі [29, 68].

По-четверте, конфігурації мають складну структуру та взаємозалежності, що ускладнює їх оптимізацію. Зміна одного параметра може впливати на інші, що створює необхідність у використанні більш складних методів аналізу [16, 18, 19, 67].

Сукупність цих факторів показує, що ефективне управління конфігураціями не може ґрунтуватися лише на фіксованих налаштуваннях або попередньо визначених сценаріях. Натомість воно має враховувати зміну стану системи у часі, результати попередніх рішень та поточний контекст функціонування. У зв’язку з цим доцільним є застосування інтелектуальних методів прийняття рішень, які дозволяють враховувати попередній досвід функціонування системи, аналізувати поточний стан середовища, формувати ефективні конфігураційні рішення [26-29, 72, 73].

Одним із перспективних підходів є використання методів аналізу прецедентів (Case-Based Reasoning), які дозволяють приймати рішення на основі попереднього досвіду. Такий підхід забезпечує адаптивність системи та здатність до самонавчання, що є критично важливим у динамічних середовищах [10-13].

Таким чином, необхідність адаптивного управління конфігураціями визначається поєднанням високої складності мікросервісних систем, динамічності середовища та багатокритеріального характеру задачі

оптимізації [67, 73, 77]. Це формує наукову проблему, яка розглядається у даній дисертаційній роботі.

1.2 Особливості життєвого циклу програмних систем на основі мікросервісної архітектури

Сучасні програмні системи, побудовані на основі мікросервісної архітектури, характеризуються високим рівнем динамічності, масштабованості та складності. Це зумовлює необхідність перегляду класичних підходів до організації життєвого циклу програмного забезпечення, оскільки традиційні моделі розробки не враховують специфіку розподілених середовищ, хмарних платформ та безперервної інтеграції змін [18, 19, 23, 24].

Основними обмеженнями класичних моделей у контексті мікросервісної архітектури є:

1) низька адаптивність до змін, коли класичні моделі передбачають фіксацію вимог на початкових етапах, що суперечить сучасним умовам, у яких вимоги змінюються протягом усього життєвого циклу системи [41–45];

2) відсутність підтримки безперервного розгортання, коли для традиційних підходів розгортання здійснюється після завершення розробки, а для мікросервісних системах оновлення можуть відбуватися декілька разів на день [23, 24, 34, 36];

3) *централізований характер процесів*, коли класичні моделі орієнтовані на централізоване управління розробкою, а мікросервісна архітектура передбачає децентралізацію та автономність команд [18, 19, 23];

4) *недостатня увага до експлуатаційного етапу*, коли основна увага приділяється етапу розробки, а експлуатація розглядається як допоміжний процес. У мікросервісних системах експлуатація стає ключовим етапом, який визначає ефективність функціонування системи [20, 21, 59].

5) *обмежені можливості масштабування* для класичних підходів, які не враховують необхідність динамічного масштабування системи, що є критично важливим для сучасних хмарних застосунків [34, 36, 37].

Таким чином, класичні моделі життєвого циклу не відповідають вимогам, які висуваються до сучасних програмних систем, побудованих на основі мікросервісної архітектури. Це обумовлює необхідність переходу до нових підходів, зокрема DevOps, які забезпечують інтеграцію процесів розробки (Development) та експлуатації (Operations). Основною ідеєю DevOps є побудова *безперервного життєвого циклу*, у якому етапи розробки, тестування, розгортання та моніторингу інтегровані в єдиний процес [23, 24].

На рисунку 1.4 представлено узагальнену модель життєвого циклу DevOps, яка відображає безперервний процес розробки, інтеграції, розгортання та експлуатації програмних систем.

Рисунок 1.2 – Життєвий цикл DevOps

Однією з ключових особливостей DevOps є високий рівень автоматизації, який охоплює: збірку програмного коду, тестування, розгортання, управління інфраструктурою [82, 86]. Це дозволяє зменшити вплив людського фактора та забезпечити повторюваність процесів. У контексті мікросервісної архітектури DevOps набуває особливого значення, оскільки:

– кількість компонентів системи значно зростає;

- кожен сервіс має власний життєвий цикл;
- зміни відбуваються значно частіше [34, 36, 37].

Це призводить до необхідності синхронізації процесів між різними сервісами та командами, що ускладнює управління системою.

Крім того, DevOps передбачає використання *контейнеризації та оркестрації*, що забезпечує незалежність середовищ виконання та автоматичне управління ресурсами [34, 36, 37]. Таким чином, DevOps є ключовим фактором трансформації життєвого циклу програмних систем, який забезпечує безперервність процесів і створює передумови для адаптивного управління.

Контейнеризація є ключовою технологією, що забезпечує ізоляцію сервісів і стандартизацію середовища виконання. Вона дозволяє гарантувати однаковість середовищ розробки, тестування та продуктивної експлуатації, спростити перенесення застосунків між різними інфраструктурами, підвищити ефективність використання ресурсів [34, 36, 70, 75].

Зі збільшенням кількості мікросервісів виникає необхідність автоматичного управління їх розміщенням, масштабуванням та взаємодією. Ці функції виконують системи оркестрації, зокрема Kubernetes, які забезпечують:

- автоматичне масштабування (auto-scaling);
- балансування навантаження;
- відновлення сервісів після збоїв;
- управління конфігураціями та секретами [34, 36, 37].

У мікросервісних системах конфігурації є одним із ключових елементів, що визначають поведінку системи на всіх етапах її життєвого циклу - від розробки до експлуатації. Зі зростанням кількості сервісів та складності їх взаємодії управління конфігураціями стає критично важливою задачею [76].

Також, оскільки конфігурації змінюються залежно від середовища виконання (dev, test, production), характеристик інфраструктури та поточного навантаження, для одного і того ж мікросервісу необхідна підтримка різних варіантів конфігурацій [37, 39, 40].

Традиційні методи управління конфігураціями базуються на статичних правилах, заздалегідь визначених сценаріях і ручному налаштуванні і не здатні ефективно реагувати на зміни середовища та не враховують накопичений досвід функціонування системи [26–28, 72, 73]. У результаті виникає необхідність переходу до адаптивних підходів, які дозволяють автоматично змінювати конфігурації залежно від поточного стану системи [10–13, 70].

1.3 Аналіз сучасних підходів до оцінки якості та управління конфігураціями мікросервісних програмних систем

Мікросервісна архітектура (МСА) є одним із домінуючих підходів до побудови сучасних розподілених програмних систем, що забезпечує високу масштабованість, гнучкість та можливість незалежного розвитку компонентів [16, 18, 19, 23, 90]. Разом з тим, перехід від монолітних до мікросервісних систем суттєво ускладнює процес забезпечення якості, оскільки такі системи характеризуються високим рівнем динамічності, розподіленості та неоднорідності середовища виконання [41–45, 65, 66].

У сучасних оглядових дослідженнях підкреслюється, що якість МСА визначається не лише класичними атрибутами (надійність, продуктивність, супроводжуваність), але й такими специфічними характеристиками, як:

- рівень декомпозиції (granularity);
- ступінь зв’язаності (coupling) між сервісами;
- внутрішня згуртованість (cohesion);
- здатність до автоматичного масштабування;
- адаптивність до змін навантаження та середовища [43, 46–48].

Аналогічні висновки отримані і у роботах, де показано, що ключові атрибути якості МСА пов’язані із структурними характеристиками системи та параметрами її функціонування .

Таким чином, якість мікросервісних систем має багатовимірний характер і залежить як від архітектурних рішень, так і від параметрів конфігурації та умов експлуатації [76] (див. рисунок 1.2).

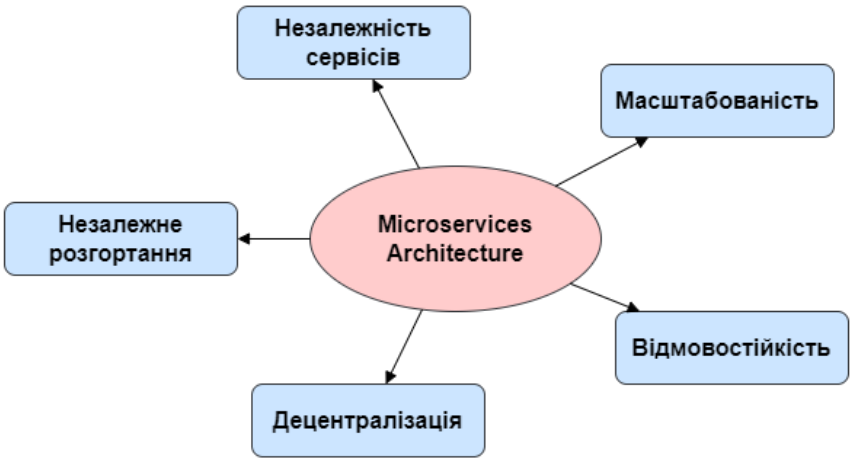


Рисунок 1.5 –Фактори, що впливають на якість мікросервісних систем

Аналіз сучасних досліджень дозволяє виділити декілька основних підходів до оцінки якості мікросервісних систем.

Найбільш поширеним є метрико-орієнтовані підходи де використовуються кількісні метрики, що характеризують окремі аспекти якості (зчеплення, складність сервісів, частота міжсервісних викликів, критичність та важливість сервісів). Однак сучасні оглядові дослідження підкреслюють, що окремі метрики не забезпечують комплексної оцінки, відсутня універсальна система метрик, значна частина підходів орієнтована лише на статичні характеристики системи.

Підходи на основі агрегованих показників пропонують використовувати узагальнені показники (наприклад, maintainability), які формуються на основі набору метрик із застосуванням експертних або нечітких моделей . Недоліками цього підходу є суб’єктивність експертних оцінок, складність калібрування моделей, відсутність врахування показників якості на етапі виконання (runtime).

У сучасних дослідженнях значна увага приділяється підходами до оцінки якості на етапі виконання (runtime-based) з використанням таких показників як: Response Time; Error Rate; CPU та RAM usage; пропускна здатність системи [59–61]. У даній роботі ці підходи реалізовані у вигляді

експериментального полігону, де проводиться порівняння статичних та адаптивних конфігурацій системи.

Підходи до декомпозиції монолітних систем становлять окрему групу, що базуються на аналізі взаємодії компонентів, частоти викликів, структурних залежностей [41–45]. Проте ці підходи орієнтовані переважно на етап проєктування і не вирішують задач адаптивного управління конфігураціями під час експлуатації.

Аналіз літератури показує, що більшість підходів не поєднують runtime та structural метрики, також в них відсутні універсальні моделі інтеграції цих показників [43, 46–48, 76]. У таблиці 1.1 наведено порівняння підходів до оцінки якості МСА

Таблиця 1.1 – Порівняльний аналіз підходів до оцінки якості МСА

Підхід	**Основні характеристики**	**Переваги**	**Недоліки**
Метрико-орієнтований	Використання статичних метрик	Формалізація	Не враховує runtime
Агрегований	Інтегральні показники	Комплексність	Суб’єктивність
Runtime-based	Моніторинг системи	Реалістичність	Відсутність структурного аналізу
Комбінований	Поєднання підходів	Висока точність	Відсутність стандартів

Складність етапів розгортання та супроводу сучасних систем з МСА також значною мірою впливає на якість таких систем [34, 36, 37]. На відміну від монолітних систем, де управління здійснюється централізовано, у МСА виникає необхідність координації великої кількості автономних компонентів, що функціонують у розподіленому середовищі [18, 19, 23].

У цьому контексті процеси супроводу та управління програмними системами набувають системного характеру і базуються на поєднанні кількох

підходів, які відрізняються рівнем автоматизації, адаптивності та використання даних [70, 72, 73].

На основі аналізу сучасних досліджень та практичних реалізацій доцільно виділити такі узагальнені підходи до організації супроводу МСА.

1) *Централізовано-керований підхід*. Підхід передбачає використання єдиного центру управління конфігураціями, розгортанням і моніторингом системи. У цьому випадку всі зміни конфігурацій здійснюються через централізовані сервіси або адміністраторів. Такий підхід забезпечує контрольованість і передбачуваність, однак погано масштабується та не враховує динаміку середовища [26–28].

2) *Декларативно-орієнтований підхід*. Цей підхід базується на описі бажаного стану системи у вигляді декларацій (наприклад, конфігураційних файлів), після чого система автоматично приводиться у відповідність до цього стану. Даний підхід широко використовується в сучасних cloud-native системах і забезпечує відтворюваність середовища, однак не вирішує задачі вибору оптимальної конфігурації [34, 36, 37].

3) *Процесно-орієнтований (DevOps) підхід*. В цьому підході передбачається інтеграція процесів розробки та експлуатації, автоматизація CI/CD, безперервне розгортання та моніторинг. У межах цього підходу значна увага приділяється швидкості внесення змін і зворотному зв’язку, однак прийняття рішень щодо конфігурацій часто залишається або ручним, або базується на простих правилах [23, 24, 34, 36].

4. *Політико-орієнтований (policy-driven) підхід*. Тут передбачається управління системою на основі набору політик і правил (наприклад, SLA, обмежень ресурсів). Такий підхід дозволяє частково автоматизувати прийняття рішень, проте він є обмеженим через жорсткість правил і складність їх адаптації до нових умов [72, 73].

5. *Підхід, заснований на даних (Data-driven approach)*. Базується на використанні метрик, журналів та інших експлуатаційних даних для прийняття рішень щодо управління системою. Він створює передумови для

переходу до інтелектуальних методів, однак сам по собі не визначає механізмів формування рішень [59–61, 70, 88, 89].

У таблиці 1.2 надано узагальнення підходів до організації супроводу МСА. Аналіз дозволяє зробити висновок, що жоден із розглянутих підходів не забезпечує повноцінного вирішення задачі адаптивного управління конфігураціями в умовах динамічного середовища МСА [70, 72, 73].

Таблиця 1.2 – Класифікація підходів до організації супроводу МСА

Підхід	**Основна ідея**	**Переваги**	**Обмеження**
Централізований	Єдиний центр управління	Контрольованість	Низька масштабованість
Декларативний	Опис бажаного стану	Відтворюваність	Відсутність адаптивності
DevOps	Безперервна інтеграція та розгортання	Швидкість змін	Обмежена інтелектуалізація
Policy-driven	Управління через правила	Автоматизація	Жорсткість
Дані-орієнтований	Використання метрик	Обґрунтованість рішень	Відсутність механізму прийняття рішень

Зокрема:

- централізовані та декларативні підходи орієнтовані на статичні конфігурації;
- policy-driven підходи обмежені набором наперед визначених правил;
- DevOps підхід фокусується на процесах, а не на інтелектуальному прийнятті рішень;
- дані-орієнтовані підходи не забезпечують формалізованого механізму використання накопиченого досвіду.

Таким чином, виникає необхідність переходу до підходів, які поєднують: використання даних і метрик; врахування контексту

функціонування системи; можливість адаптації у runtime; використання накопиченого досвіду [10–13, 70].

Саме ці вимоги формують передумови для застосування інтелектуальних методів управління, що покладені в основу методологічних положень, представлених у Розділі 2 дисертаційної роботи.

На рисунку 1.6 представлено узагальнену еволюцію підходів до управління та супроводу програмних систем з мікросервісною архітектурою. Схема відображає поступовий перехід від централізованих і статичних підходів до більш автоматизованих та дані-орієнтованих рішень, що використовуються у сучасних cloud-native середовищах.

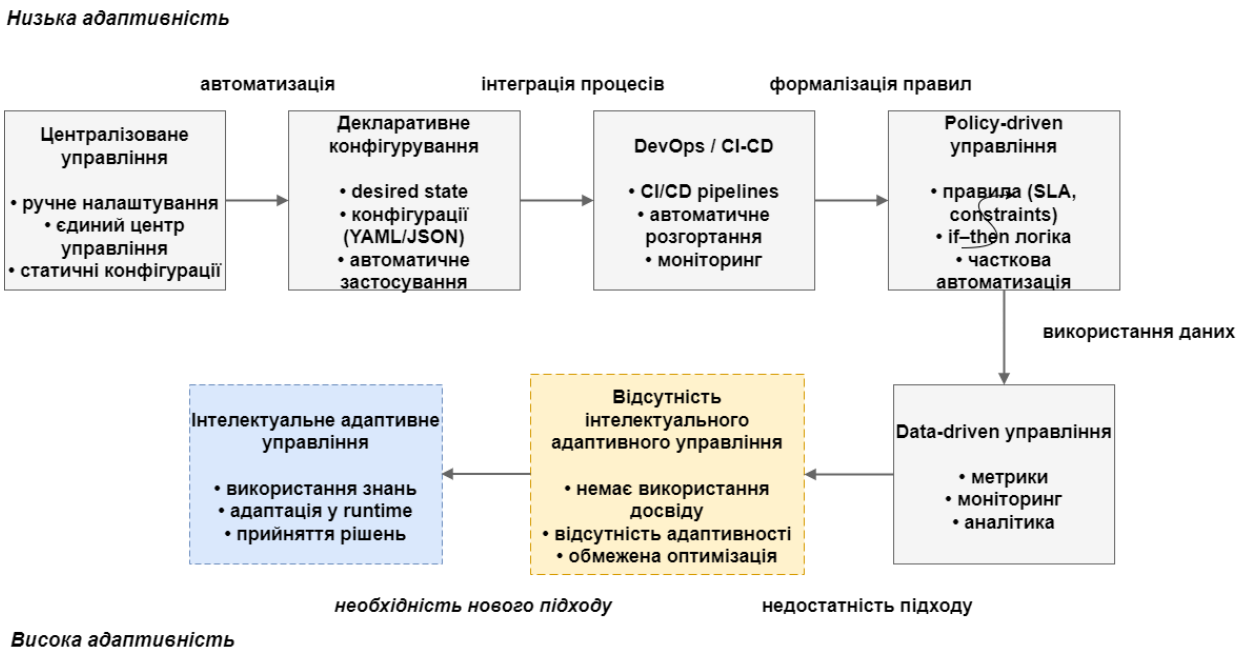


Рисунок 1.6 – Еволюція підходів до управління та супроводу ПС з МСА

Початковий етап представлений централізованим підходом, у межах якого управління конфігураціями та процесами супроводу здійснюється вручну або через єдиний центр управління. Такий підхід забезпечує контрольованість системи, однак не відповідає вимогам масштабованості та динамічності, характерним для мікросервісної архітектури [26–28, 72, 73].

Подальший розвиток пов’язаний із переходом до декларативного підходу, який базується на описі бажаного стану системи. Це дозволяє автоматизувати процеси розгортання та конфігурації, проте не забезпечує вибору оптимальних параметрів функціонування системи в умовах змінного середовища [34, 36, 37].

Наступним етапом є впровадження DevOps-підходу, який інтегрує процеси розробки та експлуатації, забезпечує безперервну інтеграцію та розгортання, а також скорочує час внесення змін. Разом із тим, у межах цього підходу прийняття рішень щодо конфігурацій здебільшого не автоматизується на інтелектуальному рівні [23, 24, 34, 36].

Подальша еволюція приводить до використання policy-driven підходів, у яких управління системою здійснюється на основі заданих правил і політик. Це дозволяє частково автоматизувати прийняття рішень, однак такі рішення є жорстко обмеженими наперед визначеними умовами та не враховують повною мірою динаміку середовища виконання [72, 73].

Сучасний етап розвитку характеризується переходом до data-driven підходів, що базуються на використанні метрик, журналів і даних моніторингу для аналізу стану системи. Незважаючи на це, такі підходи не забезпечують повноцінного механізму формування рішень, а лише створюють інформаційну основу для їх прийняття [59–61, 70].

Як показано на рисунку, між існуючими підходами та вимогами до ефективного управління мікросервісними системами існує концептуальний розрив, який полягає у відсутності інтелектуального адаптивного механізму прийняття рішень. Зокрема, сучасні підходи не забезпечують:

- врахування накопиченого досвіду функціонування системи [10–13, 69];
- адаптацію конфігурацій у режимі реального часу [70, 75, 76];
- багатокритеріальну оптимізацію параметрів функціонування [43, 46];
- інтеграцію знань про поведінку системи в процес прийняття рішень [29].

Подальший розвиток підходів до управління та супроводу програмних систем з МСА пов’язаний із переходом до використання інтелектуальних

методів, які забезпечують прийняття рішень на основі аналізу даних, знань та попереднього досвіду функціонування системи [70, 75, 76]. Такі підходи орієнтовані на розширення функціональних можливостей систем управління за рахунок включення механізмів аналізу, прогнозування та адаптації. У загальному випадку вони передбачають використання даних моніторингу, журналів подій та історії функціонування системи для формування обґрунтованих управлінських рішень [59–61, 70].

Серед основних напрямів інтелектуалізації управління мікросервісними системами доцільно виділити наступні:

1. *Rule-based підходи*. Базуються на використанні набору правил типу «якщо–то», що визначають поведінку системи у певних ситуаціях. Вони є логічним розвитком policy-driven підходів і дозволяють автоматизувати окремі рішення. Разом із тим, їх ефективність обмежується складністю формалізації правил та неможливістю врахування нетипових ситуацій [26–28].

2. *Підходи на основі машинного навчання*. Передбачають використання статистичних моделей для виявлення закономірностей у даних та прогнозування поведінки системи. Такі підходи здатні враховувати складні залежності між параметрами, однак потребують значних обсягів навчальних даних і не завжди забезпечують інтерпретованість результатів [70, 75, 76].

3. *Підходи на основі підкріплювального навчання*. Спрямовані на формування стратегій управління шляхом взаємодії з середовищем і отримання зворотного зв’язку. Вони є перспективними для задач оптимізації, проте їх застосування у практичних системах ускладняється високою обчислювальною складністю та тривалістю навчання [70, 75, 76].

4. *Підходи на основі аналізу прецедентів (Case-Based Reasoning, CBR)*. Передбачають використання накопиченого досвіду у вигляді бази прецедентів для прийняття рішень у нових ситуаціях. Основна ідея полягає у пошуку найбільш подібних раніше вирішених задач та адаптації відповідних рішень до поточного контексту. Такий підхід є доцільним для слабко формалізованих

задач, до яких саме і належить управління конфігураціями МСА [10–13, 69, 70]. Узагальнення характеристик зазначених підходів наведено у таблиці 1.3.

Таблиця 1.3 – Порівняльна характеристика інтелектуальних підходів до управління МСА

Підхід	**Основна ідея**	**Переваги**	**Обмеження**
Rule-based	Правила «якщо–то»	Простота реалізації	Жорсткість, слабка адаптивність
Machine Learning	Виявлення закономірностей	Висока точність	Потреба у даних, слабка інтерпретованість
Reinforcement Learning	Навчання через взаємодію	Оптимізація стратегій	Висока складність
CBR	Використання досвіду	Гнучкість, адаптивність	Залежність від бази прецедентів

Як видно з таблиці, інтелектуальні підходи відкривають можливості для підвищення адаптивності та ефективності управління мікросервісними системами, однак кожен із них має свої обмеження. Зокрема, rule-based підходи не забезпечують гнучкості, методи машинного навчання потребують значних обсягів даних і складні для інтерпретації, а підходи підкріплювального навчання є обчислювально затратними [26–28, 70, 75, 76].

У цьому контексті підходи на основі аналізу прецедентів (CBR) представляють особливий інтерес, оскільки вони поєднують можливість використання накопиченого досвіду з відносною простотою інтерпретації результатів. Крім того, CBR дозволяє ефективно працювати в умовах неповної формалізації задачі, що є характерним для управління конфігураціями мікросервісних систем [10–13, 69].

Водночас слід зазначити, що розглянуті підходи у більшості існуючих досліджень застосовуються фрагментарно і не інтегруються у єдиний модельно-технологічний інструментарій, орієнтований на повний життєвий

цикл конфігурацій мікросервісних систем [70, 75]. Це обмежує їх практичну застосовність у реальних умовах експлуатації.

Таким чином, проведений огляд інтелектуальних підходів дозволяє зробити висновок про доцільність використання методів аналізу прецедентів як основи для побудови адаптивного механізму управління конфігураціями МСА. Разом із тим, виникає необхідність розробки цілісного модельно-технологічного інструментарію, який забезпечує інтеграцію такого підходу в процеси розгортання та супроводу систем.

1.4 Постановка задачі розробки та дослідження модельно-технологічного інструментарію для адаптивного управління конфігураціями на етапах розгортання та супроводу ПС з МСА

Аналіз сучасних підходів до розробки та супроводу програмних систем з мікросервісною архітектурою, виконаний у підрозділах 1.1–1.3, показав, що процеси управління конфігураціями є одними з ключових факторів, що визначають ефективність функціонування таких систем. При цьому складність зазначених процесів зумовлена специфічними особливостями МСА, серед яких розподіленість, динамічність середовища виконання, значна кількість взаємодіючих компонентів та наявність багаточисельних конфігураційних параметрів [37, 39, 40, 63].

У загальному випадку задача управління конфігураціями мікросервісних систем полягає у виборі та підтримці такого набору параметрів функціонування окремих сервісів і їх взаємодії, який забезпечує досягнення заданих показників якості системи [43, 46–48, 91].

Існуючі підходи до управління конфігураціями, розглянуті у підрозділі 1.3, частково вирішують зазначені проблеми, однак не забезпечують комплексного підходу до адаптивного управління [26–28, 72, 73]. Таким чином, задача управління конфігураціями мікросервісних систем може бути сформульована як задача вибору оптимальних конфігурацій у

багатовимірному просторі параметрів за умов невизначеності та динамічності середовища. Для її розв'язання необхідно враховувати як поточний стан системи, так і результати її попереднього функціонування [10-13, 69].

Враховуючи наведене, виникає необхідність переходу до підходів, які дозволяють: інтегрувати дані моніторингу та історію функціонування системи; враховувати багатокритеріальні показники якості; забезпечувати адаптацію конфігурацій у режимі реального часу; використовувати накопичений досвід для прийняття рішень.

Зазначені вимоги узгоджуються з напрямами дослідження, визначеними у роботі, зокрема з необхідністю розробки алгоритмічних моделей, методів аналізу прецедентів та інформаційного базису для підтримки процесів прийняття рішень .

Для узагальнення взаємозв'язку між факторами, що впливають на процеси розробки та супроводу програмних систем, їх наслідками та підходами до вирішення проблем доцільно використати концептуальну схему дослідження, що представлена на рисунку 1.7.

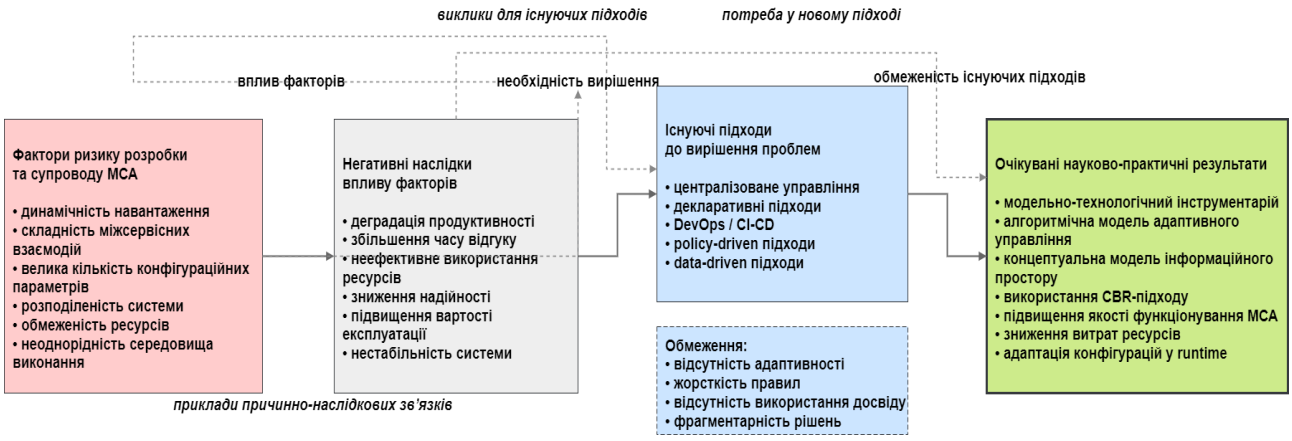


Рисунок 1.7 – Концептуальна схема дослідження

У загальному вигляді процес адаптивного управління конфігураціями МСА може бути представлений як цілеспрямована зміна параметрів функціонування системи з метою досягнення або підтримки заданих показників якості в умовах змінного середовища. При цьому управління

здійснюється на основі аналізу поточного стану системи, а також результатів її попереднього функціонування.

У рамках даного дослідження доцільно виділити п’ять складових задачі адаптивного управління.

1. *Об’єкт управління*. Об’єктом управління є програмна система з мікросервісною архітектурою, що складається з множини взаємодіючих сервісів, кожен з яких має власні конфігураційні параметри. До таких параметрів належать, зокрема, ресурси (CPU, пам’ять), параметри масштабування, мережеві налаштування, політики балансування навантаження тощо [34, 36, 37].

2. *Керуючі впливи*. Керуючими впливами виступають зміни конфігураційних параметрів мікросервісів, які можуть здійснюватися як на рівні окремих сервісів, так і на рівні їх взаємодії. Такі впливи повинні забезпечувати адаптацію системи до змін зовнішніх та внутрішніх умов функціонування [70, 75].

3. *Вхідні дані задачі* до яких входять: метрики функціонування системи (навантаження, час відгуку, використання ресурсів); дані моніторингу та журналів подій; інформація про поточну конфігурацію системи; історичні дані про попередні стани системи та відповідні конфігураційні рішення, які є принципово важливими, оскільки дозволяють враховувати накопичений досвід функціонування системи при прийнятті рішень [10–13, 69].

4. *Вихідні результати*. Результатом розв’язання задачі є вибір або формування нової конфігурації системи, яка забезпечує покращення або підтримку заданих показників якості її функціонування [43, 46–48].

5. *Критерії ефективності*. Оцінювання ефективності прийнятих рішень здійснюється на основі сукупності критеріїв, що характеризують якість функціонування системи. До них можуть належати: продуктивність; затримки обробки запитів; рівень використання ресурсів; вартість експлуатації; надійність та доступність системи [43, 46–48, 76].

З урахуванням наведеного, задача адаптивного управління конфігураціями МСА може бути інтерпретована як задача багатокритеріального вибору у багатовимірному просторі параметрів за умов невизначеності та динамічності середовища [43, 46–48, 76].

При цьому ключовою особливістю задачі є необхідність поєднання двох аспектів аналіз поточного стану системи (оперативний рівень) та використання накопиченого досвіду (стратегічний рівень). Саме така постановка задачі створює передумови для застосування інтелектуальних методів, здатних забезпечити прийняття рішень на основі даних і знань, а також їх адаптацію до нових умов функціонування.

Для наочного відображення взаємозв'язків між складовими задачі адаптивного управління доцільно використати узагальнену концептуальну модель, що представлена на рисунку 1.8.

Рисунок 1.8 – Концептуальна модель задачі адаптивного управління конфігураціями МСА

Центральним елементом моделі є мікросервісна система, що розглядається як об'єкт управління та включає сукупність взаємодіючих сервісів з визначеними конфігураційними параметрами. Функціонування системи супроводжується безперервним збором даних про її стан, які формуються у вигляді метрик продуктивності, показників використання

ресурсів, журналів подій та інших характеристик, що відображають поточний режим роботи системи [59–61].

Важливою складовою є також накопичені історичні дані, що містять інформацію про попередні конфігурації та результати їх застосування. Використання таких даних дозволяє враховувати досвід функціонування системи при прийнятті нових рішень та підвищує обґрунтованість управлінських впливів .

Отримані вхідні дані надходять до блоку прийняття рішень, у якому здійснюється аналіз стану системи, оцінювання якості її функціонування за визначеними критеріями та формування нових конфігураційних рішень. При цьому процес прийняття рішень повинен враховувати багатокритеріальний характер задачі та забезпечувати узгодження між різними показниками якості, такими як продуктивність, затримки, використання ресурсів і надійність системи [43, 46–48].

Результатом функціонування блоку прийняття рішень є нова конфігурація мікросервісної системи, яка передається до середовища виконання та застосовується з використанням відповідних інструментальних засобів розгортання та управління. Після застосування конфігурації відбувається зміна стану системи, що знову фіксується у вигляді метрик та інших даних моніторингу.

Таким чином, формується замкнений контур управління із зворотним зв’язком, у межах якого результати попередніх рішень використовуються для подальшого вдосконалення процесу управління. Наявність такого зворотного зв’язку є необхідною умовою забезпечення адаптивності системи та її здатності реагувати на зміни середовища функціонування.

Отже, наведена концептуальна модель відображає загальні принципи організації процесу адаптивного управління конфігураціями мікросервісних систем та визначає ключові елементи, які повинні бути враховані при розробці відповідного модельно-технологічного інструментарію.

Сформульована концептуальна постановка задачі адаптивного управління конфігураціями мікросервісних систем визначає необхідність створення спеціалізованого модельно-технологічного інструментарію, який забезпечує реалізацію відповідних процесів прийняття рішень. Такий інструментарій повинен інтегрувати існуючі засоби управління інфраструктурою з інтелектуальними механізмами обробки даних та формування конфігураційних рішень, .

Враховуючи специфіку задачі, вимоги до інструментарію доцільно поділити на функціональні та нефункціональні.

До основних функціональних вимог належать:

1) *Збір та обробка даних про стан системи*. Інструментарій повинен забезпечувати інтеграцію з системами моніторингу та логування з метою отримання актуальної інформації про функціонування мікросервісів (метрики, події, журнали) [59–61].

2) *Зберігання та використання історичних даних*. Повинна бути реалізована можливість накопичення інформації про попередні конфігурації та результати їх застосування, що є основою для подальшого аналізу та використання досвіду [10–13, 69].

3) *Підтримка механізмів прийняття рішень*. Інструментарій має забезпечувати формування рекомендацій або автоматичний вибір конфігурацій на основі аналізу поточного стану системи та історичних даних.

4) *Генерація та застосування конфігурацій*. Повинна бути реалізована можливість автоматичного формування нових конфігурацій та їх інтеграції в існуючу інфраструктуру розгортання (оркестратори, CI/CD-процеси) [36, 37].

5) *Оцінювання ефективності прийнятих рішень*. Інструментарій має забезпечувати аналіз результатів застосування конфігурацій з використанням визначених метрик якості [43, 46–48].

До ключових нефункціональних вимог можна віднести:

1) *Адаптивність*. Забезпечення можливості зміни конфігурацій у відповідь на зміну умов функціонування системи в режимі реального часу.

2) *Масштабованість*. Забезпечення здатності обробляти великі обсяги даних та працювати з великою кількістю мікросервісів.

3) *Інтегрованість*. Інструментарій має бути сумісним із сучасними платформами розгортання та управління МСА (оркестратори, CI/CD).

4) *Ефективність обчислень*. Алгоритми прийняття рішень повинні забезпечувати прийнятний час обробки даних та формування рішень.

5) *Інтерпретованість результатів*. Можливість пояснення прийнятих рішень, що є критичним для практичного використання інструментарію.

З урахуванням наведених вимог, модельно-технологічний інструментарій повинен бути побудований як комплексна система, що поєднує: моделі представлення стану системи; механізми зберігання та обробки даних; алгоритми прийняття рішень; засоби інтеграції з інфраструктурою виконання.

Для узагальнення вимог до інструментарію доцільно представити їх у вигляді таблиці 1.4.

Таблиця 1.4 – Основні вимоги до модельно-технологічного інструментарію адаптивного управління МСА

Тип вимог	**Група вимог**	**Зміст**
Функціональні	Дані	Збір, обробка та зберігання метрик і журналів
Функціональні	Прийняття рішень	Формування конфігурацій на основі аналізу даних
Функціональні	Інтеграція	Взаємодія з оркестраторами та CI/CD
Функціональні	Оцінка	Аналіз ефективності конфігурацій
Нефункціональні	Адаптивність	Динамічна зміна конфігурацій
Нефункціональні	Масштабованість	Робота з великою кількістю сервісів
Нефункціональні	Інтерпретованість	Пояснення прийнятих рішень

Як показано у таблиці 1.4, ефективне вирішення задачі адаптивного управління конфігураціями потребує комплексного підходу, що поєднує

функціональні можливості обробки даних та прийняття рішень із вимогами до адаптивності та масштабованості системи.

Постановка задач дослідження

Проведений аналіз проблеми управління конфігураціями мікросервісних систем, її концептуальна постановка, а також сформульовані вимоги до модельно-технологічного інструментарію дозволяють перейти до визначення основних задач дослідження, спрямованих на розв’язання поставленої наукової проблеми.

Метою дослідження є підвищення якості процесів розгортання та супроводу розподілених програмних систем з мікросервісною архітектурою шляхом розробки модельно-технологічного інструментарію для адаптивного управління конфігураціями їх компонентів .

Для досягнення поставленої мети у дисертаційній роботі необхідно розв’язати такі основні задачі:

1) Провести аналіз особливостей мікросервісної архітектури розподілених програмних систем з метою виявлення факторів, що впливають на процеси управління конфігураціями та показники якості їх функціонування.

2) Виконати огляд існуючих методів і інструментальних засобів управління конфігураціями та визначити їх обмеження щодо забезпечення адаптивності в умовах динамічного середовища.

3) Дослідити можливості застосування інтелектуальних підходів, зокрема методу аналізу прецедентів (Case-Based Reasoning), для задач адаптивного управління конфігураціями мікросервісних систем.

4) Розробити концептуальну модель багатовимірного інформаційного простору, що забезпечує представлення стану системи та підтримку процесів пошуку і вибору конфігураційних рішень .

5) Розробити алгоритмічну модель адаптивного управління конфігураціями на основі використання інтелектуальних методів, що дозволяє враховувати поточний стан системи та накопичений досвід її функціонування.

6) Проаналізувати та обґрунтувати вибір методів пошуку релевантних рішень, зокрема алгоритмів вилучення прецедентів (Nearest Neighbor, Feature-Based, Cluster-Based, Indexing та ін.), з метою підвищення ефективності процесу прийняття рішень.

7) Розробити архітектуру та здійснити програмну реалізацію інструментального засобу для адаптивного управління конфігураціями мікросервісних систем з використанням інтелектуальних підходів.

8) Побудувати тестове середовище (полігон) мікросервісної системи для проведення експериментальних досліджень ефективності запропонованого підходу.

9) Провести експериментальне дослідження та оцінювання ефективності розробленого інструментарію, зокрема шляхом порівняння з існуючими підходами та аналізу отриманих результатів.

1.5. Висновки по Розділу 1

У першому розділі дисертаційної роботи виконано системний аналіз сучасного стану проблеми розробки та супроводу програмних систем на основі мікросервісної архітектури, що дозволило сформувати цілісне уявлення про особливості функціонування таких систем, їх переваги, обмеження та ключові виклики, пов’язані з управлінням їх конфігураціями.

Проведений аналіз показав, що мікросервісна архітектура на сучасному етапі розвитку інформаційних технологій є домінуючим підходом до побудови складних розподілених програмних систем завдяки своїй здатності забезпечувати масштабованість, гнучкість, модульність і відмовостійкість. Разом із тим, перехід від монолітних до мікросервісних систем супроводжується суттєвим зростанням складності управління, що проявляється у збільшенні кількості конфігураційних параметрів, підвищенні рівня взаємозалежності компонентів та необхідності врахування динамічних змін середовища функціонування.

Встановлено, що однією з центральних проблем мікросервісних систем є управління конфігураціями, яке безпосередньо впливає на такі критичні характеристики системи, як продуктивність, ефективність використання ресурсів, надійність і доступність сервісів. При цьому традиційні підходи до конфігурування, що базуються на статичних правилах або ручному налаштуванні, не здатні забезпечити необхідний рівень адаптивності в умовах динамічних навантажень та змін параметрів середовища. Це призводить до виникнення ефекту накопичення помилок та до зниження якості функціонування системи в цілому.

Аналіз чинників, що визначають необхідність адаптивного управління конфігураціями, показав, що ця задача має багатокритеріальний характер і потребує одночасного врахування великої кількості параметрів, включаючи навантаження, використання ресурсів, показники якості обслуговування та експлуатаційні витрати. Додатково встановлено, що у процесі функціонування систем накопичується значний обсяг історичних даних, який може бути використаний для підвищення ефективності прийняття рішень, однак у більшості існуючих підходів цей потенціал використовується недостатньо.

У рамках аналізу життєвого циклу програмних систем на основі мікросервісної архітектури встановлено, що класичні моделі розробки програмного забезпечення не відповідають сучасним вимогам, оскільки не забезпечують необхідного рівня гнучкості, масштабованості та інтеграції процесів розробки й експлуатації. Показано, що концепція DevOps є ключовим фактором трансформації життєвого циклу таких систем, оскільки забезпечує безперервність процесів розробки, тестування, розгортання та моніторингу. Водночас, навіть у рамках DevOps залишається невирішеною задача інтелектуального адаптивного управління конфігураціями, що вимагає додаткових досліджень.

Окрему увагу приділено аналізу сучасних інструментальних засобів, зокрема технологій контейнеризації та оркестрації, а також систем моніторингу, які забезпечують автоматизацію окремих аспектів управління

мікросервісами. Встановлено, що такі засоби, як Docker, Kubernetes, Prometheus та інші, створюють необхідну інфраструктурну основу для управління мікросервісними системами, проте не вирішують у повному обсязі задачі адаптивного прийняття рішень щодо конфігурації на основі поточного стану системи та накопиченого досвіду.

У результаті проведеного аналізу обґрунтовано доцільність використання інтелектуальних підходів до управління конфігураціями, зокрема методів аналізу прецедентів (Case-Based Reasoning). Показано, що такі методи дозволяють враховувати попередній досвід функціонування системи, виконувати пошук аналогічних ситуацій та адаптувати раніше знайдені рішення до нових умов, що є особливо важливим для динамічних середовищ мікросервісних систем.

Таким чином, у першому розділі сформульовано наукову проблему, яка полягає у необхідності розробки інтелектуальних моделей і інструментальних засобів для адаптивного управління конфігураціями програмних мікросервісів з урахуванням багатовимірного характеру їх функціонування. Визначено, що вирішення цієї проблеми потребує поєднання методів системного аналізу, математичного моделювання, інтелектуальних методів прийняття рішень та сучасних технологій програмної інженерії. Також була сформульована мета та задачі дисертаційного дослідження, які спрямовані на створення модельно-технологічного інструментарію для автоматизованого та адаптивного управління конфігураціями мікросервісних систем на різних етапах їх життєвого циклу.

РОЗДІЛ 2. МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ ІНТЕЛЕКТУАЛЬНОГО МОДЕЛЬНО-ТЕХНОЛОГІЧНОГО ІНСТРУМЕНТАРІЮ ДЛЯ АДАПТИВНОГО УПРАВЛІННЯ ПРОГРАМНИМИ МІКРОСЕРВІСАМИ

2.1 Доменне моделювання як концептуальна основа для аналізу принципів побудови інфраструктурних особливостей функціонування ПС з МСА

У контексті дослідження програмних систем з мікросервісною архітектурою (МСА) доменне моделювання виступає ключовим інструментом формалізації знань про предметну область, що дозволяє систематизувати сутності, процеси та взаємозв’язки між компонентами системи [29, 33, 69]. На відміну від суто архітектурного або технологічного опису, доменне моделювання забезпечує абстракцію більш високого рівня, що дозволяє узагальнити підходи до управління конфігураціями незалежно від конкретних реалізацій [16, 18, 19, 66].

Особливу актуальність доменне моделювання набуває в умовах динамічних розподілених середовищ, де поведінка системи визначається не лише статично заданими параметрами, а й змінними характеристиками середовища виконання, такими як навантаження, доступність ресурсів, затримки мережі тощо [34, 36, 37, 63, 86]. У таких умовах формування адекватної доменної моделі дозволяє: ідентифікувати ключові об’єкти управління; визначити релевантні параметри конфігурації; формалізувати залежності між станом системи та її конфігурацією; створити основу для застосування інтелектуальних методів прийняття рішень.

Таким чином, доменне моделювання виступає концептуальним підґрунтям для подальшої побудови моделей адаптивного управління конфігураціями, зокрема на основі методів аналізу прецедентів (CBR) [10–13].

Аналіз сучасних підходів до побудови мікросервісних систем, а також результатів попередніх досліджень і розроблених програмних прототипів

дозволяє виділити базові сутності домену адаптивного управління конфігураціями [14, 15, 18, 19]. До ключових сутностей належать:

Мікросервіс (Microservice) – автономний компонент системи, що реалізує окрему бізнес-функцію та має власні конфігураційні параметри.

Конфігурація (Configuration) – множина параметрів, що визначають поведінку мікросервісу (обмеження ресурсів, параметри масштабування).

Середовище виконання (Execution Environment) – сукупність умов функціонування системи, включаючи апаратні ресурси, мережеві характеристики та поточне навантаження.

Метрики (Metrics) – кількісні показники, що характеризують стан системи (час відгуку, використання CPU, пам’яті, рівень доступності тощо) .

Прецедент (Case) – структурований запис, що містить опис стану системи, відповідної конфігурації та досягнутого результату.

Система адаптивного управління конфігураціями (АУКМ) – інтелектуальний компонент, що здійснює аналіз поточного стану системи та визначає оптимальні конфігурації [1, 3, 70, 75, 76].

Таблиця 2.1 узагальнює основні сутності домену адаптивного управління конфігураціями мікросервісних систем. Особливу роль відіграє прецедент як інтегруюча сутність, що поєднує опис стану системи, параметри середовища виконання та результати застосування конфігурації.

Таблиця 2.1 – Основні сутності домену АУКМ

№	Сутність	Опис	Основні атрибути	Роль у системі
1	Мікросервіс (Microservice)	Автономний компонент, що реалізує окрему бізнес-функцію	ID, назва, API, залежності	Виконує функціональні операції системи
2	Конфігурація (Configuration)	Набір параметрів, що визначають поведінку сервісу	CPU limit, RAM, replicas, timeout	Визначає режим роботи сервісу
3	Середовище виконання (Execution Environment)	Умови функціонування системи (ресурси, мережа, навантаження)	інфраструктура, latency, load	Впливає на ефективність конфігурації

Закінчення Таблиці 2.1

№	Сутність	Опис	Основні атрибути	Роль у системі
4	Метрики (Metrics)	Кількісні показники стану системи	response time, CPU, RAM, throughput	Використовуються для оцінки стану
5	Прецедент (Case)	Структурований запис "стан–конфігурація–результат"	S, E, C, R, M	Основа для прийняття рішень (CBR)
6	Система АУКМ	Інтелектуальна система управління конфігураціями	модель, база знань, алгоритм	Забезпечує адаптивне управління

Сутності домену не існують ізольовано, а утворюють складну систему взаємозв’язків, що визначає логіку функціонування мікросервісної системи.

Ключові залежності можна сформулювати наступним чином:

- конфігурація мікросервісу безпосередньо впливає на значення метрик [37, 39, 40, 59];
- метрики відображають стан системи в конкретному середовищі виконання [43, 46-48];
- сукупність метрик та параметрів середовища формує опис ситуації (стану системи) [29, 33];
- стан системи використовується як вхідні дані для пошуку релевантного прецеденту [10–13];
- вибраний прецедент визначає рекомендовану конфігурацію [13, 69].

Таким чином, формується замкнений контур управління

стан → аналіз → рішення → зміна конфігурації → новий стан.

На рисунку 2.1 наведено узагальнену доменну модель процесу адаптивного управління конфігураціями МСА. Центральним елементом моделі є система адаптивного управління конфігураціями мікросервісів (АУКМ), яка виконує функції аналізу стану системи, формування рішень та внесення змін до конфігурації.

З лівого боку схеми розташовані базові сутності, що характеризують об’єкт управління. Мікросервіс представлено як автономний функціональний

компонент системи, який має власну конфігурацію та генерує набір метрик під час функціонування. Конфігурація визначає параметри роботи сервісу, зокрема обмеження ресурсів, кількість реплік, часові параметри та інші налаштування. Метрики, у свою чергу, відображають поточний стан системи та використовуються для оцінювання ефективності її функціонування.

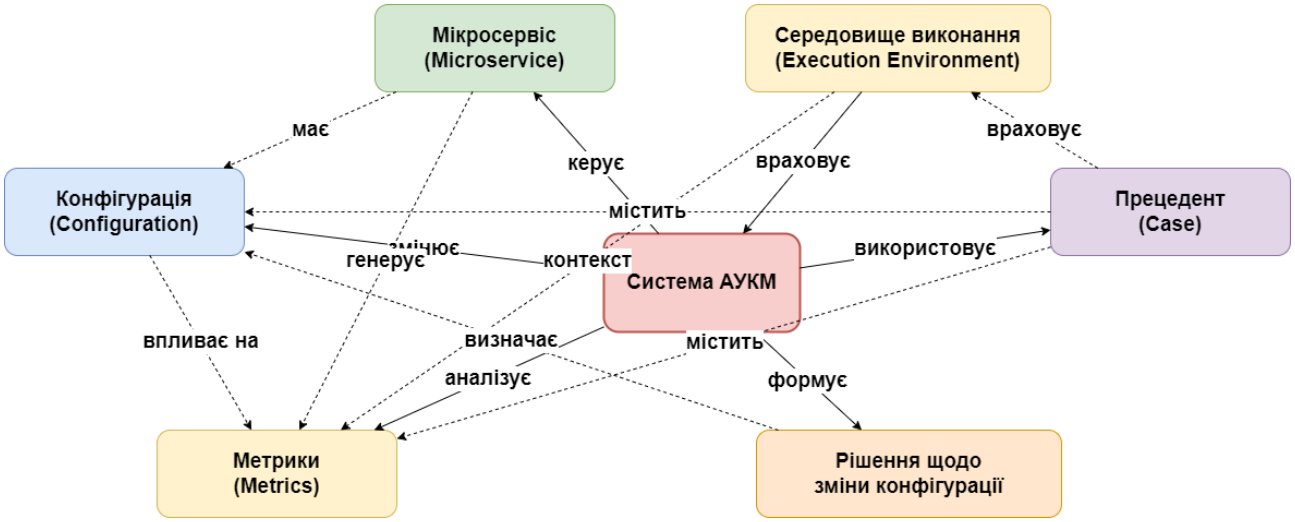


Рисунок 2.1 –Доменна модель процесу адаптивного управління МСА

Праворуч від центрального блоку подано сутності, що забезпечують інтелектуальне прийняття рішень. Середовище виконання визначає контекст функціонування системи, оскільки одна й та сама конфігурація може демонструвати різну ефективність залежно від умов навантаження, характеристик інфраструктури та мережевих параметрів [34, 36, 37]. Прецедент виступає структурованою формою подання накопиченого досвіду, що поєднує дані про метрики, конфігурацію та умови виконання. Рішення щодо зміни конфігурації формується на основі аналізу поточного стану системи та релевантних прецедентів [10–13, 69].

Пунктирні зв’язки на схемі відображають концептуальні залежності між сутностями домену. Зокрема, показано, що мікросервіс має конфігурацію, генерує метрики, а сама конфігурація безпосередньо впливає на значення цих метрик. Також підкреслено, що прецедент містить інформацію про

конфігурацію, метрики та середовище виконання, а сформоване системою рішення визначає подальшу зміну конфігурації сервісу.

Таким чином, рисунок 2.1 відображає узагальнену структуру предметної області адаптивного управління конфігураціями та демонструє замкнений контур управління, у якому поточний стан системи аналізується, використовується для формування рішення, після чого конфігурація змінюється, а система переходить у новий стан. Ця модель створює концептуальну основу для подальшого аналізу процесів, метрик та інтелектуальних методів, зокрема CBR-підходу [10–13, 29, 68].

Процеси домену: життєвий цикл конфігурації

У межах доменної моделі ключову роль відіграють процеси, що забезпечують адаптивне управління конфігураціями. Можна виділити типовий цикл [70, 72, 75]:

1) *Моніторинг* – збір метрик та параметрів середовища.
2) *Формування опису ситуації* – агрегування та нормалізація даних.
3) *Пошук прецедентів (Retrieve)* – визначення найбільш схожих випадків.
4) *Адаптація рішення (Reuse/Revise)* – коригування знайденої конфігурації.
5) *Застосування конфігурації* – внесення змін у систему.
6) *Оцінка результату* – аналіз нових значень метрик.
7) *Збереження прецеденту (Retain)* – поповнення бази знань.

Зазначений процес відповідає класичному циклу CBR та адаптований до специфіки мікросервісних систем. На рисунку 2.2 представлено життєвий цикл адаптивного управління конфігураціями в мікросервісній архітектурі. Схема відображає послідовність взаємопов’язаних процесів, які утворюють замкнений контур інтелектуального управління та відповідають адаптованому до задач конфігураційного управління циклу CBR [10–13].

Першим етапом є *моніторинг*, під час якого здійснюється збір метрик функціонування системи та параметрів середовища виконання [20, 21, 59]. Ці дані характеризують поточний стан мікросервісної системи й виступають основою для подальшого аналізу.

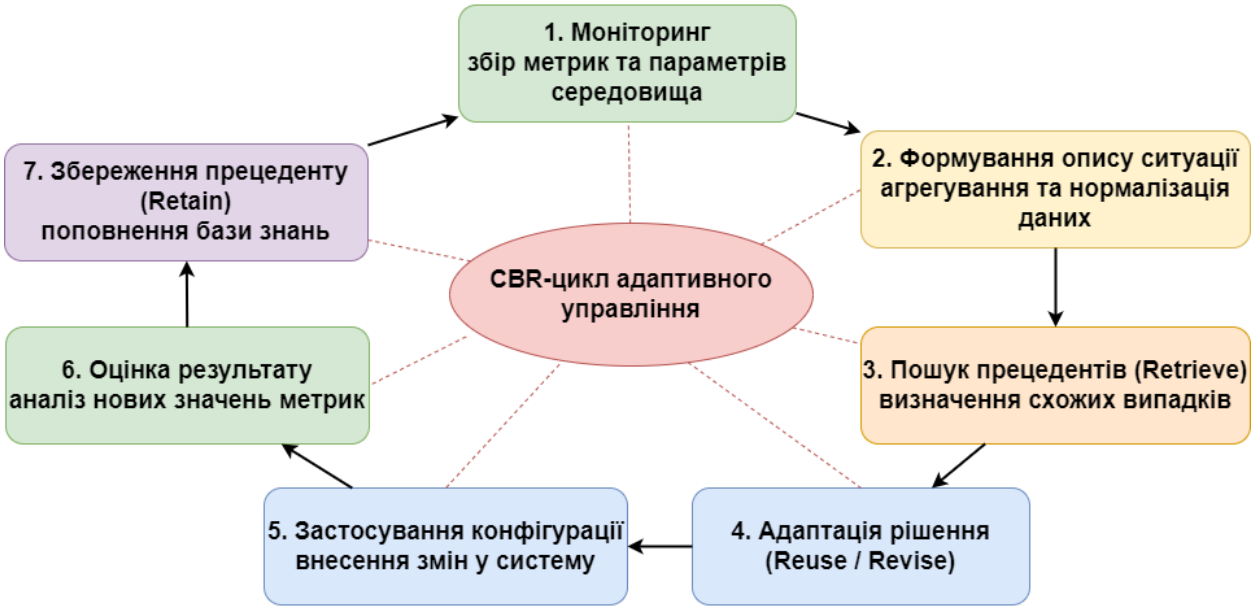


Рисунок 2.2 – Життєвий цикл адаптивного управління конфігураціями

На другому етапі виконується *формування опису ситуації*, що включає агрегування, нормалізацію та структурування отриманих даних [29, 33]. Саме на цьому кроці поточний стан системи набуває формалізованого вигляду, придатного для порівняння з наявними прецедентами.

Третій етап - *пошук прецедентів (Retrieve)* - полягає у визначенні найбільш схожих історичних випадків із бази знань. У межах CBR-підходу цей етап має ключове значення, оскільки від коректності вибору прецеденту залежить якість подальшого рішення [10–13, 69].

На четвертому етапі здійснюється *адаптація рішення (Reuse / Revise)*, тобто коригування знайденого раніше рішення відповідно до особливостей поточної ситуації. Це дозволяє не просто копіювати попередній досвід, а використовувати його як основу для формування нового, релевантного рішення.

П’ятим етапом є *застосування конфігурації*, тобто внесення змін до параметрів системи. Після цього система переходить у новий функціональний стан, який може бути оцінений за допомогою метрик [43, 46–48].

Шостий етап - *оцінка результату* - передбачає аналіз того, наскільки застосована конфігурація покращила або погіршила функціонування системи. Оцінюються продуктивність, стабільність, використання ресурсів та інші релевантні показники [43, 59–61].

На завершальному етапі виконується *збереження прецеденту (Retain)*, тобто поповнення бази знань новим досвідом. Це забезпечує навчання системи на власному досвіді та підвищення якості управління в майбутньому.

Таким чином, рисунок 2.2 демонструє, що адаптивне управління конфігураціями в МСА має циклічний характер, а його ефективність забезпечується безперервним переходом від спостереження за станом системи до накопичення нових знань про результати прийнятих рішень

Метрики є центральним елементом доменної моделі, оскільки саме вони забезпечують кількісну оцінку ефективності конфігурацій (див рисунок 2.1). Можна виділити такі основні групи метрик [43, 46–48, 59–61]:

- *ресурсні*: CPU, RAM, disk usage, network load;
- *продуктивнісні*: response time, throughput;
- *експлуатаційні:* availability, error rate;
- *економічні:* operational cost.

Метрики використовуються: як ознаки для опису прецедентів; як критерії оптимальності при виборі конфігурації; як основа для оцінки ефективності адаптації [43, 46–48].

В таблиці 2.2 надається узагальнення основних груп метрик, що використовуються у процесі адаптивного управління конфігураціями мікросервісних систем. Представлені метрики охоплюють різні аспекти функціонування системи, зокрема використання ресурсів, продуктивність, експлуатаційні характеристики та економічну ефективність [43, 46–48, 59–61].

Таблиця 2.2 – Основні метрики та їх роль у адаптивному управлінні

№	Група метрик	Метрика	Опис	Роль у системі адаптивного управління
1	Ресурсні	CPU usage (%)	Завантаження процесора	Визначає необхідність масштабування та балансування навантаження
2	Ресурсні	RAM usage (%)	Використання оперативної пам’яті	Контроль споживання пам’яті та запобігання перевантаженню
3	Ресурсні	Disk usage (%)	Використання дискових ресурсів	Оцінка стану зберігання даних та ризику переповнення
4	Ресурсні	Network load (Mbps)	Інтенсивність мережевого трафіку	Визначає необхідність оптимізації мережевих взаємодій
5	Продуктивнісні	Response time (ms)	Час відгуку сервісу	Ключовий показник якості обслуговування (QoS)
6	Продуктивнісні	Throughput (req/sec)	Кількість оброблених запитів за одиницю часу	Оцінка продуктивності системи
7	Продуктивнісні	Latency (ms)	Затримка обробки запитів	Визначає швидкодію системи
8	Експлуатаційні	Availability (%)	Рівень доступності сервісу	Оцінка надійності та безперервності роботи
9	Експлуатаційні	Error rate (%)	Частка помилкових запитів	Індикатор стабільності системи
10	Експлуатаційні	Failure rate	Частота відмов	Використовується для оцінки деградації системи
11	Економічні	Operational cost	Вартість використання ресурсів	Оцінка економічної ефективності конфігурації
12	Економічні	Resource cost per request	Вартість обробки одного запиту	Оптимізація витрат при масштабуванні
13	Композитні	SLA compliance (%)	Відповідність SLA	Інтегральна оцінка якості обслуговування
14	Композитні	System efficiency index	Комплексний показник ефективності	Використовується для порівняння конфігурацій

Ресурсні метрики відображають ступінь використання обчислювальних, пам’яті та мережевих ресурсів і дозволяють виявляти перевантаження або

неефективне використання інфраструктури [43, 46–48, 59–61]. Продуктивнісні метрики характеризують швидкодію системи та якість обробки запитів, що є критично важливим для забезпечення необхідного рівня сервісу. Експлуатаційні метрики дозволяють оцінити стабільність і надійність роботи системи, зокрема через аналіз доступності та частоти помилок [43, 46–48].

Окрему групу становлять економічні метрики, які забезпечують можливість оцінки вартості використання ресурсів і оптимізації витрат у процесі адаптації конфігурацій. Композитні метрики інтегрують декілька показників і дозволяють отримати узагальнену оцінку ефективності функціонування системи [43, 46–48, 76].

Таким чином, метрики виступають ключовим елементом доменної моделі адаптивного управління, оскільки вони використовуються як для формування опису поточного стану системи, так і для оцінки ефективності прийнятих рішень та накопичення прецедентів у базі знань [43, 46–48].

Узагальнюючи наведені положення, доменна модель адаптивного управління конфігураціями може бути представлена як багаторівнева структура, що включає [29, 33, 68, 69]:

1) рівень мікросервісів;
2) рівень конфігурацій;
3) рівень середовища виконання;
4) рівень метрик;
5) рівень знань (прецедентів);
6) рівень прийняття рішень (CBR-модуль).

Така модель забезпечує формалізацію процесів управління, інтеграцію даних моніторингу, можливість застосування інтелектуальних методів та масштабованість і розширюваність системи [20, 21]. Наступним етапом є впровадження DevOps-підходу, який інтегрує процеси розробки та експлуатації, забезпечує безперервну інтеграцію та розгортання, а також скорочує час внесення змін [23, 24, 34, 36]. Разом із тим, у межах цього підходу

прийняття рішень щодо конфігурацій здебільшого не автоматизується на інтелектуальному рівні [70, 72, 75].

Таким чином доменне моделювання дозволило сформувати цілісне уявлення про структуру та принципи функціонування систем адаптивного управління конфігураціями в мікросервісній архітектурі.

2.2 Порівняльний аналіз моделей та методів адаптивного управління в ПС з МСА

У сучасних дослідженнях адаптивного управління програмними системами з МСА відсутня єдина загальноприйнята класифікація методів, що обумовлено різноманітністю підходів, середовищ виконання та рівнів абстракції управління [41–45, 65, 66]. Водночас аналіз сучасних оглядових робіт дозволяє виділити узагальнені критерії класифікації [43, 76].

Зокрема, у роботах, присвячених аналізу та «reasoning» мікросервісних систем, підкреслюється необхідність розгляду системи на декількох рівнях: сервісному, рівні взаємодій та інфраструктурному, що безпосередньо впливає на вибір методів управління [16, 18, 19, 23].

Відповідно, методи адаптивного управління доцільно класифікувати за такими ознаками:

1) за джерелом знань:
   - rule-based (правила) [26–28];
   - data-driven (машинне навчання) [70, 75];
   - experience-driven (CBR) [10–13];
2) за характером адаптації:
   - реактивні (reactive) [72, 73];
   - проактивні (predictive) [70, 75, 76];
3) за рівнем автоматизації:
   - ручні / напівавтоматичні [26–28];
   - повністю автономні (self-adaptive systems) [70, 72, 73];

4) за часовими характеристиками:

- офлайн-оптимізація [75, 76];
- реального часу (runtime adaptation) [70, 72, 73].

Такий підхід узгоджується з концепціями самокерованих систем (self-adaptive systems), які базуються на циклі MAPE-K (*Monitor–Analyze–Plan–Execute–Knowledge*) і широко застосовуються для управління мікросервісами.

Класичні (статичні) підходи до управління конфігураціями

Класичні підходи до управління конфігураціями в МСА базуються на централізованих конфігураційних сервісах та декларативних підходах [34, 36–38]. До таких рішень належать Kubernetes ConfigMaps, Spring Cloud Config, Consul та AWS Config [34, 36, 37]. Їх ключовою особливістю є використання статичних конфігурацій, які змінюються вручну або за заздалегідь визначеними правилами [26–28]. Переваги класичних підходів до управління конфігураціями в МСА: простота реалізації; контрольованість; зрозумілість для DevOps-практик [23, 24, 34, 36]. Недоліки таких підходів: відсутність адаптивності; низька реакція на динамічні зміни середовища необхідність ручного втручання [26–28].

У сучасних умовах динамічного навантаження такі підходи демонструють обмежену ефективність, оскільки не враховують змінні характеристики системи та не використовують історичні дані [72, 73].

Оптимізаційні та евристичні методи адаптивного управління

Оптимізаційні підходи базуються на формалізації задачі управління у вигляді задачі оптимізації. У таких моделях використовуються: евристичні алгоритми; multi-objective optimization ; модель на основі обмежень. Ці підходи орієнтовані на мінімізацію або максимізацію певної цільової функції, яка може враховувати: продуктивність; використання ресурсів; вартість експлуатації [43, 46–48, 75, 76].

До переваг оптимізаційних методів адаптивного управління відносяться: формалізованість; можливість оптимізації кількох критеріїв; теоретична обґрунтованість. Недоліки повязані з складністю визначення цільових

функцій ; високими обчислювальними витратами; недостатньою гнучкістю у змінному середовищі [46-48, 75, 76].

Методи на основі машинного навчання

Сучасні дослідження активно розглядають застосування методів машинного навчання (ML) для адаптивного управління MCA. До них належать: supervised learning; reinforcement learning (RL); predictive analytics . ML широко використовується для: прогнозування навантаження [70, 75]; автоматизації масштабування та міграції сервісів [70, 75]; оптимізації розподілу ресурсів [34, 36, 37]; оптимізації розподілу ресурсів [43, 46–48]. Недоліки підходу на основі методів машинного навчання – це потреба в навчальних вибірках; потреба у великих обсягах даних ; складність інтеграції ; низька пояснюваність [70, 75, 76].

Методи самокерованих систем (Self-adaptive systems)

Окрему групу становлять підходи, що базуються на концепції самокерованих систем, які інтегрують: моніторинг; аналіз; прийняття рішень; виконання змін [70, 72, 73]. Такі системи часто реалізуються на основі циклу MAPE-K і використовуються для: автоматичного масштабування; балансування навантаження; управління ресурсами. Сучасні роботи демонструють, що застосування self-adaptive підходів дозволяє автоматизувати управління, підвищити стійкість, зменшити втручання людини . Разом з тим, такі системи потребують складної архітектури, ефективних механізмів аналізу стану, інтеграції з доменною моделлю системи [34, 36, 37].

Методи на основі аналізу прецедентів (Case-Based Reasoning)

Методи аналізу прецедентів займають окреме місце серед підходів адаптивного управління, оскільки вони базуються на використанні історичного досвіду системи. Основна ідея полягає в тому, що система зберігає попередні стани (cases), при виникненні нової ситуації виконується пошук аналогічного випадку, знайдене рішення адаптується до поточних умов [10–13].

Переваги CBR: адаптивність у реальному часі ; використання історичних даних ; висока пояснюваність ; відсутність необхідності повного навчання . CBR добре узгоджується з особливостями МСА, де система є динамічною, накопичується великий обсяг експлуатаційних даних, важлива швидка реакція на зміни [10–13, 69]. Для узагальнення проведеного аналізу використано порівняльну характеристику методів, представлену у таблиці 2.3.

Таблиця 2.3 – Порівняння методів адаптивного управління в ПС з МСА

Критерій	Статичні (rule-based)	Оптимізаційні / евристичні	Машинне навчання (ML/RL)	Self-adaptive (MAPE-K)	CBR (Case-Based Reasoning)
Рівень адаптивності	Низький	Середній	Високий	Високий	Високий
Робота в реальному часі	Обмежена	Частково	Обмежена (залежить від моделі)	Висока	Висока
Потреба у навчальних даних	Відсутня	Низька	Висока	Середня	Низька (накопичується поступово)
Використання історичного досвіду	Ні	Частково	Так	Частково	Так (ключовий елемент)
Складність реалізації	Низька	Середня–висока	Висока	Висока	Середня
Обчислювальні витрати	Низькі	Середні	Високі	Середні–високі	Середні
Пояснюваність рішень	Висока	Середня	Низька	Середня	Висока
Гнучкість до змін середовища	Низька	Середня	Висока	Висока	Висока
Масштабованість	Висока	Середня	Висока	Висока	Висока
Залежність від експертних знань	Висока	Середня	Низька	Середня	Низька
Стійкість до невизначеності	Низька	Середня	Середня	Висока	Висока
Придатність для МСА	Обмежена	Часткова	Перспективна	Висока	Висока

Аналіз, даних представлених у таблиці 2.3, дозволяє зробити наступні узагальнення.

Статичні підходи забезпечують простоту та прозорість, однак не відповідають вимогам динамічних середовищ МСА через відсутність адаптивності [26–28].

Оптимізаційні та евристичні методи дозволяють формалізувати процес прийняття рішень, але їх застосування ускладнюється необхідністю визначення адекватних функцій оптимізації та високими обчислювальними витратами [43, 46–48].

Методи машинного навчання демонструють високий потенціал, особливо в задачах прогнозування та проактивного управління, проте їх використання обмежується потребою у значних обсягах даних та складністю інтеграції у runtime-середовище [70, 75, 76, 89].

Self-adaptive системи забезпечують комплексний підхід до автоматизації управління, але характеризуються високою архітектурною складністю [70, 72, 73, 90].

Методи CBR демонструють оптимальний баланс між: адаптивністю, швидкодією, пояснюваністю рішень, та можливістю використання накопиченого досвіду [93].

У цьому контексті CBR є найбільш придатним підходом для реалізації інтелектуального адаптивного управління конфігураціями мікросервісних систем [1-3, 10–13].

Також була проведена багатокритеріальна оцінка методів адаптивного управління. Методологія оцінювання була наступна.

Для кількісного порівняння методів адаптивного управління був введений інтегральний показник ефективності:

$$Q = \sum_{1-1}^{n} w_i \cdot s_i, \tag{2.1}$$

де: w_i - ваговий коефіцієнт і-го критерію;

s_i - оцінка методу за і-м критерієм (за шкалою 1–5);

n - кількість критеріїв.

Для задач АУКМ було обрано критерії вказані в Таблиці 2.4 де сума всіх вагових коефіцієнтів дорівнює 1.

Таблиця 2.4 Вагові коефіцієнти критеріїв оцінки методів АУКМ

№	Критерій	Позначення	Вага
1	Рівень адаптивності	w_1	0.20
2	Робота в реальному часі	w_2	0.15
3	Обчислювальна ефективність	w_3	0.10
4	Пояснюваність рішень	w_4	0.10
5	Складність реалізації	w_5	0.10
6	Використання досвіду (knowledge reuse)	w_6	0.15
7	Стійкість до змін середовища	w_7	0.10
8	Придатність для МСА	w_8	0.10

У таблиці 2.5 надані загальні дані вагової оцінки методів адаптивного управління

Таблиця 2.5 – Вагова оцінка методів адаптивного управління

Метод	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	Інтегральна оцінка (Q)
Rule-based	1	2	5	5	5	1	2	2	**2,55**
Оптимізаційні	3	3	3	3	2	2	3	3	**2,80**
ML / RL	5	3	2	2	1	4	4	4	**3,25**
Self-adaptive (MAPE-K)	5	5	3	3	2	3	5	5	**3,95**
CBR	5	5	4	5	3	5	5	5	**4,65**

Отримані результати демонструють наступне:

Rule-based підходи (Q = 2.55) мають найнижчу інтегральну оцінку через відсутність адаптивності та використання досвіду.

Оптимізаційні методи (Q = 2.80) забезпечують покращення за рахунок формалізації, але поступаються за адаптивністю та гнучкістю.

ML/RL (Q = 3.25) демонструють високу адаптивність, але мають суттєві недоліки: складність, низьку пояснюваність, залежність від даних.

Self-adaptive системи (Q = 3.95) забезпечують високий рівень автоматизації, однак їх складність знижує загальну ефективність.

CBR (Q = 4.65) отримав найвищу оцінку завдяки високій адаптивності, можливості працювати в реальному часі, використанню попереднього досвіду, високій пояснюваності і хорошому балансу між складністю та ефективністю.

Проведена багатокритеріальна оцінка підтверджує, що метод аналізу прецедентів (CBR) є найбільш збалансованим підходом для задач адаптивного управління конфігураціями мікросервісних систем. Отримане значення інтегрального показника ефективності Q = 4.65 суттєво перевищує відповідні значення для альтернативних підходів, що обґрунтовує доцільність вибору CBR як базового методу для подальшого дослідження та розробки інтелектуального інструментарію.

2.3 Огляд та класифікація методів та інструментальні засоби для управління конфігураціями МСА

Управління конфігураціями в програмних системах з мікросервісною архітектурою (МСА) є складною багаторівневою задачею, що охоплює процеси визначення, зберігання, поширення та адаптації параметрів функціонування сервісів у динамічному середовищі [1, 3, 8, 37, 39].

На відміну від монолітних систем, у МСА конфігурації розподілені між сервісами, існують складні міжсервісні залежності, параметри конфігурації можуть змінюватися у runtime [34, 37, 63]. Це обумовлює необхідність використання спеціалізованих методів та інструментальних засобів, які забезпечують:

- централізоване або децентралізоване управління;
- узгодженість конфігурацій;
- підтримку адаптації до змін середовища .

На основі аналізу сучасних досліджень [9–12] та узагальнення підходів, розглянутих у підрозділі 2.2, методи управління конфігураціями доцільно класифікувати за наступними групами.

1. *Декларативні методи (Declarative configuration),* які базуються на описі бажаного стану системи у вигляді декларацій (YAML, JSON) [34, 36, 37]. Використовуються у Kubernetes, Terraform, Helm.

2. *Імперативні методи (Imperative configuration),* які передбачають явне визначення послідовності дій для зміни конфігурації [26–28]. Застосовуються у Ansible, Chef.

3. *Rule-based методи* використовують набір правил (if–then) для управління конфігураціями [26–28, 72]. Обмежені статичністю та складністю масштабування.

4. *Автоматизовані (policy-driven) методи*, базуються на політиках управління та SLA. Широко застосовуються у хмарних платформах [72, 73].

5. *Інтелектуальні методи*, які включають ML, RL, CBR, забезпечують адаптацію до змін середовища та використання історичних даних [10-13, 75, 76].

На рисунку 2.3 наведено узагальнену класифікацію методів управління конфігураціями мікросервісних архітектур. На відміну від декларативних, імперативних, rule-based та policy-driven підходів, інтелектуальні методи використовують знання, дані або попередній досвід для прийняття рішень.

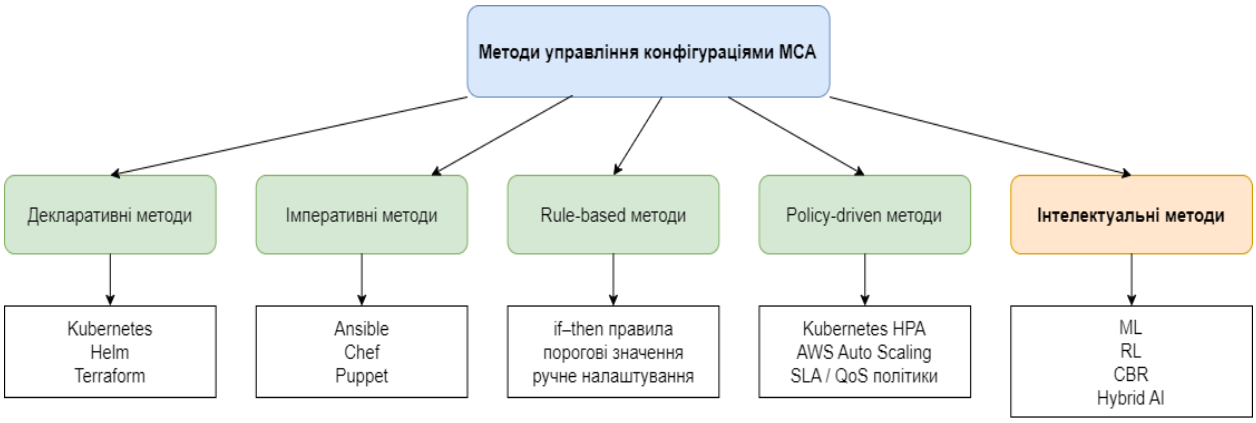


Рисунок 2.3 – Класифікація методів управління конфігураціями МСА

Інструментальні засоби управління конфігураціями можуть бути класифіковані за функціональним призначенням.

1. *Системи централізованого управління конфігураціями* Spring Cloud Config, Consul, AWS Config. Вони забезпечують централізоване зберігання та розповсюдження конфігурацій [34, 36, 37].

2. *Оркестратори контейнерів* Kubernetes, Docker Swarm, управляють життєвим циклом сервісів, забезпечують масштабування та конфігурування через ConfigMaps і Secrets [34, 36, 37, 39].

3. *Інструменти автоматизації (Infrastructure as Code)* Terraform, Ansible, Chef забезпечують автоматизацію розгортання, керування інфраструктурою [29–31].

4. *Service Mesh платформи* Istio, Linkerd управляють трафіком, конфігурують взаємодію сервісів [34, 36, 37].

5. *Інтелектуальні системи управління* - CBR-based система, що досліджується, ML-based рішення [10–13, 70, 75].

У таблиці 2.6 наведено класифікацію інструментальних засобів управління конфігураціями програмних систем з мікросервісною архітектурою за функціональним призначенням, рівнем адаптивності та можливістю роботи у runtime-середовищі.

Аналіз представлених даних показує, що більшість існуючих інструментів орієнтовані на вирішення окремих аспектів управління конфігураціями. Зокрема, централізовані системи управління конфігураціями забезпечують ефективне зберігання та розповсюдження параметрів, однак не підтримують механізми адаптації до змін середовища [36, 37]. Оркестратори контейнерів, такі як Kubernetes, розширюють ці можливості за рахунок автоматизації масштабування та підтримки runtime-змін, проте їх функціональність здебільшого базується на статичних або наперед визначених правилах [26–28].

Таблиця 2.6 – Класифікація інструментальних засобів управління конфігураціями МСА

Група інструментів	Представники	Основні функції	Рівень адаптивності	Підтримка runtime-змін	Основні обмеження
Централізовані системи управління конфігураціями (CM)	Spring Cloud Config, Consul, AWS Config	Централізоване зберігання конфігурацій, їх розповсюдження між сервісами	Низький	Частково	Відсутність автоматичної адаптації, залежність від ручного втручання
Оркестратори контейнерів	Kubernetes, Docker Swarm	Управління життєвим циклом сервісів, масштабування, конфігурація через ConfigMaps та Secrets	Середній	Так	Обмежена інтелектуальність, статичні правила масштабування
Інструменти Infrastructure as Code (IaC)	Terraform, Ansible, Chef	Автоматизація розгортання інфраструктури, управління конфігураціями на етапі deployment	Низький	Ні	Орієнтація на статичні сценарії, відсутність runtime-адаптації
Service Mesh платформи	Istio, Linkerd	Управління трафіком, політики безпеки, конфігурація взаємодії сервісів	Середній	Так	Висока складність, накладні витрати на інфраструктуру
Policy-driven системи	Kubernetes HPA, AWS Auto Scaling	Автоматичне масштабування на основі заданих політик та метрик	Середній	Так	Обмежена гнучкість, залежність від попередньо визначених правил
Інтелектуальні системи управління	CBR-системи, ML/RL системи, дослідницькі прототипи	Адаптивне управління конфігураціями на основі даних та досвіду	Високий	Так	Складність реалізації, потреба в інтеграції з інфраструктурою

Інструменти класу Infrastructure as Code (IaC) орієнтовані на автоматизацію процесів розгортання, що робить їх ефективними на етапі deployment, але малопридатними для адаптивного управління під час виконання системи. Service Mesh платформи забезпечують більш гнучке

управління взаємодією сервісів, однак їх використання пов’язане з підвищеною складністю та накладними витратами [34, 36, 37].

Policy-driven системи реалізують часткову адаптивність шляхом автоматичного масштабування, проте залишаються обмеженими рамками заданих політик і не враховують накопичений досвід функціонування системи [72, 73].

Найбільш перспективними є інтелектуальні системи управління, які використовують методи штучного інтелекту та аналізу даних [70, 75, 76]. Зокрема, CBR-підхід дозволяє враховувати історичні прецеденти та забезпечує адаптацію конфігурацій у реальному часі, що робить його придатним для використання в умовах динамічних мікросервісних середовищ.

Таким чином, результати класифікації підтверджують, що існуючі інструментальні засоби не забезпечують повною мірою інтелектуальне адаптивне управління конфігураціями, що обґрунтовує доцільність розробки нових підходів, зокрема на основі методу аналізу прецедентів [10–13, 70].

Аналіз популярних інструментів управління конфігураціями

Kubernetes ConfigMaps та Secrets є де-факто стандартом оркестрації контейнерів [34, 36, 37]. ConfigMaps та Secrets дозволяють відокремлювати конфігурацію від коду, централізовано керувати параметрами . До недоліків використання можна віднести відсутність адаптивності, статичність конфігурацій [26–28].

Spring Cloud Config забезпечує централізоване управління конфігураціями для мікросервісів [34, 36, 37]. Підтримує Git-based конфігурації, оновлення у runtime. Недоліки - обмежена автоматизація, відсутність інтелектуального аналізу [26–28].

HashiCorp Consul є системою сервіс-дискавері та управління конфігураціями [34, 36]. Переваги: розподіленість; підтримка key-value конфігурацій. Недоліки: складність інтеграції; обмежена адаптивність [72, 73].

Terraform та Ansible - *і*нструменти реалізують підхід Infrastructure as Code. Переваги: автоматизація; відтворюваність середовища. Недоліки: орієнтація на deployment, а не runtime-адаптацію [29–31].

*Service Mesh (Istio) з*абезпечує управління трафіком і конфігураціями взаємодії. Переваги: гнучке управління; політики безпеки. Недоліки: висока складність; накладні витрати [34, 36, 37].

У таблиці 2.7 наведені узагальнені результати аналізу сучасних інструментальних засобів управління конфігураціями в програмних системах з мікросервісною архітектурою, що дозволило визначити їх сильні і слабкі сторони з точки зору забезпечення адаптивного функціонування.

Таблиця 2.7 – Порівняння інструментів управління конфігураціями

Інструмент	**Тип**	**Адаптивність**	**Runtime зміни**	**Інтелектуаль-ність**	**Складність**
Kubernetes	Оркестратор	Середня	Так	Ні	Висока
Consul	CM система	Низька	Так	Ні	Середня
Spring Config	CM система	Низька	Частково	Ні	Середня
Terraform	IaC	Низька	Ні	Ні	Середня
Istio	Service Mesh	Середня	Так	Ні	Висока
CBR-система	Інтелектуальна	Висока	Так	Так	Середня

Як видно з таблиці 2.7, більшість розглянутих інструментів, зокрема Kubernetes, Consul та Spring Cloud Config, забезпечують підтримку runtime-змін конфігурацій, що є критично важливим для мікросервісних систем. Однак їх адаптивність переважно обмежується використанням заздалегідь визначених правил або політик, що знижує ефективність у випадках динамічних або непередбачуваних змін середовища [72, 73].

Зокрема, Kubernetes реалізує механізми автоматичного масштабування (наприклад, Horizontal Pod Autoscaler), що базуються на аналізі метрик (CPU, memory), але не враховує історичний досвід функціонування системи та не забезпечує інтелектуального вибору конфігурацій [34, 36, 37, 59]. Аналогічно,

системи типу Consul та Spring Cloud Config орієнтовані на централізоване управління конфігураціями, проте не містять механізмів автоматичного прийняття рішень щодо їх адаптації [26–28].

Інструменти класу Infrastructure as Code, такі як Terraform, демонструють ефективність на етапі розгортання системи, забезпечуючи відтворюваність та автоматизацію конфігурацій. Водночас вони не підтримують адаптацію конфігурацій у процесі виконання системи, що суттєво обмежує їх застосування у динамічних середовищах МСА [29–31].

Service Mesh платформи, зокрема Istio, забезпечують більш гнучке управління взаємодією між сервісами, включаючи маршрутизацію трафіку, балансування навантаження та політики безпеки. Однак їх функціональність також базується на статичних або наперед заданих правилах, що не дозволяє повною мірою реалізувати адаптивне управління [72, 73].

Виходячи з цього, можна зробити висновок, що більшість існуючих інструментів орієнтовані на статичне управління [26–28], не використовують історичні дані [29, 33, 68], не підтримують інтелектуальне прийняття рішень [70, 75, 76] і не забезпечують комплексного вирішення задачі адаптивного управління конфігураціями мікросервісних систем. Це призводить до зниження ефективності роботи ПС, перевитрати ресурсів і необхідності ручного втручання [72, 73].

На відміну від зазначених підходів, інтелектуальні системи управління, представлені у таблиці 2.7, демонструють принципово інший рівень можливостей. Це дозволяє розглядати CBR як перспективну основу для побудови систем адаптивного управління конфігураціями в МСА [10–13, 69].

Важливим результатом аналізу є також виявлення ключового протиріччя - сучасні інструментальні засоби забезпечують високу автоматизацію та масштабованість, але не забезпечують достатнього рівня інтелектуальності та адаптивності, що свідчить про необхідність інтеграції інтелектуальних методів у процеси управління конфігураціями [70, 72, 75]. Найбільш перспективним напрямом є використання інтелектуальних підходів,

зокрема методу аналізу прецедентів, що і визначає подальший напрям дослідження, розглянутий у підрозділі 2.4 [10–13].

2.4 Інтелектуальний підхід до управління функціонуванням МСА на основі використання методу аналізу прецедентів

Результати аналізу, проведеного у підрозділах 2.1–2.3, показали, що сучасні програмні системи з мікросервісною архітектурою функціонують у складному динамічному середовищі, яке характеризується змінним навантаженням, неоднорідністю обчислювальних ресурсів, високою інтенсивністю міжсервісної взаємодії та значною кількістю конфігураційних параметрів [34, 36, 37, 63]. За таких умов традиційні підходи до управління, що базуються на статичних правилах, наперед визначених політиках або ручному коригуванні параметрів, виявляються недостатньо гнучкими та не забезпечують належного рівня адаптивності [26–28, 72, 73].

Особливістю задачі управління функціонуванням МСА є те, що вона має слабко формалізований і багатокритеріальний характер. Прийняття рішення щодо зміни конфігурації або способу реагування на зміну стану системи залежить не від одного показника, а від сукупності взаємопов’язаних факторів, серед яких:

- поточне завантаження обчислювальних ресурсів;
- затримки у взаємодії сервісів;
- доступність окремих компонентів;
- стабільність виконання бізнес-операцій;
- вартість використання інфраструктури.

Крім того, у мікросервісних системах одна й та сама конфігурація може виявляти різну ефективність залежно від контексту її застосування, що ускладнює використання суто детермінованих або жорстко формалізованих підходів. Саме тому доцільним є залучення інтелектуальних методів, здатних враховувати попередній досвід функціонування системи, працювати з

неповною або нечіткою інформацією та підтримувати адаптивне прийняття рішень у runtime-середовищі [70, 75, 76].

У цьому контексті метод аналізу прецедентів розглядається як один із найбільш перспективних підходів для управління функціонуванням МСА, оскільки дозволяє використовувати накопичений досвід системи у вигляді прецедентів та формувати нові рішення шляхом пошуку і адаптації аналогічних випадків.

Метод аналізу прецедентів належить до класу knowledge-based та experience-driven підходів, у яких основним джерелом знань є не наперед закладені правила, а накопичений досвід вирішення аналогічних задач [10–13, 69, 70]. Для задач управління функціонуванням МСА це означає, що система може: фіксувати попередні стани функціонування; зберігати пов’язані з ними конфігурації; оцінювати ефективність застосованих рішень; використовувати релевантний попередній досвід для адаптації в нових умовах [10–13, 69].

Перевага CBR у порівнянні з rule-based та ML-підходами полягає у тому, що він:

- не потребує повного перенавчання моделі при накопиченні нових даних;
- забезпечує вищий рівень пояснюваності рішень;
- добре узгоджується з багатовимірною природою опису стану МСА;
- дозволяє інтегрувати як кількісні метрики, так і контекстні характеристики середовища [10–13, 43, 46–48].

Таким чином, CBR доцільно розглядати не лише як окремий метод пошуку схожих випадків, а *як методологічну основу інтелектуального управління*, у якій досвід функціонування системи стає центральним джерелом знань для прийняття рішень.

На рисунку 2.4 наведено загальну схему інтелектуального підходу до управління конфігураціями мікросервісної архітектури на основі методу аналізу прецедентів. У схемі відображено взаємозв’язок між підсистемою моніторингу, модулем формування поточної ситуації, базою прецедентів, механізмами пошуку та адаптації рішень, а також модулем застосування

конфігураційних змін. Така схема відображає методологічну основу підходу без переходу до детальної алгоритмізації, яка розглядається у наступному розділі.

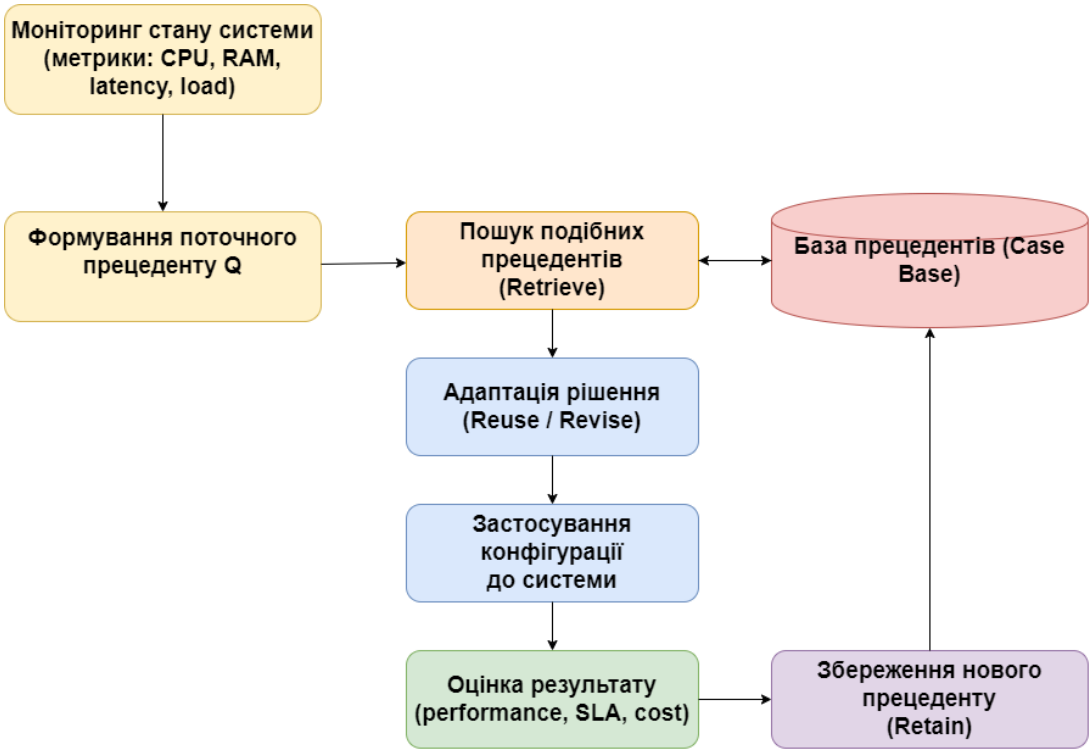


Рисунок 2.4 – Загальна схема інтелектуального підходу до управління функціонуванням МСА на основі методу аналізу прецедентів

Адаптація методу аналізу прецедентів до задач управління функціонуванням МСА потребує врахування специфіки цієї архітектури. На основі аналізу предметної області можна сформулювати такі методологічні принципи побудови CBR-підходу.

1) Принцип багатовимірного опису стану системи

Стан мікросервісної системи не може бути зведений до одного параметра. Він повинен описуватись набором ознак, що охоплюють: ресурсний стан; продуктивність; стабільність; характеристики середовища; поточну конфігурацію.

2) Принцип контекстної залежності рішень

Ефективність конфігурації визначається не лише її внутрішніми параметрами, а й умовами, у яких вона застосовується. Тому прецедент повинен враховувати не тільки “що було зроблено”, а й “у яких умовах це дало позитивний результат”.

3) Принцип повторного використання досвіду

Замість повторного пошуку рішення в кожній новій ситуації система повинна використовувати вже наявний досвід, накопичений у базі прецедентів. Це скорочує час реакції та знижує обчислювальні витрати, .

4) Принцип адаптації замість прямого копіювання

У більшості випадків знайдений прецедент не може бути використаний безпосередньо. Він повинен бути адаптований до поточного стану системи, що є важливою відмінністю CBR від простих пошукових підходів, .

5) Принцип накопичення нових знань

Кожен новий випадок функціонування системи повинен розглядатися як потенційне джерело знань для майбутніх рішень. Це дозволяє системі не лише реагувати на зміну умов, а й поступово підвищувати якість управління.

Методологічна схема обчислення подібності прецедентів

Центральним елементом CBR-підходу є оцінка подібності між поточним станом системи та наявними прецедентами. Саме від якості цього етапу залежить релевантність знайденого рішення та ефективність подальшої адаптації [10–13, 69, 70].

Для формалізації процедури порівняння прецедентів доцільно представити їх у вигляді точок у багатовимірному просторі ознак [29, 33, 68].

На рисунку 2.5 представлено інтерпретацію прецедентів у вигляді багатовимірного простору ознак, що використовується в межах методу аналізу прецедентів для задач адаптивного управління конфігураціями мікросервісних систем. Кожен прецедент у такому підході описується набором параметрів, які характеризують поточний стан системи, умови її функціонування, а також відповідну конфігурацію, що була застосована.

Кожен прецедент може бути представлений у вигляді вектору

$$\vec{C}_i = (c_{i1},\ c_{i2}, \dots, c_{in}), \tag{2.2}$$

де c_{ij} - значення *j-ї* ознаки для *i*-го прецеденту. Аналогічно, поточний стан системи (запит) описується вектором $\vec{Q} = (q_1, q_2,\ \dots, q_n)$ [29, 33, 68].

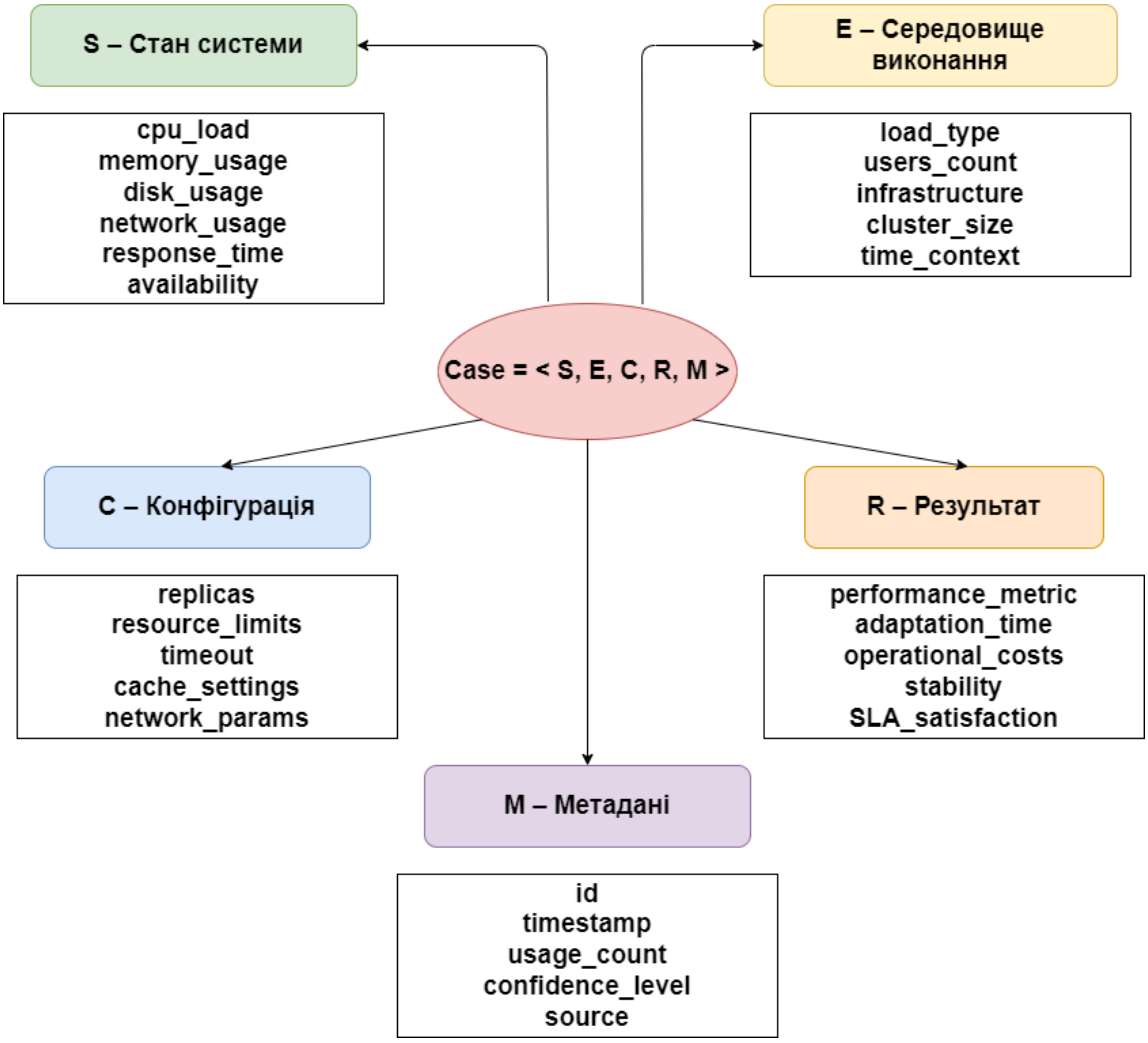


Рисунок 2.5 – Багатовимірний простір прецедентів

Кожна координатна вісь у цьому просторі відповідає окремій характеристиці системи, наприклад:

– рівню завантаження процесора;

– використанню оперативної пам’яті;

– інтенсивності мережевого трафіку;

- часу відгуку сервісів;
- кількості активних запитів.

Таким чином, множина всіх прецедентів формує дискретний простір точок, у якому задача пошуку найбільш релевантного рішення зводиться до визначення прецедентів, що знаходяться на мінімальній відстані від поточного запиту. Геометрична інтерпретація дозволяє формалізувати задачу пошуку як задачу у просторі ознак, застосовувати різні метрики подібності та забезпечити узагальнене представлення знань про поведінку системи [29, 33].

Крім того, такий підхід створює основу для подальшого використання методів машинного навчання, кластеризації та індексації прецедентів, що є важливим для підвищення ефективності обробки великих обсягів конфігураційних даних у мікросервісних архітектурах.

Варто зазначити, що саме така інтерпретація прецедентів є ключовою для реалізації процедури обчислення подібності, яка розглядається далі (рисунок 2.6), а також для побудови алгоритмічної моделі адаптивного управління у наступному розділі дисертаційної роботи.

У методологічному сенсі обчислення подібності слід розглядати як багатокритеріальний процес порівняння поточного запиту з набором історичних випадків за множиною ознак, що мають різну інформативність. Одні ознаки можуть бути критичними для вибору рішення, інші - допоміжними, тому доцільно використовувати систему вагових коефіцієнтів [43, 46–48, 76].

На рисунку 2.6 наведено узагальнену схему обчислення подібності між поточним станом мікросервісної системи та прецедентами, що зберігаються у базі знань. На першому етапі формується поточний запит $\vec{Q}$, який описує стан системи за множиною ознак $q_1, q_2, ..., q_n$. Далі цей запит порівнюється з кожним прецедентом $\vec{C}_i$, що міститься у базі прецедентів.

Порівняння здійснюється за окремими ознаками шляхом обчислення локальних функцій подібності $sim_j(q_j, c_{ij})$. Оскільки різні ознаки можуть мати

неоднаковий вплив на релевантність рішення, для їх урахування використовується система вагових коефіцієнтів w_j. Після цього виконується агрегування локальних оцінок і формується інтегральна міра подібності $Sim(\vec{Q},\ \vec{C}_i)$.

Отримані значення використовуються для ранжування прецедентів, після чого вибирається найбільш релевантний прецедент $\overrightarrow{\boldsymbol{C}^*}$, який надалі застосовується як основа для формування адаптивного рішення щодо зміни конфігурації мікросервісної системи. Таким чином, дана схема відображає методологічний принцип роботи CBR-підходу на етапі пошуку релевантного досвіду.

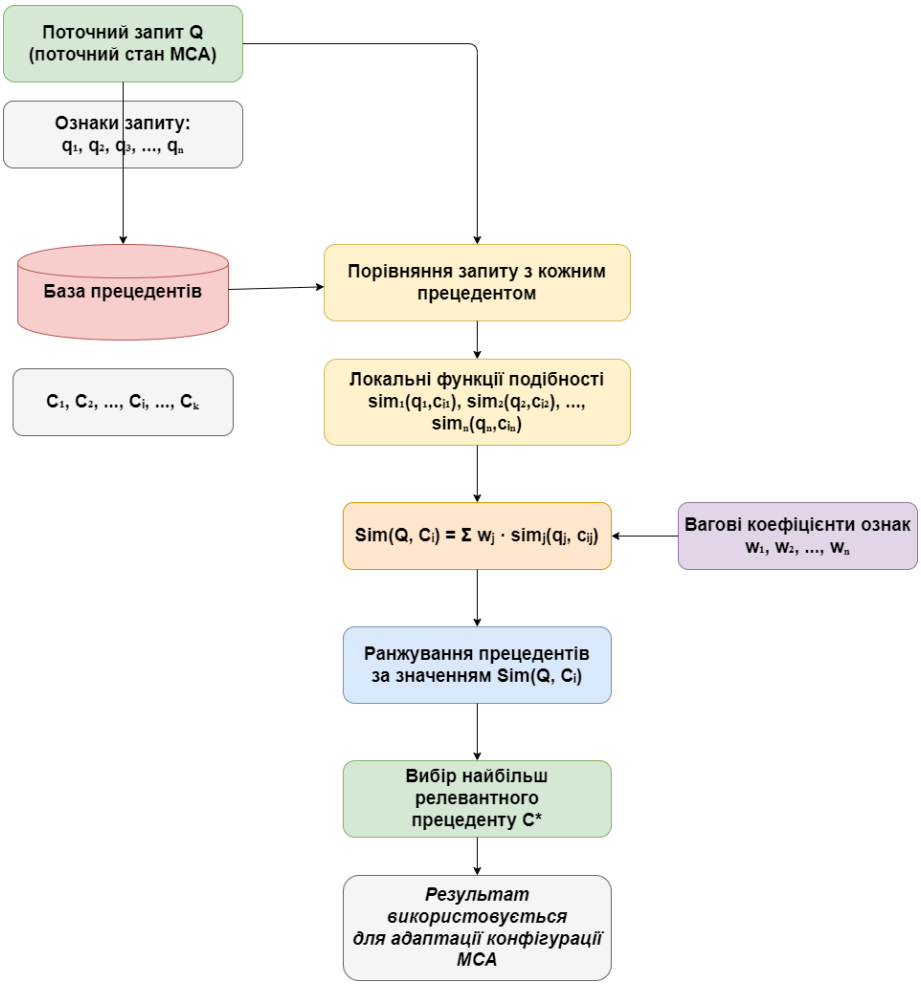


Рисунок 2.6 – Схема обчислення подібності прецедентів

Для реалізації CBR-підходу необхідно, щоб прецедент мав достатньо повний і водночас формалізований опис. З методологічної точки зору прецедент є структурованою моделлю одиниці досвіду, яка поєднує:

– опис ситуації;
– умови функціонування;
– конфігурацію;
– результат її застосування;
– службову інформацію,, .

Саме така інтерпретація прецеденту робить можливим подальшу побудову алгоритмічної моделі пошуку та адаптації, формування багатовимірного інформаційного базису, розробку архітектури інструментального засобу.

На рисунку 2.7 показано узагальнену роль прецеденту як центрального елемента інтелектуального підходу до управління конфігураціями мікросервісної архітектури. На відміну від традиційних підходів, де рішення формується на основі статичних правил або наперед визначених політик, у CBR-підході саме прецедент виступає носієм структурованого досвіду про попереднє функціонування системи.

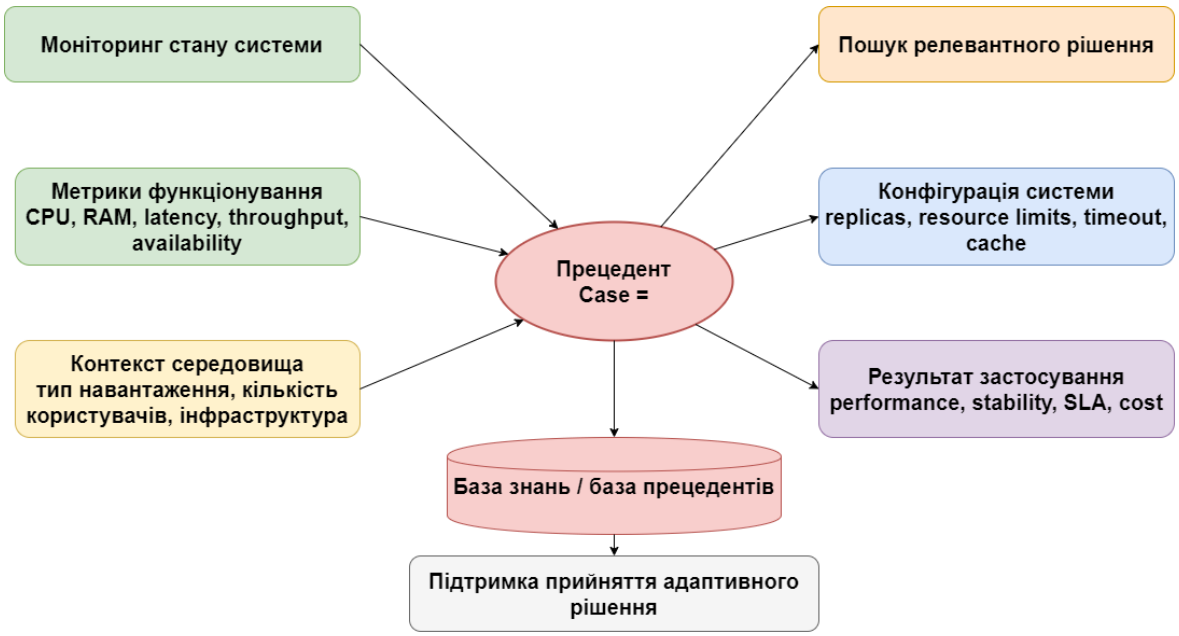


Рисунок 2.7 – Роль прецеденту в інтелектуальному підході

Як видно з рисунка, прецедент інтегрує декілька груп інформації. З одного боку, він формується на основі даних моніторингу та множини метрик, що характеризують поточний стан системи, зокрема навантаження на обчислювальні ресурси, затримки відповіді, пропускну здатність та доступність сервісів. З іншого боку, прецедент враховує контекст функціонування, до якого належать тип навантаження, кількість активних користувачів, часові характеристики та особливості інфраструктури виконання.

Центральне розміщення прецеденту на схемі відображає його зв'язуючу роль між вхідними даними системи та результатами інтелектуального аналізу. Саме через структуру прецеденту стає можливим поєднання опису ситуації, параметрів конфігурації та результатів їх застосування в єдиній моделі знань. Окреме місце на рисунку відведено базі знань або базі прецедентів. Це означає, що інтелектуальний підхід не обмежується одноразовим аналізом поточної ситуації, а передбачає накопичення та повторне використання попереднього досвіду функціонування системи. Таким чином, кожен новий прецедент не лише використовується для вирішення поточної задачі, а й поповнює знання системи для майбутніх рішень.

У методологічному контексті рисунок 2.7 виконує важливу роль, оскільки показує місце прецеденту в загальній логіці інтелектуального управління МСА від збору даних і формування ситуації до побудови бази знань і підтримки прийняття рішення. Це дозволяє розглядати прецедент не просто як запис у базі даних, а як центральну концептуальну конструкцію, на якій ґрунтується весь CBR-підхід.

Застосування методу аналізу прецедентів у задачах управління функціонуванням МСА має низку принципових переваг.

По-перше, CBR забезпечує *адаптивність у реальному часі*, оскільки рішення формується на основі вже накопиченого досвіду і не потребує повного перенавчання моделі [10–13, 69].

По-друге, CBR має вищий рівень *пояснюваності* у порівнянні з багатьма ML-підходами, оскільки кожне рішення можна співвіднести з конкретним історичним прецедентом [10–13, 69].

По-третє, цей підхід добре узгоджується з *модульною природою мікросервісних систем,* у яких стан системи та ефект від зміни конфігурації можуть бути описані через набір відносно незалежних, але взаємопов'язаних параметрів [29, 33].

По-четверте, CBR підтримує *накопичення знань* у процесі експлуатації, що дозволяє системі поступово підвищувати ефективність управління [10, 69].

У таблиці 2.8 надане узагальнення методологічних переваг використання CBR методу для управління функціонуванням МСА.

Таблиця 2.8 – Методологічні переваги використання CBR у задачах управління функціонуванням МСА

Перевага	**Зміст**
Адаптивність	Можливість реагування на зміну стану системи в runtime
Пояснюваність	Рішення може бути обґрунтоване через знайдений прецедент
Використання досвіду	Накопичення та повторне використання знань про функціонування системи
Гнучкість	Підтримка різнорідних типів ознак і контекстних факторів
Масштабованість	Придатність до використання у великих мікросервісних системах

Методологічні обмеження та перспективи розвитку підходу

Попри значні переваги, CBR-підхід має і певні методологічні обмеження. До них належать:

- залежність від якості та повноти бази прецедентів;
- складність вибору релевантного набору ознак;
- ризик зниження ефективності при надмірному зростанні бази знань;

– необхідність узгодження локальних і глобальних критеріїв подібності.

На мій погляд ці обмеження не знижують цінності самого підходу, а скоріше визначають напрями його подальшого розвитку. До таких напрямів можна віднести побудову ефективних алгоритмів індексації прецедентів, комбінування CBR з методами машинного навчання, формування багатовимірного інформаційного базису для підвищення якості пошуку, розробку архітектури інструментального засобу, що реалізує даний підхід.

Ключовим етапом методу аналізу прецедентів є етап Retrieve, який полягає у пошуку в базі знань такого прецеденту або множини прецедентів, що найбільш відповідають поточному стану системи. Саме якість виконання цього етапу визначає ефективність усього циклу CBR (Retrieve–Reuse–Revise–Retain), оскільки помилка на стадії вибору релевантного прецеденту призводить до накопичення неточностей на наступних етапах і, як наслідок, до зниження загальної якості адаптивного управління, .

У загальному вигляді задача пошуку прецеденту формулюється як задача знаходження такого елемента $Case* \in C$, для якого значення функції подібності між поточним станом системи $\vec{S}_q$ та станом, що відповідає прецеденту $\vec{S}_i$, є максимальним:

$$Case* = \underbrace{arg\ \ max\ \ Sim}_{Case_i \in C}(\vec{S}_q, \vec{S}_i), \tag{2.3}$$

де: C - множина прецедентів;

$Sim(\vec{S}_q, \vec{S}_i)$ - скалярне значення функції подібності;

$\vec{S}_q = (s_{q1}, s_{q2}, \dots s_{qn})$ - вектор поточного стану системи;

$\vec{S}_i = (s_{i1}, s_{i2}, \dots s_{in})$ - вектор стану для і-го прецеденту;

s_{qj}, s_{ij}- скалярні значення j-ї ознаки відповідних векторів.

У випадку використання множини найближчих прецедентів задача має вигляд:

$$N_k(\vec{S}_q) = \{Cese_i \in C \,|\, Sim(\vec{S}_q, \vec{S}_i) \in Top - k\}, \tag{2.4}$$

де $N_k(\vec{S}_q)$ - множина k найбільш подібних прецедентів до поточного стану $\vec{S}_q$.

Таким чином, етап Retrieve може бути інтерпретований як задача пошуку в багатовимірному просторі ознак, що вже було введено у попередніх підпунктах.

Важливо підкреслити, що задачі пошуку найбільш релевантних прецедентів не можуть бути ефективно розв'язані за допомогою єдиного універсального алгоритму. У цьому плані доцільним є використання різних підходів, кожен з яких має власні переваги та обмеження щодо точності, швидкодії та масштабованості. Саме тому у сучасних CBR-системах застосовується набір методів пошуку, що дозволяє адаптувати процес вибору прецедентів до конкретних умов задачі [10–13, 70, 75].

Nearest Neighbor (NN)

Метод найближчого сусіда є базовим підходом до реалізації етапу Retrieve та полягає у повному порівнянні вектора поточного стану $\vec{S}_q$ з векторами станів усіх прецедентів $\vec{S}_i$, що зберігаються в базі. Найчастіше функція подібності визначається через метрику відстані:

$$Sim(\vec{S}_q, \vec{S}_i) = \frac{1}{1+d(\vec{S}_q,\vec{S}_i)}, \quad (2.5)$$

де $d(\vec{S}_q, \vec{S}_i)$ - скалярне значення подібності;

$d(\vec{S}_q, \vec{S}_i) = \sqrt{\sum_{j=1}^{n}(s_{qj} - s_{ij})^2}$ - скалярне значення евклідової відстані між двома векторами станів.

Перевагою методу є висока точність, однак його обчислювальна складність становить $O(N)$ (де N - кількість прецедентів у базі) що обмежує застосування при великих обсягах даних [68, 69].

Weighted K-Nearest Neighbors (WKNN)

Метод зважених k-найближчих сусідів є узагальненням NN та передбачає врахування вагових коефіцієнтів ознак [43, 46]. Для цього вводиться вектор ваг: $\overrightarrow{W} = (w_1, w_{2,} \dots, w_n)$, де w_j - скалярна вага j-ї ознаки. Тоді зважена евклідова відстань між поточним станом і прецедентом може бути подана так:

$$d_w(\vec{S}_q, \vec{S}_i) = \sqrt{\sum_{j=1}^{n} w_j (s_{qj} - s_{ij})^2}. \quad (2.6)$$

Після визначення множини k найближчих прецедентів формується агреговане конфігураційне рішення, наприклад:

$$\vec{C}^* = \sum_{Case_i \in N_k(\vec{s}_q)} a_i \vec{C}_i,$$

де $\vec{C}^*$ - шуканий вектор рекомендованої конфігурації;

$\vec{C}_i$ - вектор конфігурації і-го прецеденту;

a_i - скалярний ваговий коефіцієнт і-го прецеденту.

Цей підхід дозволяє враховувати важливість різних параметрів системи та підвищує точність пошуку в умовах високої розмірності простору [68, 69].

Feature-Based Retrieval

У цьому підході пошук здійснюється не за повним вектором стану $\vec{S}$, а за його підвектором, сформованим із найбільш значущих ознак [29, 33, 68]. Якщо позначити множину вибраних ознак як: $F \subseteq \{1,2,\dots,n\}$, тоді підвектор поточного стану можна записати як: $\vec{S}_q^F = (s_{qj})_{j \in F}$, а підвектор стану *i*-го прецеденту: $\vec{S}_i^F = (s_{ij})_{j \in F}$. Функція подібності за вибраними ознаками може бути подана так:

$$Sim_F(\vec{S}_q, \vec{S}_i) = \sum_{j \in F} w_i \cdot sim_j(s_{qj}, s_{ij}), \quad (2.7)$$

де $Sim_F(\vec{S}_q, \vec{S}_i)$ - скалярне значення подібності за вибраною множиною ознак;

$sim_j(s_{qj}, s_{ij})$ - локальна скалярна функція подібності для *j*-ї ознаки.

Цей метод дозволяє зменшити розмірність порівняння, однак його ефективність значною мірою залежить від коректності вибору множини ознак *F*. Складність такого підходу можна оцінити як $O(N \cdot |F|)$, де $|F|$ - кількість вибраних ознак [68, 69].

Cluster-Based Retrieval

Метод кластерного пошуку передбачає попереднє групування прецедентів у кластери відповідно до подібності їхніх векторів станів [29, 33,

68, 69]. Множину прецедентів можна подати як об'єднання кластерів: $C = \bigcup_{k=1}^{K}(C_k)$,де C_k - k-й кластер прецедентів; K - кількість кластерів. Для кожного кластера може бути визначений вектор центроїда: $\vec{\mu}_k = (\mu_{k1}, \mu_{k1}, \dots, \mu_{kn})$.

Пошук виконується у два етапи:

1) визначення найближчого кластера: $k^* = \arg\min_k d(\vec{S}_q, \vec{\mu}_k)$, де k^* - індекс найближчого кластера, а $\vec{\mu}_k$ - вектор центроїда k-го кластера.

2) пошук прецеденту всередині кластера:

$$Case^* = \arg\ max_{Case_i \in C_{k^*}}\ Sim(\vec{S}_q, \overrightarrow{S_i}) \tag{2.8}$$

Це дозволяє зменшити складність пошуку до: $O(log\ K\ +\ |C_{k^*}|)$

Такий підхід зменшує область пошуку та підвищує масштабованість системи, особливо за великої кількості прецедентів [68, 69].

Indexing & Hashing

Методи індексації та хешування спрямовані на прискорення доступу до релевантних прецедентів за рахунок використання допоміжних структур даних [29, 33, 68]. У загальному вигляді хеш-функція може бути подана як відображення вектора стану в індексну область: $h: \vec{S} \to Z$, де $h(\vec{S})$ - хеш-функція, що відображає схожі стани в одну область.

Для поточного стану $\vec{S}_q$формується хеш-значення $h_q = h(\vec{S}_q)$, після чого пошук виконується серед кандидатних прецедентів:

$$C_{h_q} = \{Case_i \in C \mid h(\vec{S}_i) = h(\vec{S}_q)\}.$$

Пошук виконується лише серед кандидатів з однаковими або близькими значеннями хешу:

$$Case^* = \arg\ max_{Case_i \in C_{h_q{}^*}}\ Sim(\vec{S}_q, \overrightarrow{S_i}). \tag{2.9}$$

Методи індексації та хешування дозволяють значно скоротити простір пошуку, що є особливо важливим для систем, які мають працювати в режимі,

близькому до реального часу. Це дозволяє зменшити складність пошуку до $O(1)$), що є критично важливим для систем реального часу [70, 75, 76, 87. 91].

У таблиці 2.10 наведено порівняльну характеристику розглянутих методів.

Таблиця 2.10 – Порівняння методів пошуку прецедентів

Метод	**Точність**	**Складність**	**Масштабованість**	**Особливості**
NN	Висока	$O(N)$	Низька	Повний перебір
WKNN	Висока	$O(N)$	Середня	Ваги ознак
Feature-Based	Середня	$O(N \cdot \|F\|)$	Середня	Вибір найбільш релевантних ознак, зменшення розмірності порівняння
Cluster-Based	Середня/висока	$O(log\ N)$	Висока	Кластеризація
Indexing & Hashing	Середня	$O(1)$	Дуже висока	Індекси

Таким чином, етап Retrieve у CBR-системах є визначальним з точки зору якості адаптивного управління, а вибір конкретного методу пошуку повинен здійснюватися з урахуванням розміру бази прецедентів, вимог до швидкодії та характеру даних. Використання комбінованих підходів дозволяє досягти компромісу між точністю та обчислювальною ефективністю [70, 75].

Якість реалізації етапу Retrieve визначає ефективність усього CBR-підходу, що обґрунтовує необхідність використання різних методів пошуку прецедентів залежно від умов функціонування системи.

2.5 Висновки до Розділу 2

У Розділі 2 сформовано методологічні основи розробки інтелектуального модельно-технологічного інструментарію для адаптивного управління програмними мікросервісами, а саме:

– було сформовано концептуальну основу дослідження;

- обґрунтовано вибір CBR як базового інтелектуального підходу;
- визначено місце структури прецеденту, механізмів подібності та багатовимірного опису станів у загальній методології;
- підготовлено теоретичне підґрунтя для розробки алгоритмічної моделі та архітектури інструментальних засобів адаптивного управління конфігураціями МСА.

У межах розвитку методологічних основ застосування методу CBR у роботі додатково досліджено підходи до реалізації етапу Retrieve, який є ключовим у CBR-циклі та визначає якість прийняття рішень. Розглянуто основні методи пошуку прецедентів, зокрема Nearest Neighbor, Weighted K-Nearest Neighbors, Feature-Based Retrieval, Cluster-Based Retrieval та Indexing & Hashing, які відрізняються обчислювальною складністю, точністю та масштабованістю. Показано, що задачі пошуку релевантних прецедентів не мають універсального розв’язання та потребують вибору відповідного методу залежно від характеристик даних і вимог до системи. Отримані результати формують теоретичну основу для побудови алгоритмічної моделі адаптивного управління конфігураціями, розглянутої у наступному розділі.

Таким чином, результати отримані у Розділі 2 дають основу для подальшої розробки алгоритмічної моделі адаптивного управління конфігураціями МСА з використанням методу аналізу прецедентів та побудови концептуальної моделі багатовимірного інформаційного базису для її застосування, що буде розглянуто у Розділі 3.

РОЗДІЛ 3. РОЗРОБКА АЛГОРИТМІЧНОЇ МОДЕЛІ ТА АРХІТЕКТУРИ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ АДАПТИВНОГО УПРАВЛІННЯ КОНФІГУРАЦІЯМИ МСА

3.1 Розробка алгоритмічної моделі для адаптивного управління конфігураціями МСА з використанням методу аналізу прецедентів

У Розділі 1 було встановлено, що однією з ключових проблем сучасних програмних систем з мікросервісною архітектурою є забезпечення ефективного управління конфігураціями в умовах динамічного навантаження та змін середовища функціонування. У Розділі 2 обґрунтовано доцільність застосування інтелектуальних методів, зокрема методу аналізу прецедентів (Case-Based Reasoning, CBR), для розв’язання цієї задачі.

На основі отриманих результатів можна сформулювати задачу адаптивного управління конфігураціями як задачу вибору оптимального набору параметрів конфігурації мікросервісної системи на основі поточного стану системи та накопиченого досвіду її функціонування [43, 46–48, 91].

Припустимо:

S – простір станів системи;

C – простір можливих конфігурацій;

M – множина метрик;

R – множина результатів функціонування системи.

Тоді задача управління полягає у визначенні відображення $F: S \rightarrow C$, яке забезпечує досягнення оптимальних значень цільових метрик за наявних обмежень [88, 90].

Вхідними даними задачі є вектор стану системи, сформований на основі метрик продуктивності, використання ресурсів та параметрів середовища та історичні дані про функціонування системи (прецеденти) [59–61, 93].

Вихідними даними є конфігурація мікросервісів, що визначає параметри їх функціонування (кількість реплік, обмеження ресурсів, параметри балансування навантаження тощо) [34, 36, 37, 87].

У таблиці 3.1 наведено формалізацію основних параметрів задачі.

Таблиця 3.1 – Формалізація вхідних і вихідних параметрів задачі адаптивного управління

Позначення	**Назва**	**Опис**
S	Стан системи	Вектор метрик (CPU, RAM, latency, throughput тощо)
C	Конфігурація	Набір параметрів мікросервісів
M	Метрики	Показники ефективності функціонування
R	Результат	Досягнуті значення метрик після застосування конфігурації
F	Функція управління	Відображення стану у конфігурацію

Таким чином, задача адаптивного управління конфігураціями може бути інтерпретована як задача багатокритеріального вибору в умовах невизначеності, де рішення приймається на основі попереднього досвіду системи [43, 46, 76].

На рисунку 3.1 показано узагальнену схему задачі адаптивного управління конфігураціями мікросервісної архітектури, у межах якої процес прийняття рішень реалізується як замкнений контур керування. У лівій частині схеми відображено джерело вхідної інформації - поточний стан мікросервісної системи, який формується на основі сукупності ресурсних, продуктивнісних та експлуатаційних показників. Центральне місце займає модуль алгоритмічного прийняття рішень на основі методу аналізу прецедентів, який використовує як поточний опис ситуації, так і накопичений досвід функціонування системи, представлений у базі прецедентів. Результатом роботи цього модуля є формування керуючого впливу у вигляді нової або уточненої конфігурації. Права частина схеми відображає наслідок застосування конфігураційного рішення до МСА-застосунку, що проявляється

у зміні показників якості функціонування. Наявність зворотного зв’язку підкреслює циклічний характер адаптивного управління, коли результати попереднього циклу стають основою для наступного аналізу стану системи.

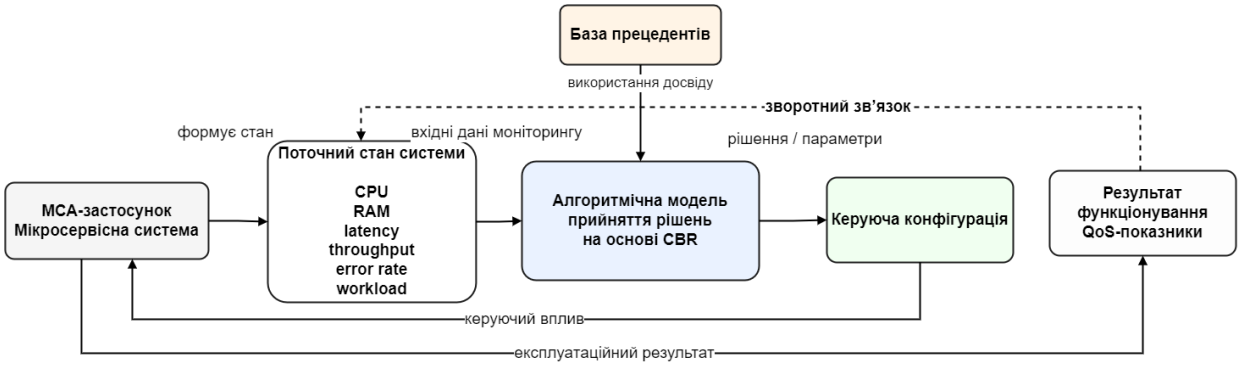


Рисунок 3.1 – Узагальнена схема задачі АУКМ

Таким чином задача адаптивного управління конфігураціями формалізована як відображення простору станів у простір конфігурацій з урахуванням множини метрик та накопиченого досвіду, що створює основу для побудови алгоритмічної моделі на базі CBR [10–13, 69, 93].

Ключовим елементом CBR-підходу є прецедент, який відображає попередній досвід функціонування системи у вигляді структурованого запису [1, 10, 69]. У контексті задачі адаптивного управління конфігураціями прецедент повинен містити інформацію про стан системи, застосовану конфігурацію та отриманий результат.

На рисунку 3.2 продемонстровано структурну організацію прецеденту як елементарної одиниці знань, що використовується в алгоритмічній моделі адаптивного управління конфігураціями МСА. Схема відображає, що прецедент складається з трьох базових компонентів: опису стану системи, конфігураційного рішення та результату його застосування. Така побудова дозволяє інтерпретувати прецедент як формалізований запис досвіду, що поєднує умови функціонування системи, прийняте керуюче рішення та досягнутий ефект. Окремо на схемі показані додаткові атрибути, які розширюють базову структуру прецеденту, зокрема контекст виконання,

часові характеристики, параметри середовища та тип сервісного сценарію. У сукупності це формує придатне для машинної обробки представлення знань, яке може бути використане для пошуку подібних ситуацій, порівняння альтернативних конфігурацій і подальшого поповнення бази досвіду [33, 68].

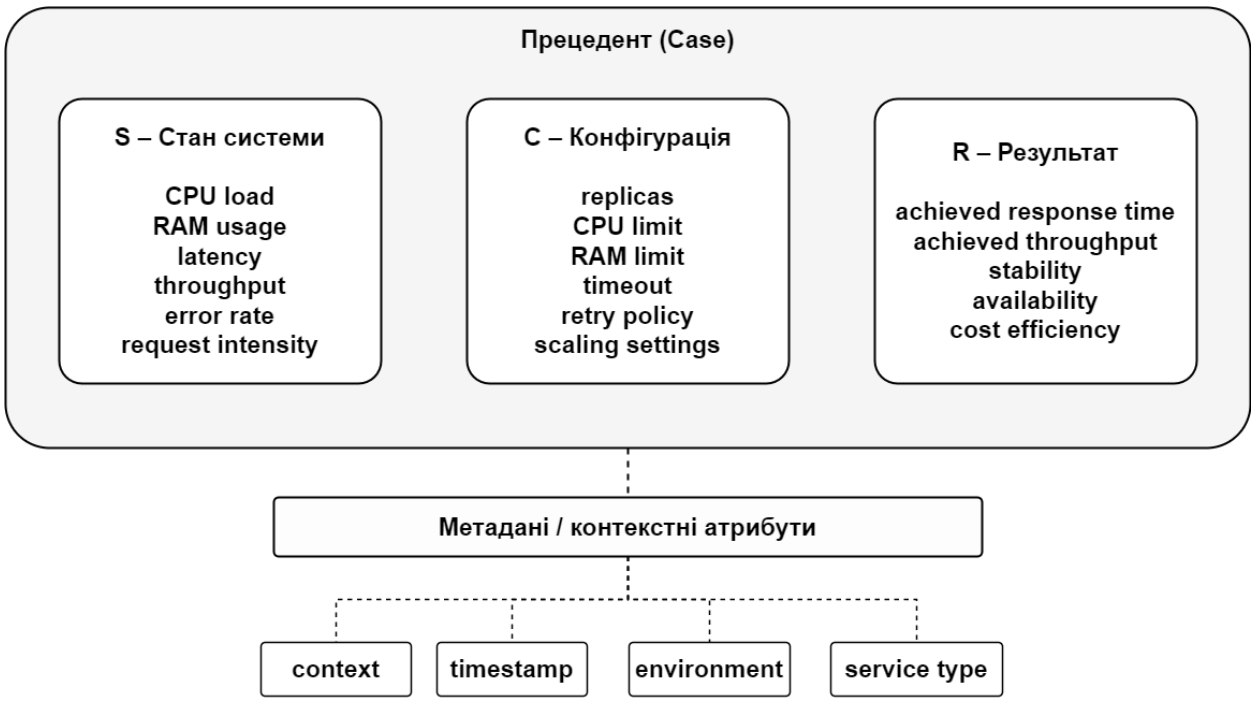


Рисунок 3.2 – Структура прецеденту в задачі АУКМ

Запропонована структура прецеденту забезпечує формалізоване представлення знань про функціонування системи та створює основу для реалізації алгоритмів пошуку та прийняття рішень [10–13].

Алгоритм адаптивного управління конфігураціями на основі CBR реалізує циклічний процес використання накопиченого досвіду для прийняття рішень. У загальному вигляді він включає чотири основні етапи: пошук (Retrieve), повторне використання (Reuse), адаптація (Revise) та збереження (Retain) [1, 2, 10, 69]. У контексті задачі управління конфігураціями ці етапи інтерпретуються наступним чином:

Retrieve – пошук у базі прецедентів випадків, подібних до поточного стану;

Reuse – вибір відповідної конфігурації;

Revise – адаптація конфігурації до поточних умов;

Retain – збереження нового досвіду у базі.

Алгоритм функціонує у вигляді циклу зворотного зв’язку, що забезпечує поступове накопичення знань та підвищення якості прийняття рішень.

На рисунку 3.3 надана концептуальна схема алгоритму АУКМ системи на основі циклу CBR. На схемі відображено послідовність основних кроків, починаючи з отримання актуального опису стану системи та формування запиту до бази прецедентів. Далі показано фазу пошуку подібних випадків, вибір відповідного конфігураційного рішення, його адаптацію до поточних умов та застосування до мікросервісного застосунку.

Окремий логічний вузол схеми відведений для оцінці результату, що дає змогу перевірити, чи забезпечує обране рішення досягнення заданих вимог до функціонування системи. Якщо результат виявляється прийнятним, новий досвід фіксується в базі прецедентів; у протилежному випадку виконується повторний цикл уточнення або пошуку іншого рішення.

Таким чином, рисунок ілюструє не лише послідовність операцій, а й замкнений характер алгоритму, орієнтованого на поступове накопичення знань і підвищення якості управлінських рішень.

Розроблений алгоритм забезпечує адаптивне управління конфігураціями шляхом використання накопиченого досвіду та реалізації циклу CBR [13, 69]. Важливо відзначити, що ключовим етапом реалізації CBR-підходу в задачі адаптивного управління конфігураціями є процедура пошуку релевантних прецедентів (Retrieve), від якості якої безпосередньо залежить ефективність подальших етапів прийняття рішень. Як було показано в Розділі 2, існує декілька підходів до реалізації цього етапу, зокрема методи найближчого сусіда, зважених k-найближчих сусідів, пошуку за ознаками, кластеризації та індексації, які відрізняються за точністю, обчислювальною складністю та масштабованістю [29, 33, 68, 87].

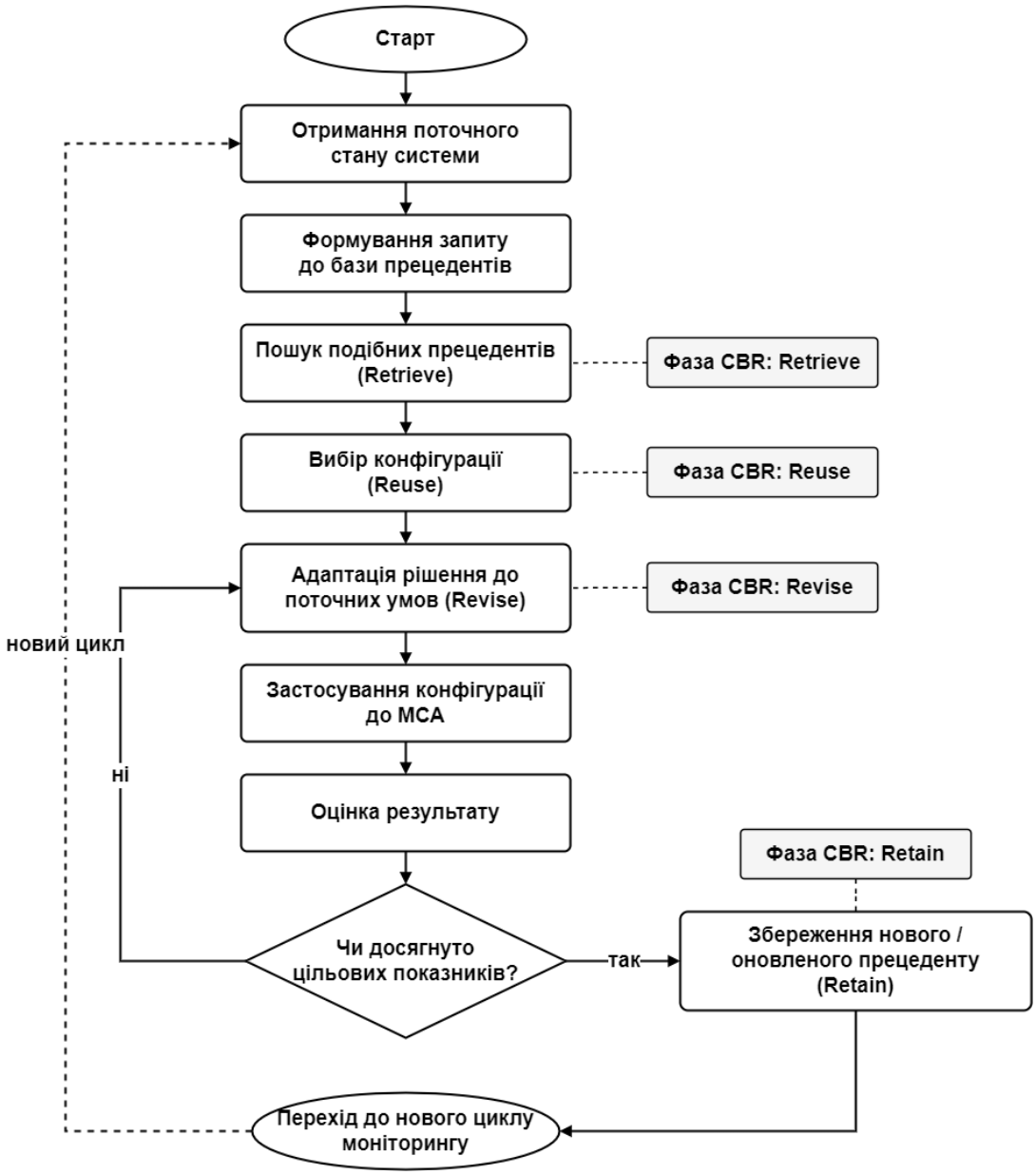


Рисунок 3.3 – Алгоритм адаптивного управління конфігураціями МСА

У межах розроблюваної алгоритмічної моделі ці методи розглядаються як альтернативні або комбіновані стратегії реалізації модуля пошуку подібних прецедентів. Вибір конкретного підходу визначається характеристиками інформаційного базису, розміром бази прецедентів, а також вимогами до швидкодії системи. Таким чином, алгоритм адаптивного управління не обмежується фіксованим методом пошуку, а передбачає можливість використання різних стратегій Retrieve, що забезпечує його гнучкість і адаптивність до умов функціонування мікросервісної архітектури.

Процес пошуку подібних прецедентів є ключовим етапом CBR, від якого залежить якість прийнятого рішення. У задачі управління конфігураціями прецеденти можуть бути представлені як точки у багатовимірному просторі параметрів [11, 12, 69, 93].

Кожен вимір відповідає окремій метриці або параметру середовища: CPU; RAM; latency; throughput; load. У таблиці 3.2 наведено приклад параметрів такого простору.

Таблиця 3.2 – Параметри багатовимірного простору прецедентів

Параметр	**Тип**	**Опис**
CPU usage	числовий	Завантаження процесора
RAM usage	числовий	Використання пам’яті
Latency	числовий	Затримка
Throughput	числовий	Пропускна здатність
Load	числовий	Навантаження

Пошук рішення полягає у визначенні найближчих прецедентів до поточного стану системи у цьому просторі. Це дозволяє знайти конфігурацію, яка раніше демонструвала ефективні результати за подібних умов [29, 33, 68].

На рисунку 3.4 показано геометричну інтерпретацію бази прецедентів у вигляді багатовимірного простору ознак, у якому кожен прецедент представлений окремою точкою. Для наочності простір подано у спрощеному вигляді через декілька координатних осей, що відповідають найбільш суттєвим параметрам стану мікросервісної системи, таким як завантаження процесора, використання оперативної пам’яті та затримка обробки запитів. Окремо виділено точку, що відповідає поточному стану системи, для якої необхідно визначити доцільне конфігураційне рішення. На схемі також позначено найближчі до неї прецеденти, які утворюють множину кандидатів для подальшого аналізу та вибору. Такий спосіб подання дозволяє наочно пояснити ідею близькості ситуацій і продемонструвати, що пошук рішення виконується не серед усіх можливих станів, а серед тих, які мають найбільшу

подібність до поточної ситуації. Крім того, просторове представлення прецедентів дає підстави для подальшого застосування методів найближчих сусідів, кластеризації та індексованого пошуку [29, 33, 68, 87].

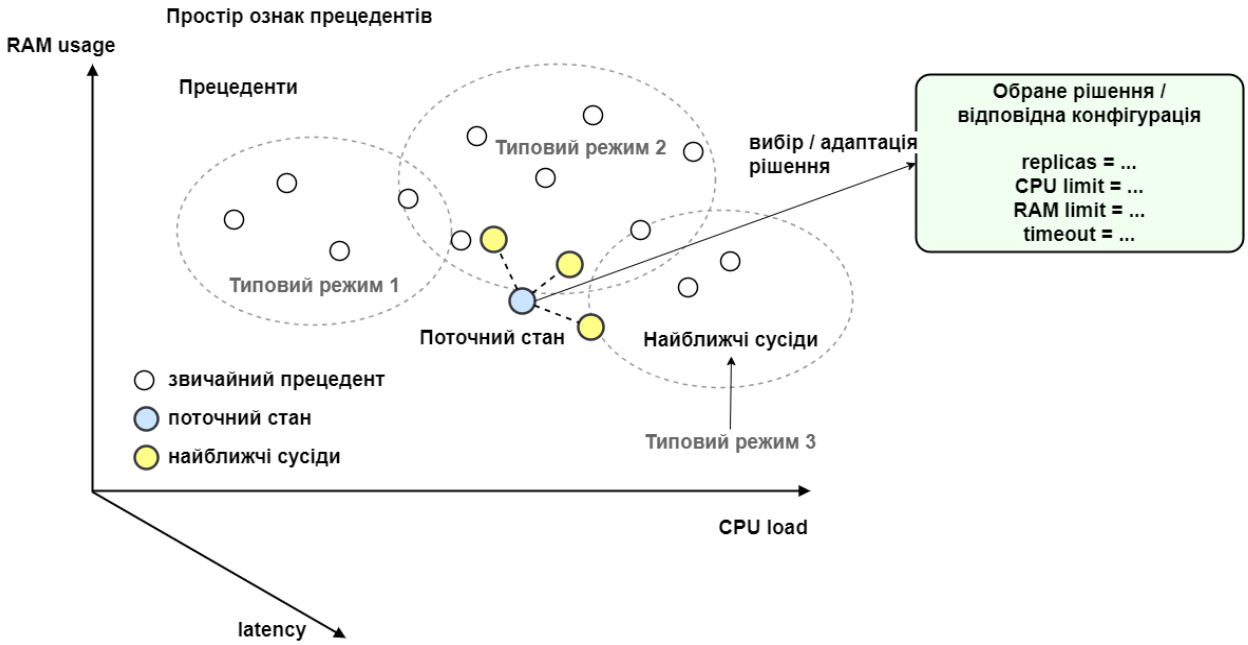


Рисунок 3.4 – Багатовимірний простір прецедентів та пошук найближчого рішення

Вибір оптимальної конфігурації здійснюється з урахуванням множини критеріїв, що характеризують ефективність функціонування системи [43, 76].

До основних критеріїв належать: продуктивність (response time); ефективність використання ресурсів; стабільність; доступність. У таблиці 3.3 наведено узагальнення критеріїв.

Таблиця 3.3 – Критерії оцінювання конфігурацій

Критерій	**Опис**	**Вплив**
Response time	Час відгуку	Продуктивність
CPU usage	Використання CPU	Вартість
Memory usage	Використання RAM	Ефективність
Availability	Доступність	Надійність

Адаптація конфігурації передбачає модифікацію знайденого рішення з урахуванням поточного стану системи. На рисунку 3.5 видно схему багатокритеріального вибору конфігураційного рішення, яка відображає логіку оцінювання кількох допустимих варіантів за системою узгоджених критеріїв. На вході схеми представлено множину кандидатних конфігурацій, сформованих на основі пошуку подібних прецедентів.

Кожна з цих конфігурацій аналізується за сукупністю показників, що характеризують різні аспекти функціонування мікросервісної системи: швидкодію, ефективність використання ресурсів, доступність, стабільність та економічну доцільність. Окремий блок схеми відведено ваговим коефіцієнтам або пріоритетам, які дають змогу враховувати значущість окремих критеріїв залежно від поточних цілей управління. Після цього формується інтегральна оцінка кожного варіанта, на підставі якої визначається найдоцільніше конфігураційне рішення. У цілому рисунок ілюструє, що в межах запропонованої моделі вибір конфігурації є не однофакторною процедурою, а результатом узгодження кількох взаємопов’язаних вимог до функціонування системи.

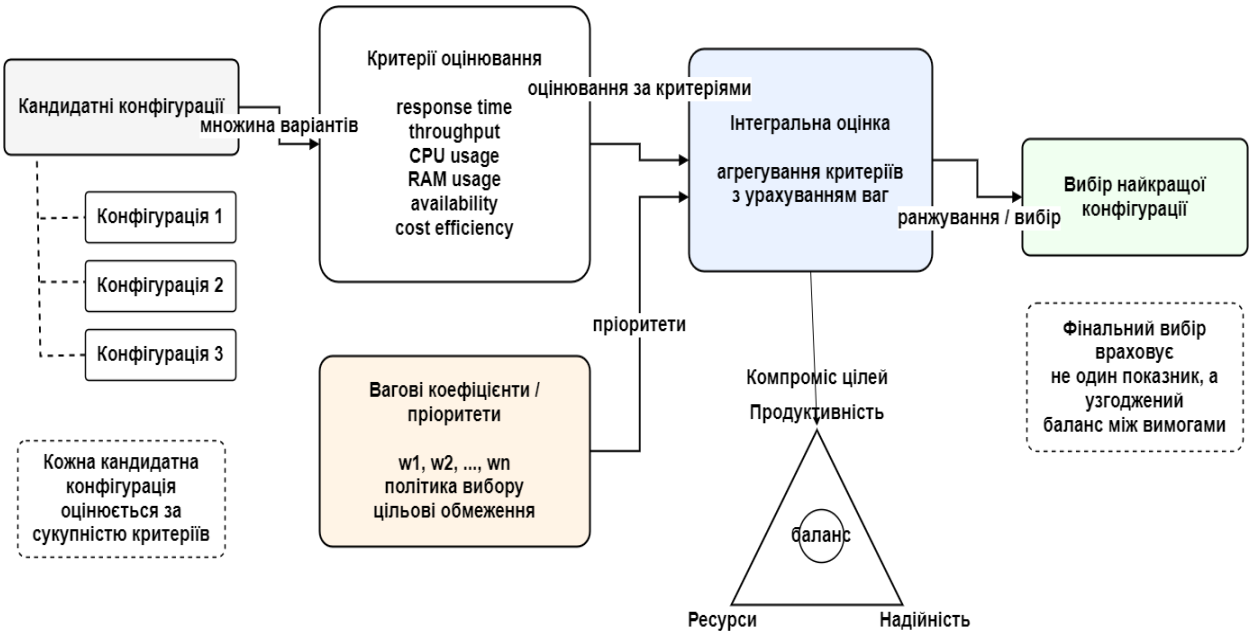


Рисунок 3.5 – Схема багатокритеріального вибору конфігурації

На основі проведених досліджень сформовано узагальнену алгоритмічну модель адаптивного управління конфігураціями, яка інтегрує: формалізацію задачі; структуру прецедентів; алгоритм CBR; механізм пошуку та вибору рішень [10–13, 70, 75]. Модель описує повний цикл функціонування системи управління конфігураціями та забезпечує її адаптивність у динамічному середовищі.

На рисунку 3.6 показано узагальнену алгоритмічну модель адаптивного управління конфігураціями мікросервісної архітектури, яка інтегрує основні функціональні підсистеми запропонованого підходу в єдиний контур керування. У структурі моделі можна виокремити інформаційний рівень, де виконується моніторинг стану системи, формування вектора ознак, звернення до бази прецедентів і реалізація механізму CBR-висновку, а також операційний рівень, де здійснюється оцінювання обраного рішення, застосування конфігураційних змін та спостереження за наслідками їх використання.

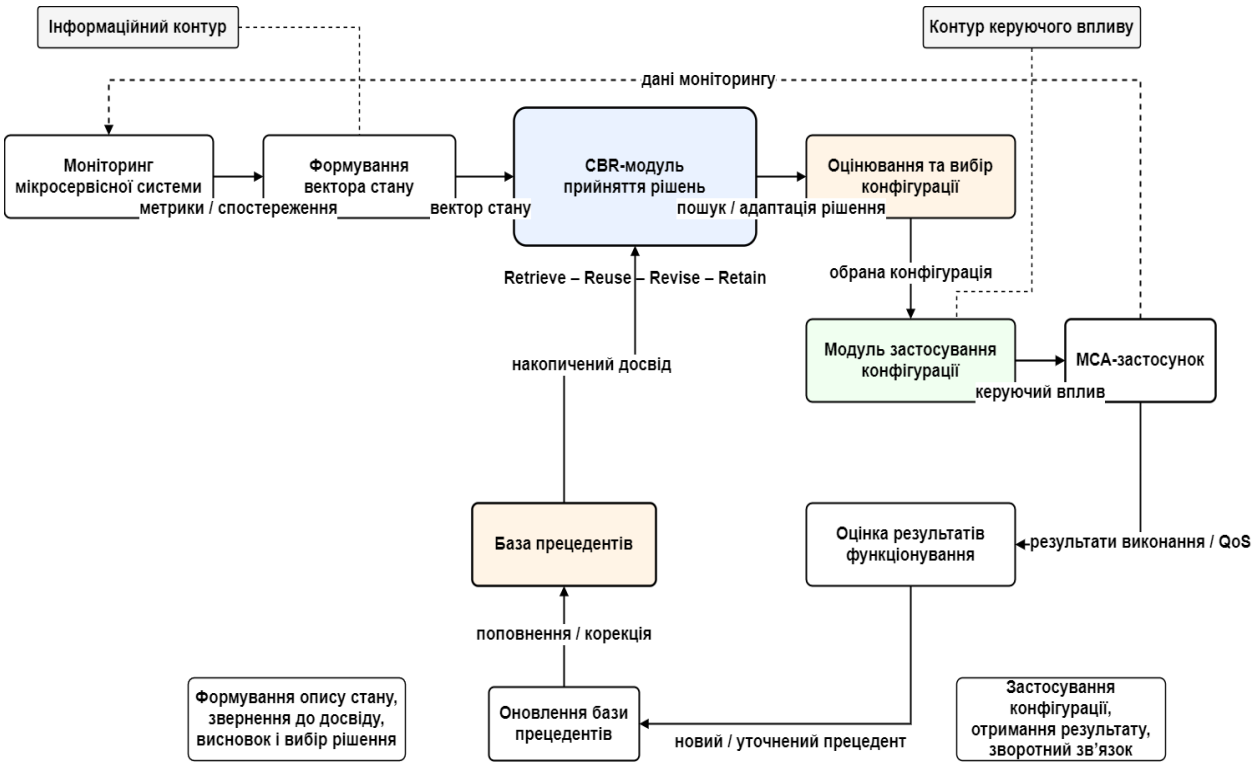


Рисунок 3.6 – Узагальнена алгоритмічна модель АУКМ

Окремо в моделі відображено процес оновлення бази прецедентів, завдяки якому результати функціонування системи перетворюються на новий досвід і можуть бути використані в наступних циклах адаптації. Така побудова підкреслює, що розроблена модель не є статичною схемою вибору параметрів, а являє собою самонавчальну систему підтримки прийняття рішень, здатну накопичувати знання про ефективні конфігурації в умовах змінного середовища функціонування [10–13, 93].

3.2 Побудова концептуальної моделі багатовимірного інформаційного базису для застосування алгоритмічної моделі

У підрозділі 3.1 було розроблено алгоритмічну модель адаптивного управління конфігураціями мікросервісної архітектури, яка базується на використанні методу аналізу прецедентів. Ефективність функціонування такої моделі безпосередньо залежить від способу організації та представлення знань, що визначає необхідність побудови відповідного інформаційного базису.

Важливою особливістю інформаційного базису є його здатність відображати динамічний характер функціонування МСА, оскільки значення параметрів системи змінюються у часі під впливом навантаження та зовнішніх факторів [34, 36, 37, 90]. З позицій теорії інтелектуальних систем, інформаційний базис повинен задовольняти такі вимоги: повнота представлення знань; узгодженість даних; можливість масштабування; ефективність пошуку та доступу до інформації [29, 33, 68]. Таким чином, побудова інформаційного базису є ключовим етапом реалізації алгоритмічної моделі, що забезпечує її практичну застосовність [10–13].

Для забезпечення можливості використання алгоритмів аналізу прецедентів необхідно представити систему у вигляді багатовимірного простору ознак, у якому кожен стан описується набором параметрів [29, 33, 68] (див. 2.1).

До складу вектора стану доцільно включити такі групи параметрів: ресурсні (CPU, RAM, disk usage); продуктивнісні (latency, throughput); навантаження (кількість запитів, інтенсивність); мережеві характеристики [59–61, 88]. У таблиці 3.4 наведено узагальнену структуру вектора стану системи.

Таблиця 3.4 – Структура багатовимірного вектора стану системи

Група параметрів	**Параметри**	**Опис**
Ресурсні	CPU, RAM	Використання ресурсів
Продуктивність	latency, throughput	Швидкодія
Навантаження	request rate	Інтенсивність
Мережа	delay, packet loss	Якість мережі

Для наочного подання процесу формування багатовимірного інформаційного простору (БІП) на рисунку 3.7 показано узагальнену схему інтеграції параметрів стану системи та конфігураційних характеристик.

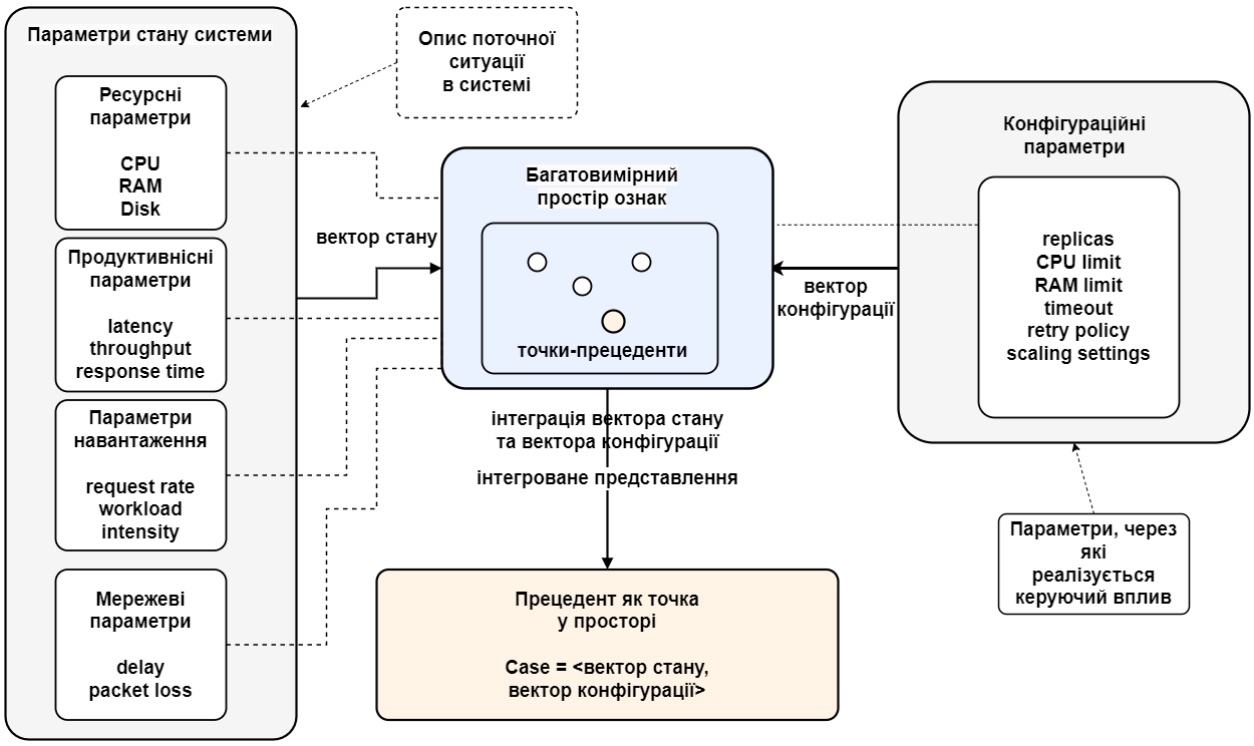


Рисунок 3.7 – Схема формування БІП станів і конфігурацій МСА

Як видно зі схеми, формування простору ознак здійснюється шляхом об’єднання різнорідних груп параметрів, що описують як поточний стан мікросервісної системи, так і можливі варіанти її конфігурації. При цьому кожен прецедент може бути інтерпретований як окрема точка у сформованому просторі, що визначається відповідною комбінацією значень параметрів. Такий підхід забезпечує уніфіковане представлення даних і створює основу для подальшого застосування методів пошуку та аналізу подібності.

Такий підхід дозволяє розглядати задачу управління конфігураціями як задачу пошуку у БІП, що є характерним для інтелектуальних систем прийняття рішень [43, 46–48]. Представлення станів і конфігурацій у вигляді багатовимірних векторів створює основу для застосування методів пошуку та аналізу прецедентів. На основі сформованих векторних представлень будується концептуальна модель інформаційного простору, що об’єднує всі компоненти інформаційного базису.

Концептуальне представлення БІП наведено на рисунку 3.8**,** де показано взаємозв’язок між простором станів системи, простором конфігурацій та простором результатів функціонування. Як видно зі схеми, кожен прецедент формується як відображення між цими просторами, що дозволяє інтерпретувати накопичений досвід у вигляді структурованих знань [10–13, 69]. Включення бази прецедентів у цю модель підкреслює її роль як інтегруючого елемента, у якому зберігаються та узагальнюються результати функціонування системи. Така концептуалізація забезпечує цілісність інформаційного представлення та підтримує процеси прийняття рішень на основі попереднього досвіду [43, 46–48].

У цій моделі кожен прецедент інтерпретується як відображення $S \rightarrow C \rightarrow R$, що дозволяє формалізувати процес прийняття рішень як відображення у просторі знань. Особливістю моделі є те, що вона забезпечує інтеграцію структурованих даних (метрики), конфігураційних параметрів та історичного досвіду. Це створює передумови для використання ефективних механізмів пошуку, класифікації та узагальнення знань, а також забезпечує

цілісне представлення інформаційного простору задачі та визначає структуру взаємозв’язків між її елементами [29, 33, 68, 87].

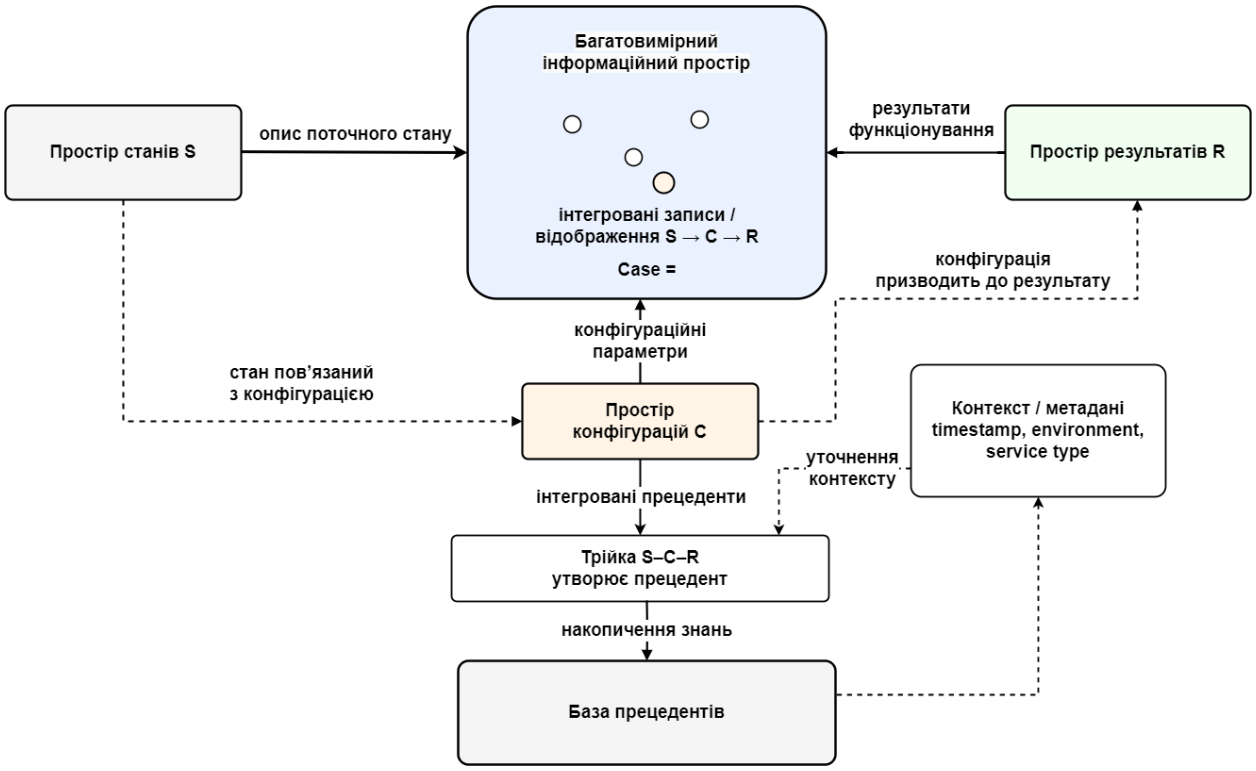


Рисунок 3.8 - Концептуальна схема побудови БІП

Інформаційний базис реалізується у вигляді бази прецедентів, яка містить структуровані записи досвіду функціонування системи. Основними компонентами бази є: множина прецедентів; механізми індексації; засоби доступу до даних. У таблиці 3.5 наведено структуру інформаційного базису.

Таблиця 3.5 – Структура інформаційного базису прецедентів

Компонент	**Опис**
Прецеденти	Записи виду <S, C, R>
Індекси	Засоби прискорення пошуку
Метадані	Контекстні параметри
Зв’язки	Відношення між прецедентами

Організацію інформаційного базису прецедентів доцільно розглядати відповідно до схеми, наведеної на рисунку 3.9**,** де показано багаторівневу структуру зберігання та обробки даних. Зокрема, схема демонструє послідовність перетворення первинних даних моніторингу у структуровані прецеденти, а також їх доповнення метаданими та індексами [59–61, 88]. Як видно з рисунка, наявність механізмів індексації забезпечує ефективний доступ до релевантних записів, що є критично важливим у випадку великої кількості прецедентів [29, 33, 68]. Крім того, відображені зв’язки між елементами базису дозволяють враховувати контекст функціонування системи, що підвищує точність пошуку та прийняття рішень. Особливу роль відіграють механізми індексації, що дозволяють зменшити час пошуку релевантних прецедентів у багатовимірному просторі [87].

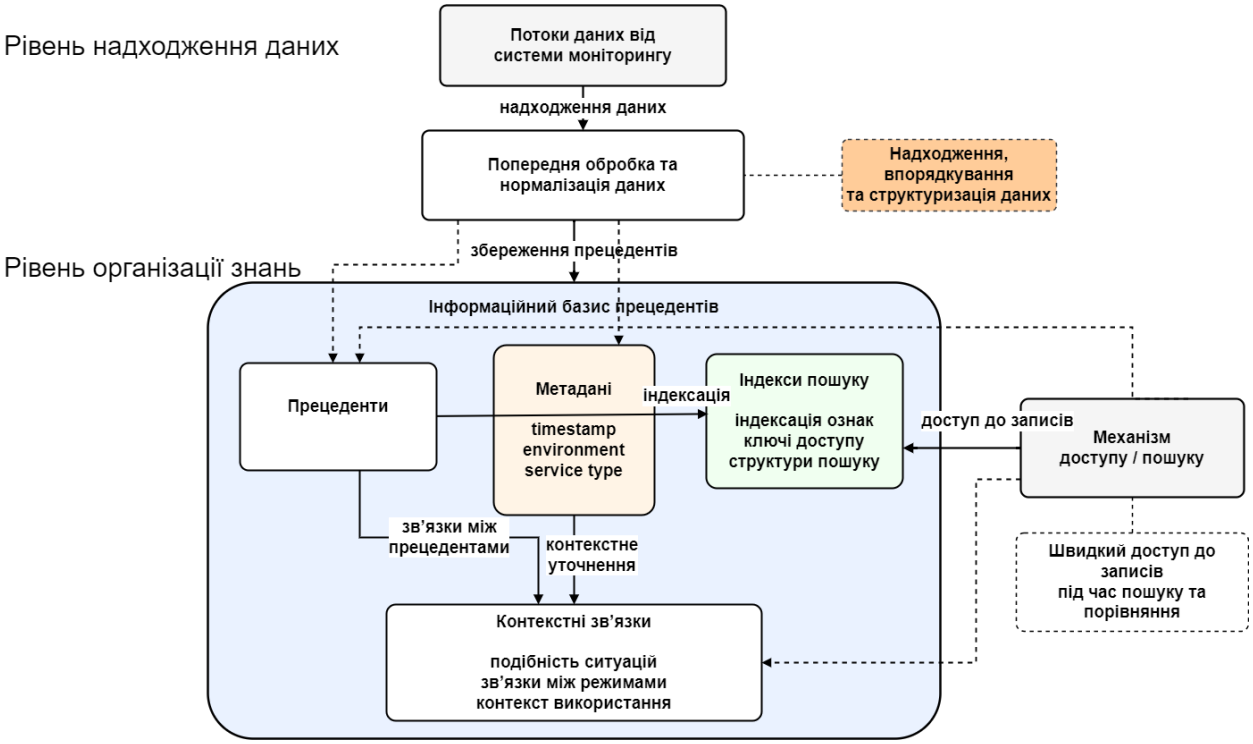


Рисунок 3.9 – Узагальнена схема організації інформаційного базису

Побудований багатовимірний інформаційний базис забезпечує формалізоване представлення знань, необхідних для реалізації алгоритмічної моделі адаптивного управління конфігураціями. Розроблена концептуальна

модель інформаційного базису формує основу для реалізації інтелектуального управління конфігураціями та забезпечує ефективну підтримку алгоритму CBR.

3.3 Проектування компонентної архітектури інструментальних засобів для адаптивного управління конфігураціями

У попередніх підрозділах було розроблено алгоритмічну модель адаптивного управління конфігураціями та сформовано концептуальний багатовимірний інформаційний базис. Наступним етапом є проектування інструментальних засобів, що забезпечують практичну реалізацію запропонованого підходу [10–13, 70].

Архітектура інструментальних засобів повинна враховувати специфіку, яка характеризується: децентралізованістю компонентів; високою динамічністю станів; необхідністю масштабування; асинхронною взаємодією сервісів [34, 36, 37, 87].

З урахуванням цього до архітектури системи висуваються такі основні вимоги:

- модульність, що забезпечує незалежність компонентів;
- масштабованість, необхідна для обробки великого обсягу даних;
- відмовостійкість, яка гарантує стабільність функціонування;
- адаптивність, що забезпечує підтримку динамічних змін;
- інтегрованість, яка дозволяє взаємодіяти з існуючими системами моніторингу [59–61, 88].

Для узагальнення вимог до побудови системи на рисунку 3.10 показано концептуальну модель архітектури інструментальних засобів адаптивного управління конфігураціями. Як видно зі схеми, архітектура розглядається як система, що формується під впливом ключових нефункціональних характеристик, серед яких є модульність, масштабованість, відмовостійкість, адаптивність та інтегрованість. Центральний компонент відображає

інтелектуальне ядро системи, яке функціонує у взаємодії з мікросервісним середовищем та інфраструктурними сервісами. Така інтерпретація дозволяє розглядати архітектуру не лише як технічну структуру, а як сукупність принципів організації системи, що визначають її поведінку в умовах змінного середовища [70, 75, 76].

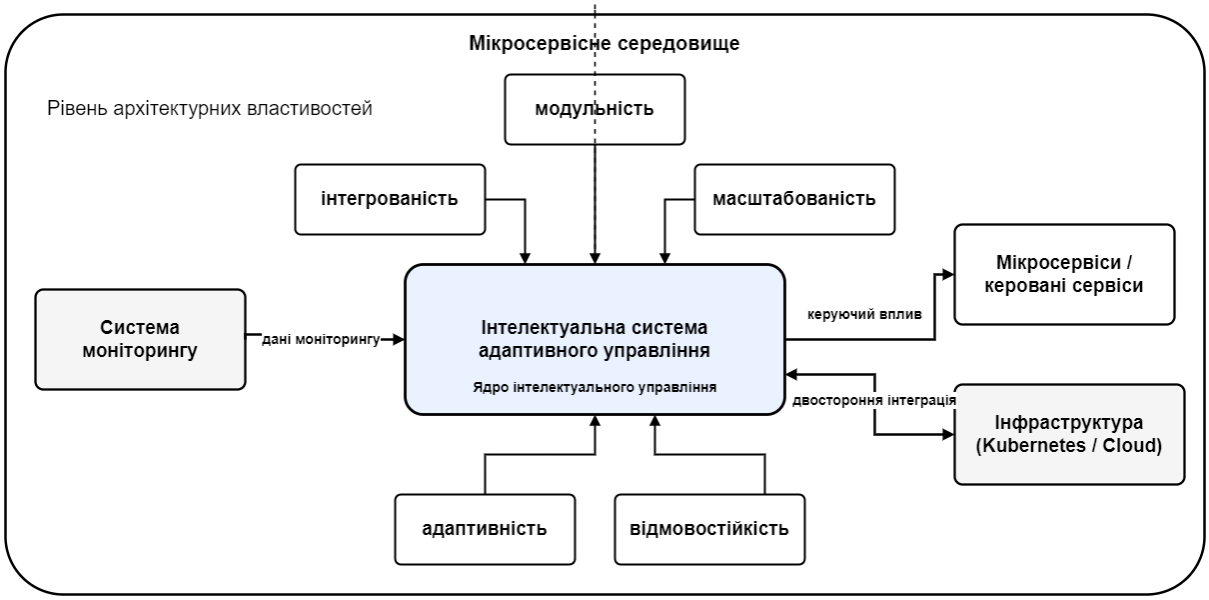


Рисунок 3.10 – Концептуальна модель архітектури інструментальних засобів адаптивного управління конфігураціями

На основі визначених вимог була сформована загальна структурна схема системи, яка реалізує алгоритмічну модель, описану у підрозділі 3.1. Архітектура системи включає такі основні компоненти:

- модуль збору даних (моніторинг) збору метрик;
- модуль обробки та нормалізації даних призначений для підготовки даних;
- інформаційний базис прецедентів призначений для зберігання знань;
- модуль CBR призначений для пошуку рішень;
- модуль прийняття рішень призначений для формування конфігурацій;
- модуль управління конфігураціями призначений для застосування конфігурацій;

– інтерфейс взаємодії з користувачем або API призначений для інтеграції.

Для деталізації складу системи та взаємозв’язків між її компонентами на рисунку 3.11 показано структурну схему компонентної архітектури. Як видно зі схеми, система організована у вигляді послідовності функціональних блоків, що забезпечують обробку даних від етапу збору метрик до формування керуючих впливів. Центральне місце займає модуль аналізу прецедентів, який взаємодіє з інформаційним базисом та забезпечує вибір релевантних конфігурацій. При цьому двосторонній зв’язок з базою прецедентів дозволяє не лише використовувати накопичений досвід, але й поповнювати його новими даними. Така структура відображає логіку перетворення вхідної інформації у керуючі рішення [10–13, 93].

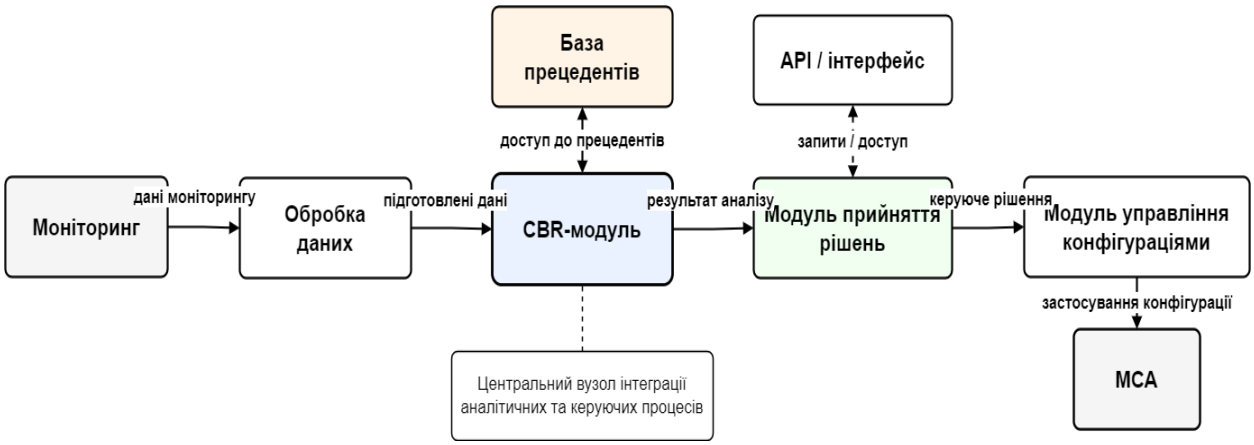


Рисунок 3.11 – Структурна схема компонентної архітектури системи

Для забезпечення гнучкості та масштабованості система декомпонується на функціональні модулі. Основні функціональні модулі:

– Data Collector – збір метрик;
– Preprocessing Module – обробка даних;
– Case Storage Module – управління базою прецедентів;
– Similarity Engine – пошук подібних прецедентів;
– Adaptation Engine – адаптація рішень;
– Execution Module – застосування конфігурацій.

У таблиці 3.10 наведено деталізацію функціональних модулів.

Таблиця 3.10 – Функціональні модулі системи

Модуль	**Функції**
Data Collector	Збір даних
Preprocessing	Фільтрація
Case Storage	Зберігання
Similarity Engine	Пошук
Adaptation Engine	Адаптація
Execution	Виконання

Декомпозицію системи на функціональні модулі наведено на рисунку 3.12, де кожен модуль відповідає окремому етапу обробки інформації в межах загального процесу адаптивного управління. Як видно зі схеми, система реалізує послідовний ланцюг перетворення даних, починаючи зі збору та попередньої обробки, і завершуючи застосуванням конфігураційних рішень. Особливістю представленої структури є наявність тісного взаємозв’язку між модулем пошуку подібності та сховищем прецедентів, що забезпечує ефективне використання накопиченого досвіду. Така модульна організація створює передумови для масштабування системи та незалежного розвитку її окремих компонентів.

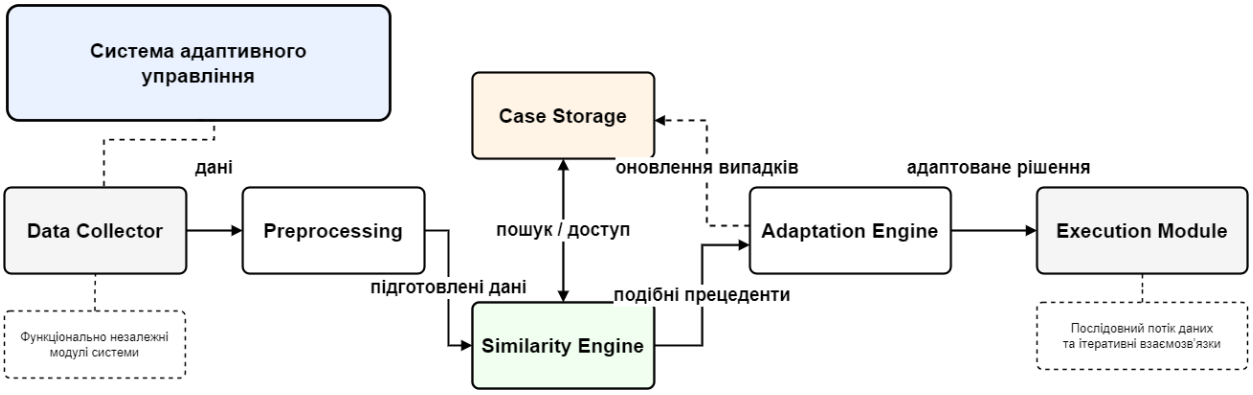


Рисунок 3.12 – Декомпозиція системи на функціональні модулі

З урахуванням специфіки мікросервісної архітектури важливим є визначення моделі взаємодії між компонентами системи. Взаємодія може

здійснюватися синхронно (REST, gRPC) та асинхронно (через брокери повідомлень) [34, 36, 37, 87].

Особливості взаємодії компонентів у мікросервісному середовищі відображено на рисунку 3.13, де показано поєднання синхронних та асинхронних механізмів обміну даними. Як видно зі схеми, синхронна взаємодія реалізується через стандартні протоколи, що забезпечують швидкий обмін запитами, тоді як асинхронна взаємодія базується на використанні брокера повідомлень і дозволяє зменшити зв'язаність компонентів. Такий підхід забезпечує гнучкість системи та підвищує її масштабованість у умовах змінного навантаження. Наявність API-шару додатково спрощує інтеграцію компонентів та уніфікує доступ до сервісів [23, 24, 34, 36].

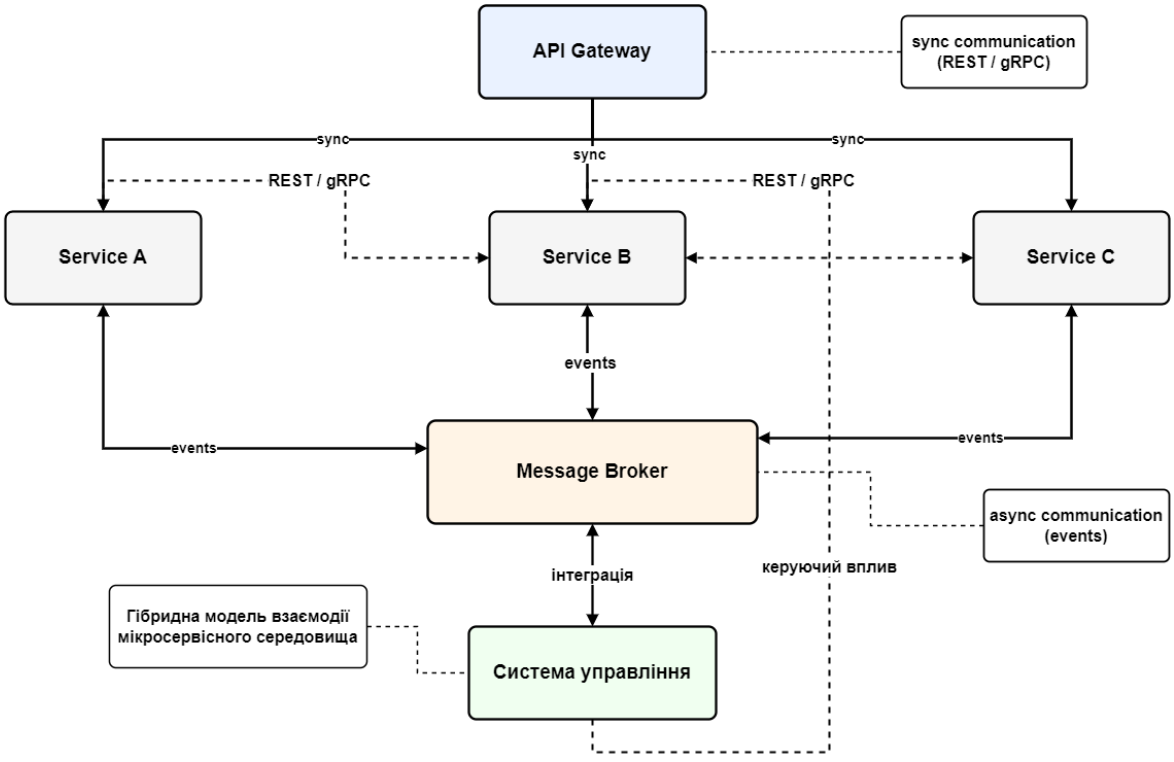


Рисунок 3.13 – Модель взаємодії компонентів у мікросервісному середовищі

На основі вище сказаного була розроблена узагальнена архітектура інструментального засобу адаптивного управління конфігураціями, яка представлена на рисунку 3.14 та показує повний цикл функціонування

системи. Як видно зі схеми, система реалізує замкнений контур управління, що включає етапи збору даних, їх аналізу, прийняття рішень та застосування конфігурацій. Важливим елементом є механізм зворотного зв’язку, який забезпечує накопичення нового досвіду у базі прецедентів та підвищення ефективності системи з часом. Така організація дозволяє розглядати систему як самонавчальну, що адаптується до змін умов функціонування мікросервісної архітектури [70, 75, 76].

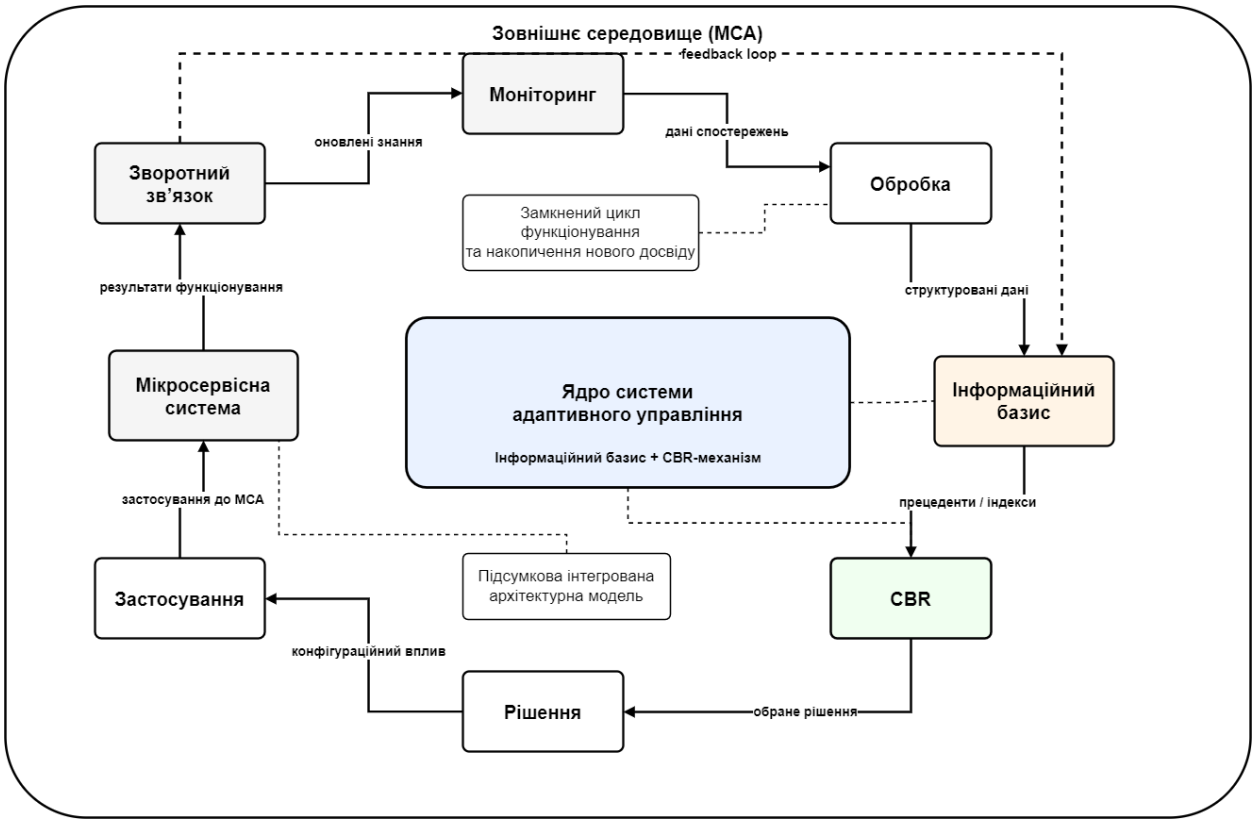


Рисунок 3.14 – Узагальнена архітектура інструментальних засобів адаптивного управління

Особливістю архітектури є інтеграція інформаційного базису з алгоритмічним ядром системи, що забезпечує адаптивність та самонавчання. Запропонована архітектура забезпечує реалізацію алгоритмічної моделі адаптивного управління та створює основу для подальшої програмної реалізації.

3.4 Висновки до Розділу 3

У третьому розділі дисертаційної роботи вирішено науково-практичну задачу розробки моделей та архітектурних рішень для реалізації інтелектуального адаптивного управління конфігураціями програмних мікросервісів на основі методу аналізу прецедентів.

У результаті проведених досліджень отримано такі основні результати:

1) Формалізовано задачу АУКМ як задачу відображення поточного стану системи у простір конфігурацій з урахуванням накопиченого досвіду функціонування. На відміну від існуючих підходів, запропонована постановка враховує багатовимірний характер параметрів системи та дозволяє інтегрувати історичні дані у процес прийняття рішень.

2) Розроблено алгоритмічну модель АУКМ на основі методу аналізу прецедентів (CBR), яка включає етапи пошуку, повторного використання, адаптації та накопичення рішень. Особливістю запропонованої моделі є її орієнтація на роботу в умовах динамічного навантаження та можливість самонавчання за рахунок поповнення бази прецедентів.

3) Запропоновано модель представлення знань у вигляді структурованих прецедентів, що включають параметри стану системи, конфігураційні характеристики та результати функціонування. Це дозволяє враховувати контекст виконання та часові аспекти, що підвищує точність пошуку подібних рішень.

4) Побудовано концептуальну модель багатовимірного інформаційного базису, яка забезпечує інтеграцію параметрів стану, конфігурацій та результатів у єдиний інформаційний простір.

5) Розроблено структуру інформаційного базису прецедентів, що включає механізми індексації, зберігання та доступу до даних. Це забезпечує підвищення ефективності пошуку релевантних прецедентів у багатовимірному просторі ознак та скорочення часу прийняття рішень.

6) Сформульовано вимоги до архітектури інструментальних засобів адаптивного управління, які враховують специфіку мікросервісних систем,

зокрема їх розподіленість, динамічність та необхідність масштабування. Обґрунтовано доцільність використання модульної та слабкозв’язаної архітектури.

7) Запропоновано компонентну архітектуру інструментальних засобів, що включає модулі збору даних, обробки інформації, аналізу прецедентів, прийняття рішень та управління конфігураціями. Архітектура забезпечує реалізацію повного циклу адаптивного управління.

8) Виконано декомпозицію системи на функціональні модулі, що дозволяє забезпечити гнучкість, масштабованість та можливість незалежного розвитку компонентів. Виділено ключові модулі, зокрема модулі пошуку подібності та адаптації рішень.

9) Розроблено модель взаємодії компонентів у мікросервісному середовищі, яка поєднує синхронні та асинхронні механізми обміну даними. Показано, що використання брокерів повідомлень дозволяє зменшити зв’язаність компонентів та підвищити стійкість системи.

10) Сформовано узагальнену архітектуру інструментальних засобів адаптивного управління, яка реалізує замкнений цикл обробки даних із використанням механізму зворотного зв’язку. Це забезпечує можливість накопичення знань та підвищення ефективності системи у процесі її експлуатації.

Таким чином, було сформовано теоретичну та методичну основу для створення інтелектуальних інструментальних засобів систем АУКМ. Отримані результати є базою для подальшої програмної реалізації та експериментального дослідження ефективності запропонованого підходу, що буде розглянуто у Розділі 4.

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ ЗАПРОПОНОВАНОГО ПІДХОДУ

4.1 Опис тестового застосунку з мікросервісною архітектурою для предметної області електронної торгівлі

Однією з ключових задач даного дослідження є експериментальна перевірка ефективності запропонованого підходу до адаптивного управління конфігураціями мікросервісних застосунків в умовах динамічного навантаження. У зв’язку з цим як предметну область було обрано електронну торгівлю.

Системи електронної торгівлі є типовим прикладом сучасних розподілених програмних систем, для яких характерними є значні коливання навантаження, велика кількість одночасних користувачів та підвищені вимоги до продуктивності й доступності [80, 83, 85]. За таких умов використання статичних конфігурацій мікросервісів є недостатньо ефективним, що обумовлює необхідність застосування адаптивних підходів до управління ресурсами [10–13, 91].

Додатковою перевагою обраної предметної області є її структурованість, що дозволяє природним чином виділити основні функціональні компоненти системи у вигляді окремих мікросервісів. Це створює зручні умови для побудови тестового полігону та дослідження процесів адаптивного управління конфігураціями [70, 75].

Функціональні вимоги визначають можливості тестового полігону та системи адаптивного управління конфігураціями мікросервісів, що забезпечують їх ефективне функціонування у змодельованих умовах експлуатації.

Тестовий полігон призначений для відтворення роботи мікросервісного застосунку, що складається з трьох основних сервісів: сервісу автентифікації користувачів (auth_service), сервісу управління каталогом продуктів

(product_service) та сервісу обробки замовлень (order_service). Така структура відповідає типовій організації систем електронної торгівлі [80, 81].

Полігон забезпечує моделювання різних рівнів навантаження, а також підтримує зміну конфігураційних параметрів мікросервісів через програмні інтерфейси, що дозволяє здійснювати експериментальне дослідження їх впливу на продуктивність системи.

Система АУКМ реалізує інтелектуальний рівень управління, забезпечуючи збір метрик, аналіз стану системи та автоматизований вибір конфігурацій на основі методів Case-Based Reasoning. Вона взаємодіє з тестовим полігоном через API, що дозволяє застосовувати нові конфігурації у режимі, наближеному до реального часу. Таким чином, тестовий полігон у поєднанні із системою АУКМ створює експериментальне середовище для дослідження ефективності адаптивного управління конфігураціями мікросервісів.

Загальна структура тестового полігону представлена на рисунку 4.1.

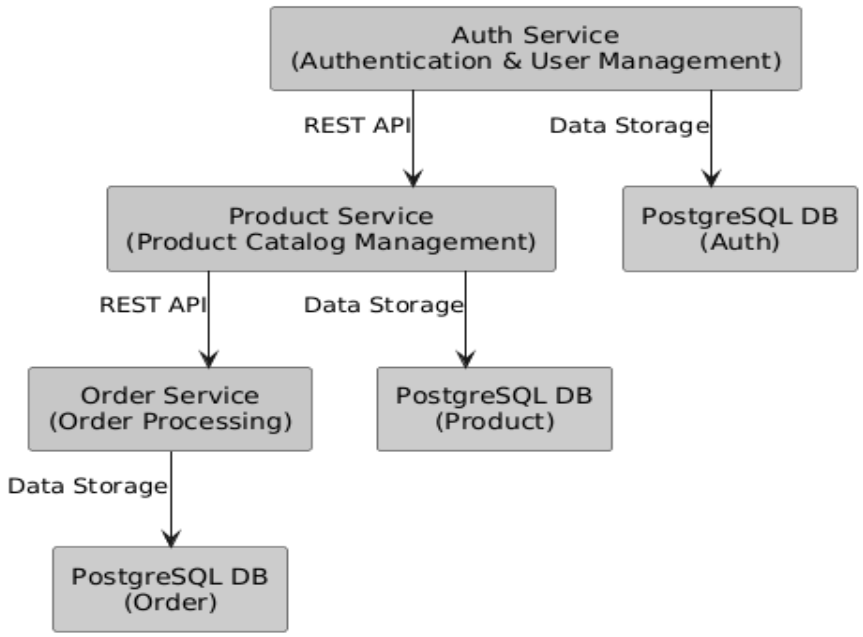


Рисунок 4.1 – Загальна архітектура тестового полігону

Архітектура полігону базується на мікросервісному підході, у межах якого кожен сервіс виконує окрему функціональну роль та функціонує як незалежний компонент системи [83, 85, 87].

Сервіс auth_service забезпечує автентифікацію та авторизацію користувачів, реалізуючи механізми перевірки облікових даних, генерації та валідації токенів доступу. Він відіграє ключову роль у забезпеченні безпеки системи.

Сервіс product_service відповідає за управління каталогом товарів і обробку запитів, пов’язаних із отриманням інформації про продукти. Його функціонування характеризується переважанням операцій читання.

Сервіс order_service реалізує логіку обробки замовлень, включаючи їх створення, обробку та взаємодію з іншими сервісами. Даний компонент має більш складну логіку та підвищені вимоги до узгодженості даних [80, 81].

Взаємодія між мікросервісами здійснюється через RESTful API, що забезпечує стандартизований механізм обміну даними та слабке зв’язування компонентів системи [23, 24, 34, 36]. Це дозволяє забезпечити незалежність розвитку та масштабування окремих сервісів.

Взаємодія користувачів із тестовим полігоном описується за допомогою діаграми прецедентів, наведеної на рисунку 4.2.

У системі передбачено три ролі користувачів: неавторизований користувач, зареєстрований користувач та адміністратор.

Неавторизований користувач має обмежений доступ і може виконувати лише перегляд інформації. Зареєстрований користувач отримує розширені можливості, включаючи створення та відстеження замовлень. Адміністратор має повний доступ до функціональності системи, включаючи управління даними та конфігураціями.

Всі операції виконуються через API-інтерфейси, що забезпечує контроль доступу та узгодженість взаємодії компонентів системи.

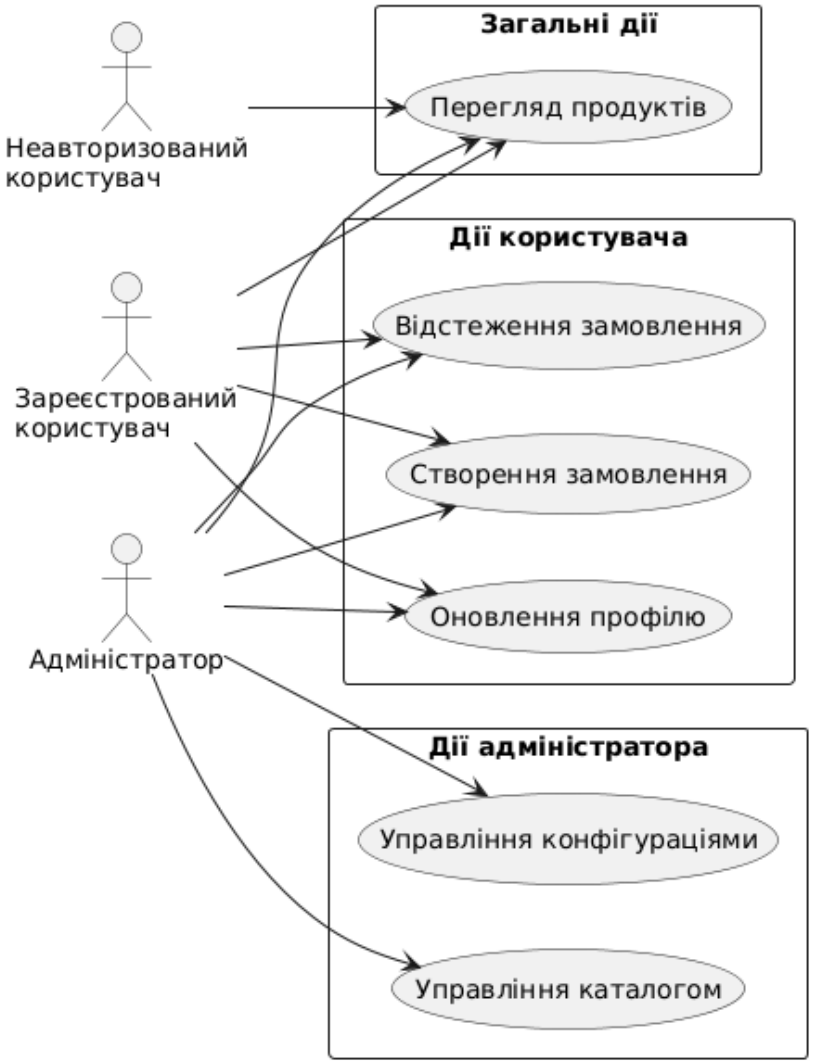


Рисунок 4.2 – Діаграма прецедентів тестового полігону

Кожен мікросервіс використовує власне сховище даних, реалізоване на базі PostgreSQL, що забезпечує ізоляцію даних та відповідає принципам мікросервісної архітектури. Для візуалізації структури даних використано узагальнену ER-модель, представлену на рисунку 4.3.

У сервісі автентифікації зберігається інформація про користувачів, включаючи їх облікові дані та ролі. Сервіс продуктів містить інформацію про товари та їх характеристики. Сервіс замовлень реалізує структуру даних, що відображає як самі замовлення, так і їх складові елементи. Такий підхід дозволяє моделювати типові процеси обробки даних у системах електронної торгівлі [80, 81].

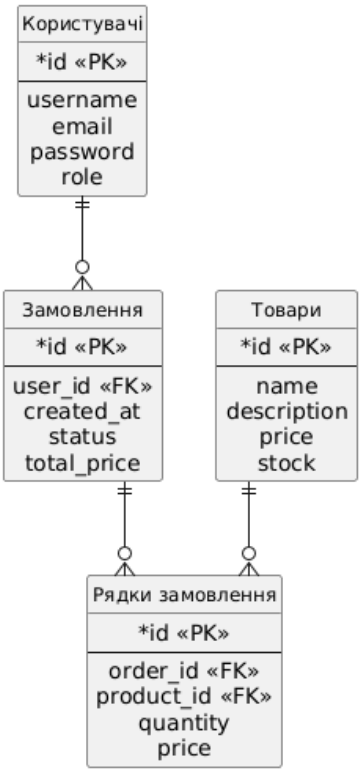


Рисунок 4.3 – ER-діаграма бази даних тестового полігону

Архітектурні зв’язки між компонентами тестового полігону представлені на рисунку 4.4.

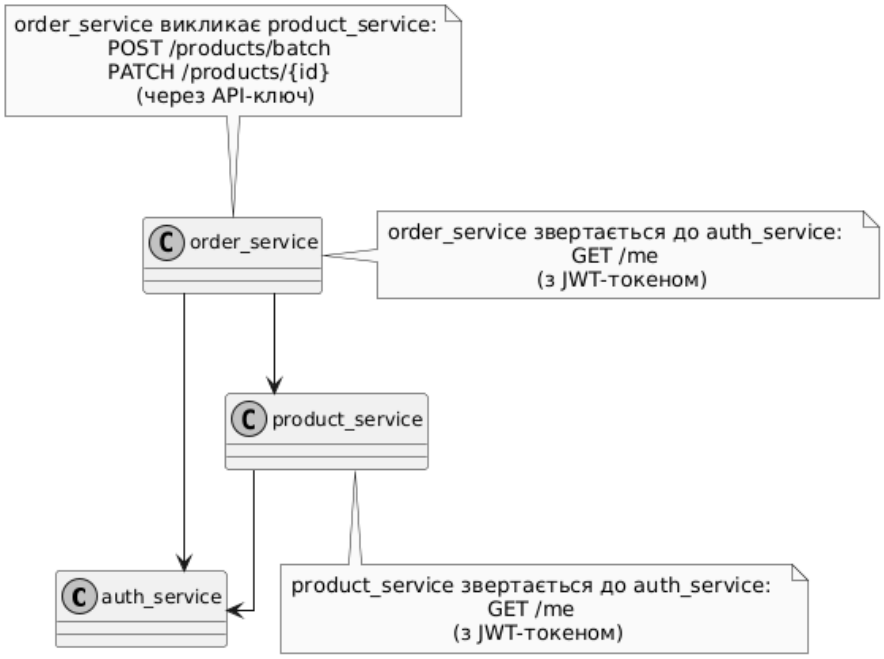


Рисунок 4.4 – Архітектурна діаграма тестового полігону із зв’язками між мікросервісами

Сервіси функціонують як автономні компоненти та можуть розгортатися у контейнеризованому середовищі [87, 90]. Взаємодія між ними реалізується напряму через REST-запити без використання централізованого шлюзу. Для забезпечення безпеки використовується механізм токен-орієнтованої автентифікації, що дозволяє контролювати доступ до ресурсів системи.

Кожен мікросервіс підтримує набір конфігураційних параметрів, які можуть змінюватися у процесі функціонування системи. До таких параметрів належать характеристики підключення до бази даних, обмеження на інтенсивність запитів, а також параметри, що визначають обсяг оброблюваних даних та затримки відповіді. Зміна цих параметрів дозволяє адаптувати поведінку системи до різних режимів навантаження, включаючи низький, середній, високий та піковий [10–13, 91]. Таким чином, тестовий полігон забезпечує необхідні умови для проведення експериментів із адаптивного управління конфігураціями та дозволяє оцінити ефективність запропонованого підходу в умовах змінного навантаження.

4.2 Особливості програмної реалізації адаптивного управління конфігураціями мікросервісів на основі запропонованого підходу

Програмна реалізація запропонованого підходу до системи АУКМ базується на розділенні системи на два функціонально взаємопов’язані рівні - тестовий полігон, що виступає об’єктом управління, та програмний застосунок адаптивного управління конфігураціями мікросервісів (АУКМ), який реалізує функції інтелектуального аналізу та прийняття рішень.

Тестовий полігон, описаний у підрозділі 4.1, забезпечує моделювання поведінки мікросервісної системи в умовах змінного навантаження та надає можливість зміни конфігураційних параметрів. У свою чергу, система АУКМ виконує функції збору метрик, аналізу стану системи та формування керуючих впливів у вигляді нових конфігурацій [59–61, 88].

Така організація відповідає концепції замкненого контуру управління, розробленій у розділі 3, та забезпечує реалізацію повного циклу адаптації: від спостереження за станом системи до застосування керуючих рішень [70, 75, 76].

Архітектура програмного застосунку АУКМ побудована відповідно до модульного принципу та включає сукупність компонентів, кожен з яких відповідає за окремий етап процесу адаптивного управління [34, 36, 37, 87]. Ключовими компонентами системи є:

- модуль збору та агрегації метрик;
- модуль представлення стану системи;
- база прецедентів;
- модуль пошуку та вибору релевантних прецедентів;
- модуль адаптації конфігурацій;
- модуль застосування конфігурацій.

Модуль збору метрик здійснює отримання даних про стан мікросервісів тестового полігону, включаючи показники використання ресурсів та продуктивності [59–61]. Отримані дані агрегуються та перетворюються у структурований опис поточного стану системи. Цей опис використовується як вхідний вектор ознак для подальшого аналізу в межах CBR-підходу. Загальна архітектуру програмного застосунку АУКМ представлена на рисунку 4.5.

У рамках реалізації CBR-підходу ключовим елементом є формування прецедентів, які відображають досвід функціонування системи в різних умовах. Кожен прецедент описується множиною параметрів, що характеризують стан системи, а також відповідною конфігурацією мікросервісів і результатами її застосування. До складу ознак прецеденту входять показники навантаження, використання ресурсів, часові характеристики та інші параметри, що визначають якість функціонування системи [43, 46–48].

Представлення прецедентів у вигляді багатовимірного вектора ознак узгоджується з концептуальною моделлю інформаційного базису,

розробленою у підрозділі 3.2, та забезпечує можливість застосування методів пошуку найближчих прецедентів (див рисунок 4.6) [29, 33, 68].

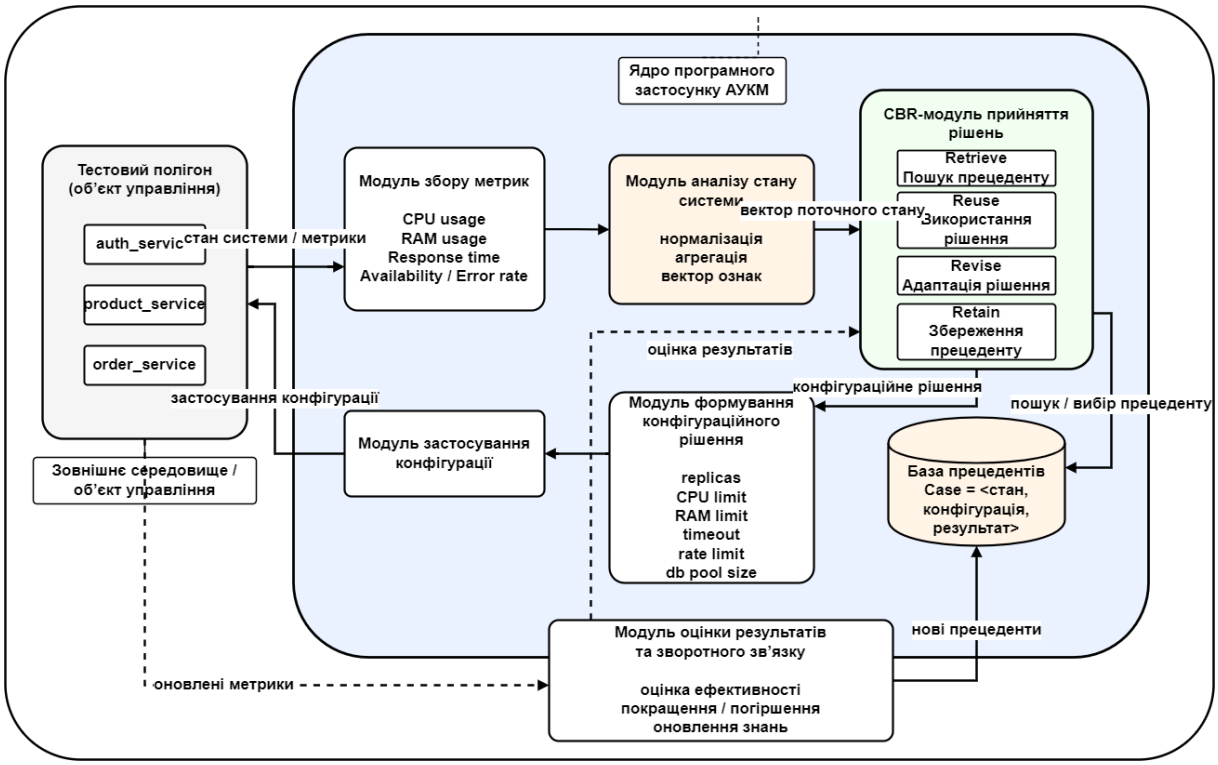


Рисунок 4.5 – Архітектура програмного застосунку адаптивного управління конфігураціями мікросервісів

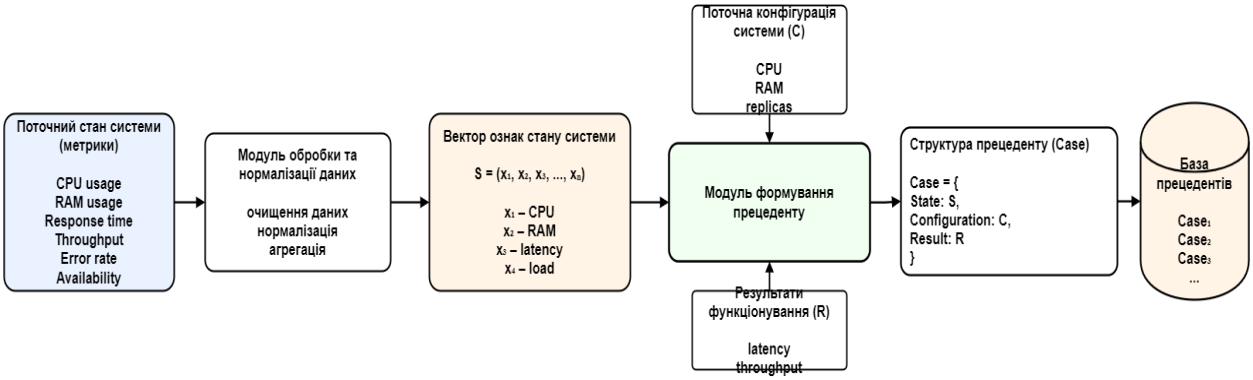


Рисунок 4.6 – Процес формування та представлення прецедентів у системі адаптивного управління конфігураціями мікросервісів

У програмному застосунку реалізовано п’ять методів пошуку прецедентів, що були обґрунтовані у розділі 2:

– метод найближчого сусіда;

- зважений метод k-найближчих сусідів;
- пошук на основі ознак;
- кластерно-орієнтований пошук;
- метод індексації та хешування [29, 33, 68, 87].

Використання декількох методів пошуку дозволяє проводити порівняльний аналіз їх ефективності в процесі експериментального дослідження [10–13, 93].

Після вибору релевантного прецеденту здійснюється етап адаптації конфігурації до поточного стану системи. Цей процес включає коригування параметрів з урахуванням відмінностей між поточним станом та умовами, у яких було отримано вибраний прецедент. Сформована конфігурація передається до тестового полігону через програмні інтерфейси, що дозволяє змінювати параметри функціонування мікросервісів у режимі, наближеному до реального часу. Таким чином реалізується керуючий вплив на систему, що забезпечує адаптацію її поведінки до змін навантаження [70, 75].

Інтеграція програмного застосунку АУКМ з тестовим полігоном забезпечує реалізацію замкненого контуру управління, який включає послідовність етапів:

- збір метрик;
- аналіз стану системи;
- пошук релевантного прецеденту;
- формування нової конфігурації;
- застосування конфігурації;
- оцінка результату.

Циклічне виконання зазначених етапів забезпечує безперервну адаптацію системи до змін умов функціонування та накопичення досвіду у базі прецедентів.

Реалізація системи виконана із використанням сучасних програмних засобів та технологій, які забезпечують підтримку мікросервісної архітектури, обробку даних та реалізацію інтелектуальних алгоритмів [34, 36, 37, 87].

В якості серверної платформи використано мову програмування Python, що дозволило ефективно інтегрувати бібліотеки машинного навчання та реалізувати алгоритми аналізу прецедентів. Для побудови REST API використано фреймворк FastAPI, який забезпечує високу продуктивність та зручність опису сервісних інтерфейсів [23, 24].

Зберігання даних реалізовано за допомогою системи керування базами даних PostgreSQL, що забезпечує надійність, масштабованість та підтримку складних структур даних [83, 85]. Для роботи з базою даних використано ORM-бібліотеку SQLAlchemy, яка дозволяє реалізувати абстракцію доступу до даних та спростити управління прецедентами.

Контейнеризація системи виконана із застосуванням Docker, що дозволило забезпечити ізоляцію компонентів, спростити розгортання та підтримати принципи DevOps [23, 24, 34, 36]. Використання контейнерів також дозволяє моделювати реальне середовище функціонування мікросервісів.

Для реалізації алгоритмів CBR використано бібліотеку scikit-learn, яка забезпечує інструменти для реалізації методів класифікації та пошуку найближчих сусідів (k-NN), що є основою механізму пошуку релевантних прецедентів [29, 33, 68].

Модуль збору метрик забезпечує отримання інформації про стан мікросервісного середовища в реальному часі. До основних метрик, що використовуються у системі, належать:

- завантаження процесора;
- використання оперативної пам’яті;
- використання дискового простору;
- мережеве навантаження;
- час відповіді сервісів;

– показники доступності.

Збір метрик реалізовано шляхом інтеграції з моніторинговими інструментами та внутрішніми механізмами логування мікросервісів. Отримані дані передаються до модуля обробки, де здійснюється їх нормалізація, фільтрація та підготовка до подальшого використання у CBR-модулі.

Особливістю реалізації є приведення метрик до уніфікованого формату, що дозволяє забезпечити коректне порівняння прецедентів. Зокрема, використовується масштабування значень у діапазоні [0;1], що є необхідною умовою для застосування методів відстані у багатовимірному просторі.

База прецедентів є ключовим компонентом системи, що забезпечує накопичення та збереження досвіду роботи системи у вигляді набору конфігурацій та відповідних їм умов функціонування. Кожен прецедент представлений у вигляді структурованого запису, який включає:

– вектор вхідних параметрів (метрики системи);
– конфігураційні параметри мікросервісів;
– результати застосування конфігурації (продуктивність, затримки, витрати ресурсів);
– додаткові характеристики (час адаптації, кількість змін конфігурацій).

Зберігання прецедентів у реляційній базі даних дозволяє ефективно виконувати запити та забезпечує можливість масштабування бази знань.

CBR-модуль реалізує основні етапи методу аналізу прецедентів: пошук, повторне використання, коригування та збереження нових рішень. На етапі пошуку здійснюється визначення найбільш подібних прецедентів до поточного стану системи. Після вибору найбільш релевантного прецеденту відбувається етап адаптації конфігурації, який може включати часткову зміну параметрів відповідно до поточного стану системи.

Модуль застосування конфігурацій відповідає за внесення змін у параметри мікросервісів. Це може включати: зміну кількості реплік сервісів; налаштування ресурсних обмежень; зміну параметрів балансування

навантаження. Застосування конфігурацій здійснюється через API мікросервісної платформи або оркестратора контейнерів. Важливою особливістю є забезпечення узгодженості змін та уникнення конфліктів між конфігураціями різних сервісів.

Після застосування нової конфігурації система переходить у режим моніторингу результатів, що дозволяє оцінити ефективність прийнятого рішення. Отримані результати використовуються для:оновлення існуючих прецедентів; додавання нових записів до бази знань; коригування параметрів алгоритмів.

Такий підхід забезпечує самонавчання системи та підвищення якості прийняття рішень з часом.

Взаємодія всіх компонентів програмної реалізації утворює єдиний замкнений цикл адаптивного управління. Дані, отримані з мікросервісного середовища, проходять етапи обробки, аналізу та використання для прийняття рішень, після чого результати знову впливають на стан системи. Реалізована система відповідає концепції інтелектуального управління, в якій рішення приймаються на основі накопиченого досвіду та поточного стану середовища.

Таким чином було розглянуто особливості програмної реалізації запропонованого підходу до адаптивного управління конфігураціями мікросервісів. Визначено архітектуру програмного застосунку АУКМ, описано механізми формування прецедентів, реалізації методів їх пошуку та адаптації конфігурацій.

Показано, що реалізована система забезпечує повний цикл адаптивного управління, узгоджений із розробленими у розділі 3 моделями, що створює основу для проведення експериментального дослідження ефективності запропонованого підходу.

Результати тестування полігону

В процесі роботи було проведене тестування роботи тестового полігону, яке підтвердило коректність реалізації функціональних можливостей усіх трьох мікросервісів та їх взаємодії в межах мікросервісної архітектури.

Зокрема, для сервісу автентифікації (auth_service) було перевірено механізми реєстрації користувачів, авторизації та розмежування прав доступу, що продемонструвало коректну роботу системи ролей і сесійної автентифікації. Також підтверджено відповідність валідаційних обмежень для вхідних даних та збереження токенів у клієнтському середовищі.

Для сервісу управління продуктами (product_service) експериментально підтверджено працездатність операцій створення та перегляду продуктів, а також реалізацію обмежень доступу до адміністративних функцій. Додатково встановлено наявність службових маршрутів моніторингу стану сервісів, що є важливим для подальшої інтеграції з системою адаптивного управління.

У межах тестування сервісу обробки замовлень (order_service) підтверджено коректність виконання операцій створення та отримання замовлень, а також реалізацію міжсервісної взаємодії з product_service, що проявляється у зміні залишків товарів при оформленні замовлень. Крім того, перевірено розмежування доступу до адміністративних функцій перегляду всіх замовлень і керування конфігураціями.

У цілому результати тестування підтверджують, що розроблений тестовий полігон забезпечує коректну реалізацію бізнес-логіки, підтримує міжсервісну взаємодію та надає необхідні інтерфейси для моніторингу стану системи. Це створює надійну основу для подальшого використання полігону як експериментального середовища для дослідження ефективності адаптивного управління конфігураціями мікросервісів.

Інтерфейс і результати тестування застосунку АУКМ

Результати реалізації інтерфейсу системи адаптивного управління конфігураціями мікросервісів свідчать про забезпечення комплексного контролю за станом системи та можливістю її оперативного налаштування. Інтерфейс побудований за принципом односторінкового застосунку та передбачає авторизований доступ із розмежуванням прав, що підтверджує коректну реалізацію механізмів безпеки (рисунок 4.7).

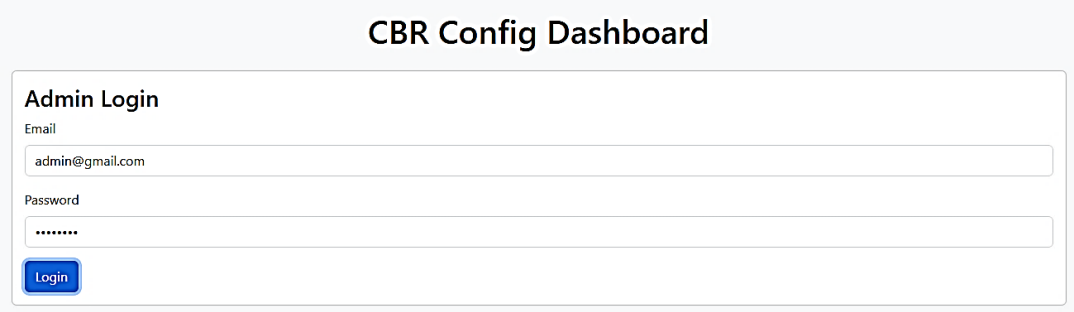


Рисунок 4.7 – Вікно авторизації в застосунку АУКМ

Функціональні можливості інтерфейсу включають відображення ключових груп метрик, які характеризують стан системи. Зокрема, метрики якості обслуговування дозволяють оцінювати затримки, час відповіді та доступність сервісів (рисунок 4.8), тоді як системні метрики забезпечують контроль за використанням обчислювальних ресурсів (рисунок 4.9). Окремо реалізовано показник операційних витрат, що інтегрує ресурсні характеристики у єдину узагальнену метрику ефективності.

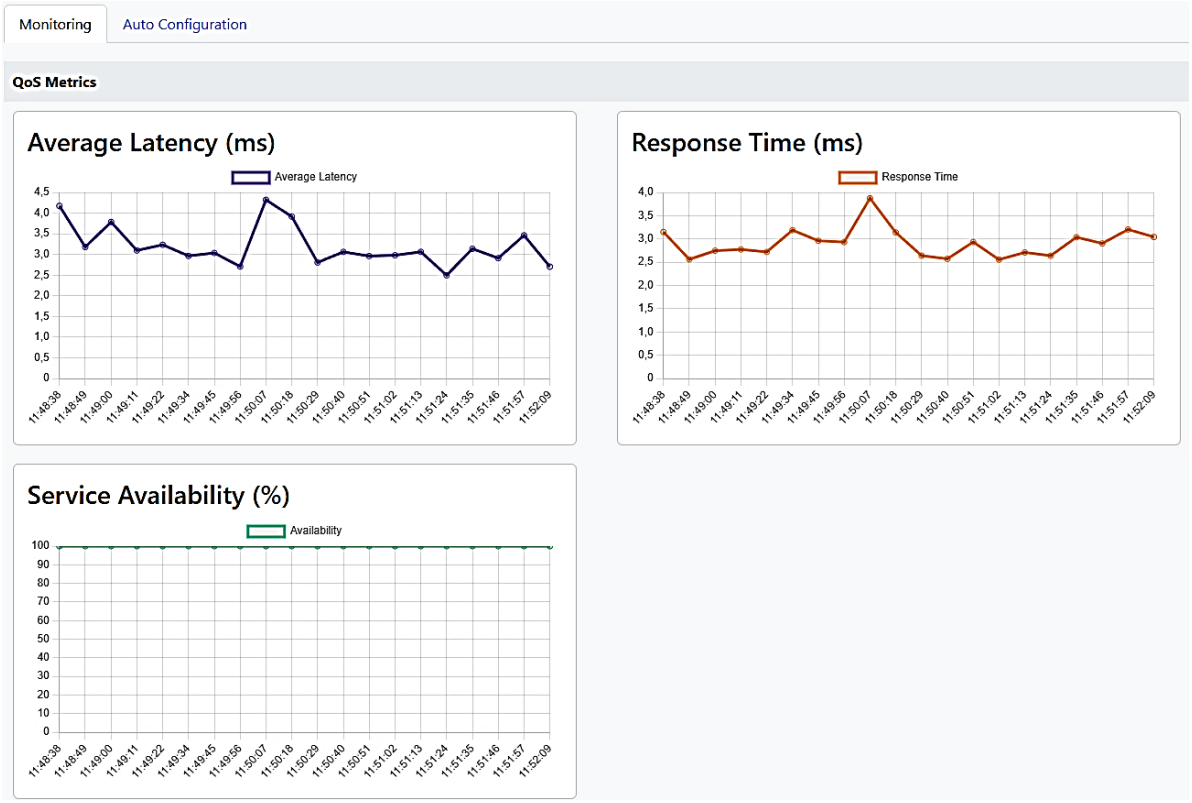


Рисунок 4.8 - Відображення ключових груп метрик категорія «QoS Metrics»

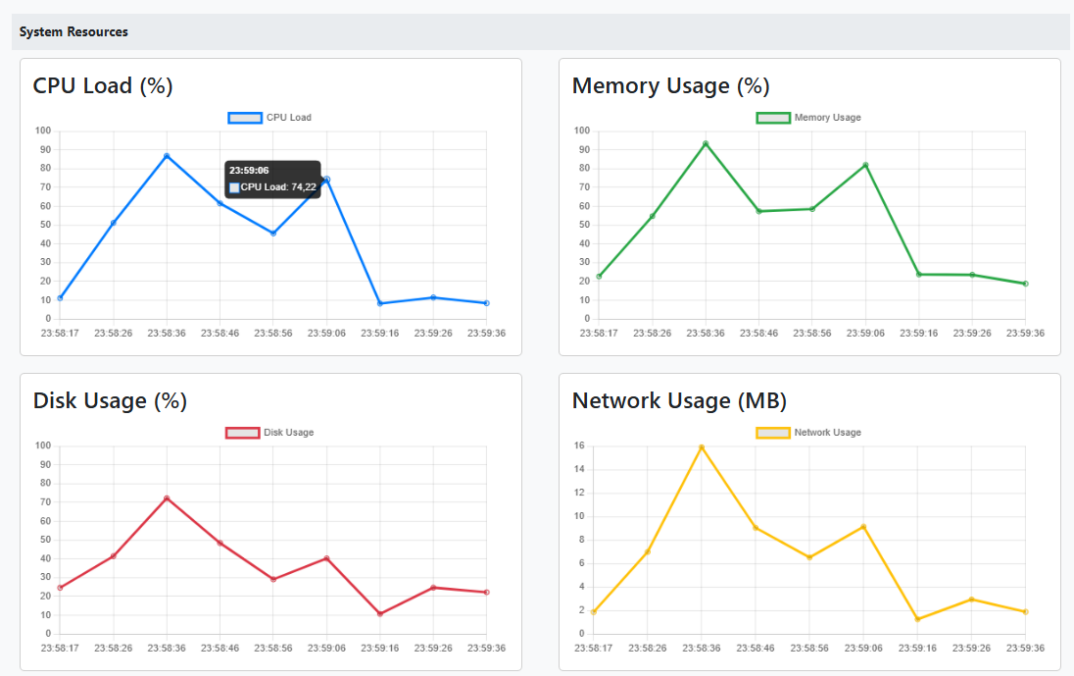


Рисунок 4.9 – Моніторинг використанням обчислювальних ресурсів

Важливою особливістю є наявність метрик адаптивності, які відображають динаміку роботи механізму автоматичного конфігурування, зокрема час адаптації та кількість реконфігурацій (рисунок 4.10). Це дозволяє безпосередньо оцінювати ефективність застосування CBR-підходу в процесі експлуатації системи.

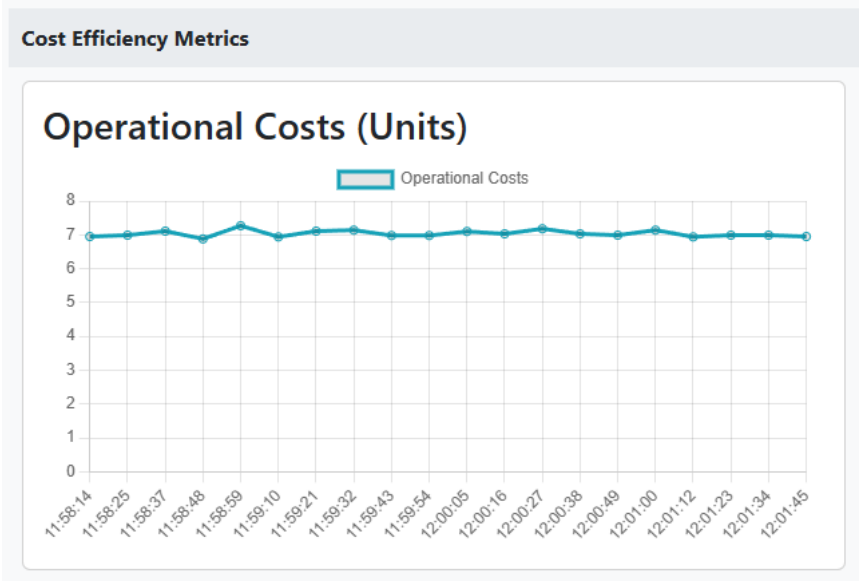


Рисунок 4.10 – Відображення метрики адаптивності категорія «Cost Efficiency Metrics»

Крім функцій моніторингу, інтерфейс забезпечує можливість перегляду та ручного коригування конфігурацій мікросервісів (рисунок 4.11), що є важливим для проведення експериментальних досліджень та порівняння автоматичних і ручних режимів управління.

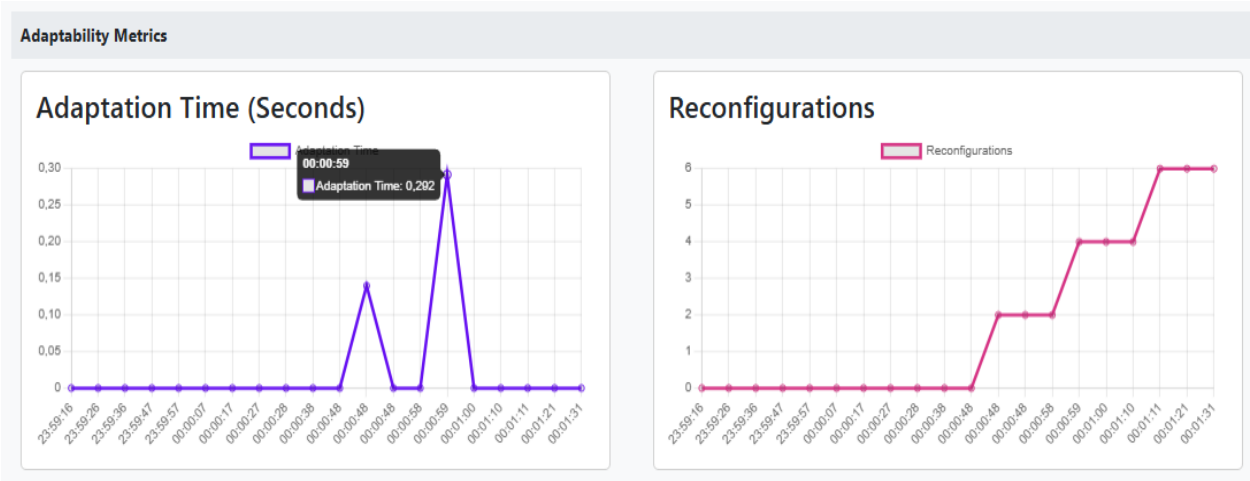
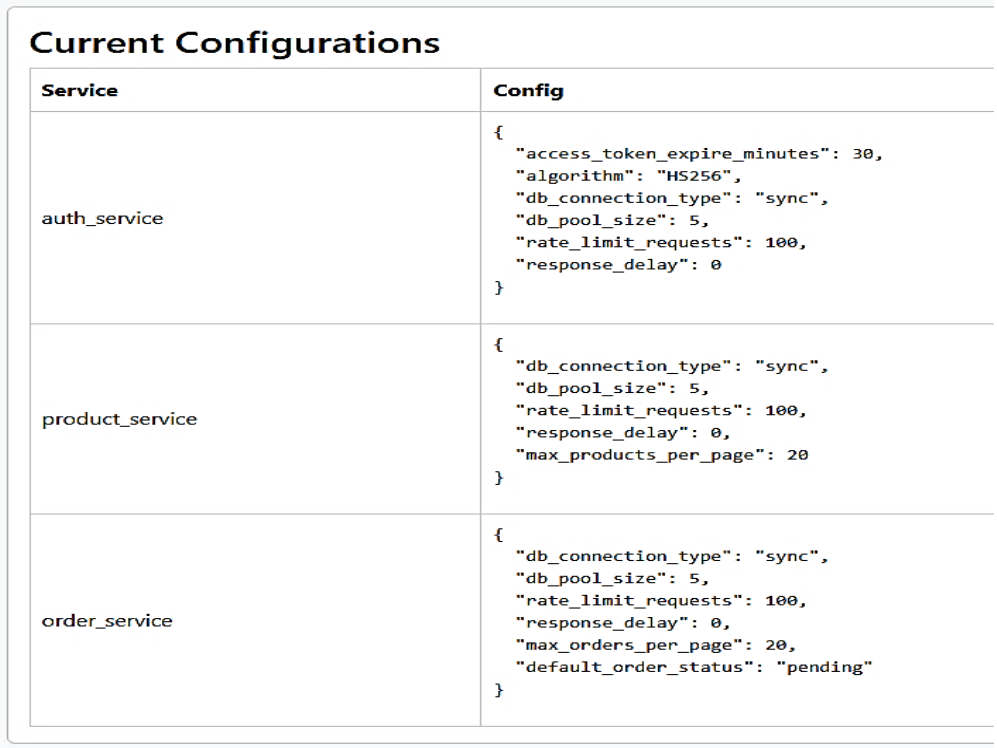


Рисунок 4.11 – Перегляд та ручне коригування конфігурацій мікросервісів (категорія «Adaptability Metrics»)

На рисунку 4.12 показані поточні конфігурації мікросервісів полігону.



Current Configurations

Service	Config
auth_service	{ "access_token_expire_minutes": 30, "algorithm": "HS256", "db_connection_type": "sync", "db_pool_size": 5, "rate_limit_requests": 100, "response_delay": 0 }
product_service	{ "db_connection_type": "sync", "db_pool_size": 5, "rate_limit_requests": 100, "response_delay": 0, "max_products_per_page": 20 }
order_service	{ "db_connection_type": "sync", "db_pool_size": 5, "rate_limit_requests": 100, "response_delay": 0, "max_orders_per_page": 20, "default_order_status": "pending" }

Рисунок 4.12 - Поточні конфігурації для усіх мікросервісів тестого полігону.

Рисунок 4.13 демонструє можливість зміни і результат успішної зміни поточної конфігурації в auth_service.

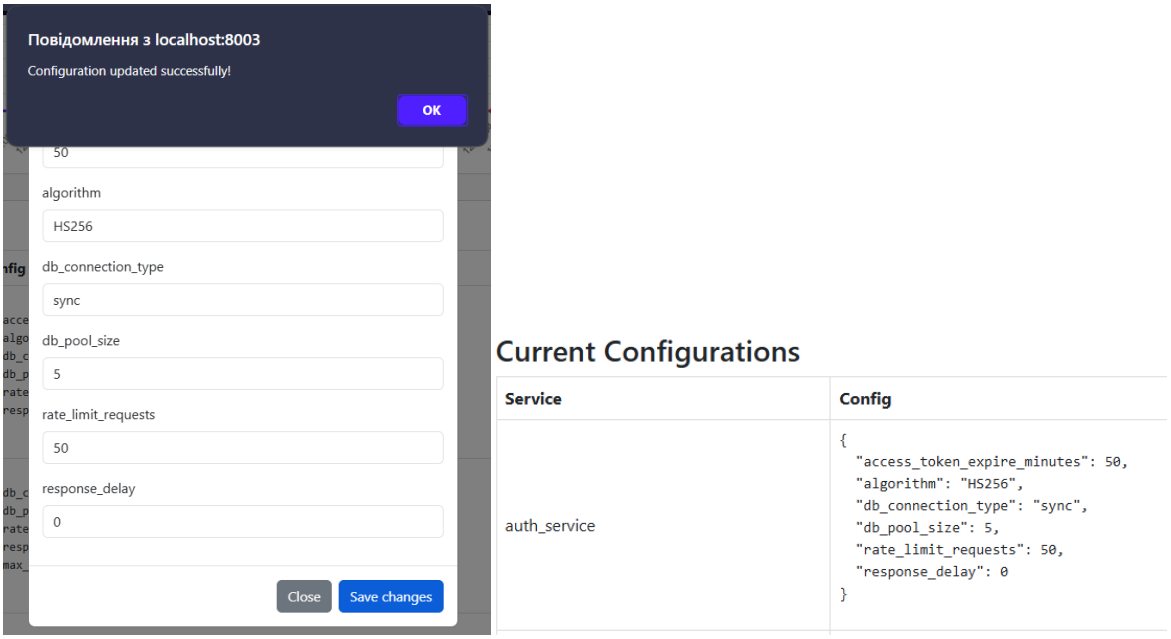


Рисунок 4.13 - Демонстрація можливості і результат успішної зміни поточної конфігурації в auth_service.

Таким чином, реалізований інтерфейс системи АУКМ забезпечує інтегроване представлення стану системи, підтримує прийняття рішень щодо конфігурації мікросервісів та створює зручне середовище для проведення експериментів з адаптивного управління.

Результати тестування функціональних можливостей системи АУКМ підтверджують коректність реалізації як ручного, так і автоматичного режимів управління конфігураціями мікросервісів. Зокрема, експериментальна зміна параметрів конфігурації для auth_service продемонструвала успішне застосування нових значень і їх вплив на поведінку системи, що свідчить про працездатність механізму керування конфігураціями.

Подальший аналіз функціоналу автоматичного конфігурування показав, що система забезпечує гнучке налаштування процесу прийняття рішень на основі CBR-підходу. Інтерфейс вкладки автоматичного конфігурування

дозволяє обирати метод пошуку прецедентів та задавати параметри його роботи, що відображено на рисунку 4.14.

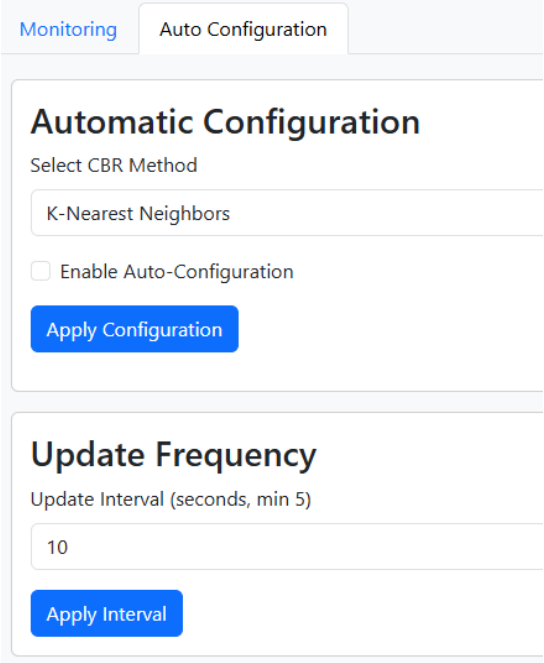


Рисунок 4.14 – Вигляд вкладки автоматичної конфігурації

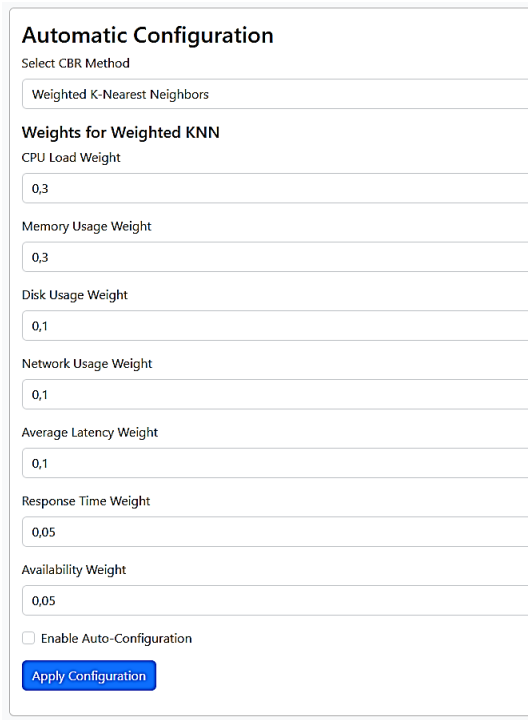


Рисунок 4.15 – Параметри для методу зважених k-найближчих сусідів (Weighted K-Nearest Neighbors)

Раніше було показано, що різні методи CBR мають різні вимоги до параметризації: зокрема, метод зважених k-найближчих сусідів передбачає визначення ваг для окремих метрик (рисунок 4.15), тоді як метод відбору за ознаками дозволяє формувати підмножину релевантних характеристик (рисунок 4.16). Кластерно-орієнтований підхід, у свою чергу, використовує параметр кількості кластерів, що впливає на структуру простору прецедентів (рисунок 4.17).

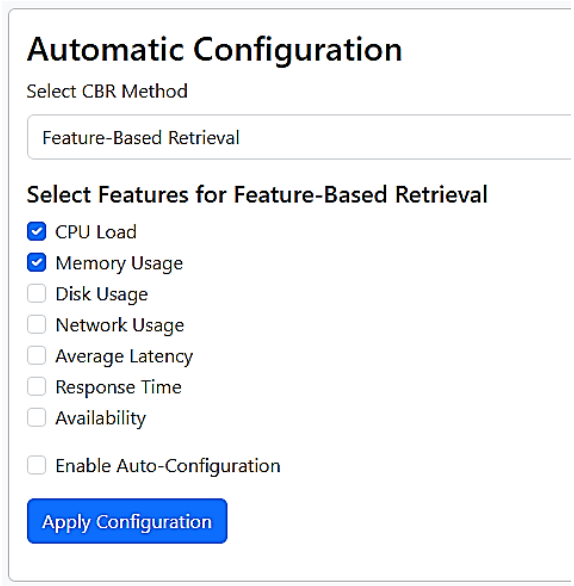


Рисунок 4.16 – Параметри для методу Feature-Based Retrieval

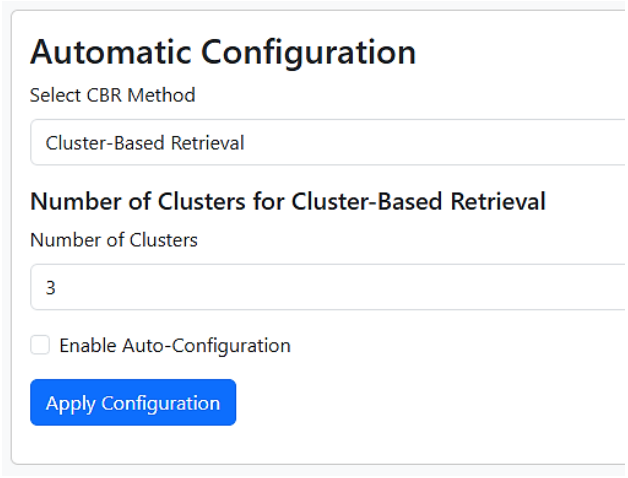


Рисунок 4.17 – Вікно налаштування параметрів для методу кластерного пошуку (Cluster-Based Retrieval)

При вдалому випадку заміни конфігурацій, користувач отримує повідомлення про це в окремому вікні (дивись рисунок 4.18)

Рисунок 4.18 – Вікно з результатом застосування конфігурацій методом k-Nearest Neighbors

Результати застосування методів автоматичного конфігурування підтверджують здатність системи підбирати конфігурації на основі наявної бази прецедентів. Зокрема, при використанні методу k-найближчих сусідів було автоматично обрано конфігурацію, що відповідає поточному стану системи, про що свідчить відповідне повідомлення інтерфейсу (рисунок 4.42).

Таким чином, реалізований механізм автоматичного конфігурування забезпечує адаптивний підбір конфігурацій мікросервісів з урахуванням поточного стану системи та параметрів обраного методу, що створює основу для подальших експериментальних досліджень ефективності різних CBR-підходів.

4.3. Методика проведення програмних експериментів та опис тестових конфігурацій програмних мікросервісів

Для забезпечення відтворюваності різних режимів функціонування системи було реалізовано програмний механізм імітації навантаження, який генерує значення ключових метрик у заданих інтервалах [59–61, 88]. Діапазони значень параметрів визначено відповідно до характерних сценаріїв навантаження: **low_load**, **medium_load**, **high_load**, **peak_load** та **suboptimal**, що наведено у таблиці 4.1.

Таблиця 4.1 – Діапазони значень метрик до кожного зі сценаріїв

Метрика	Low Load	Medium Load	High Load	Peak Load	Suboptimal
cpu_load	5.0-20.0	30.0-50.0	60.0-80.0	80.0-95.0	40.0-70.0
memory_usage	15.0-30.0	40.0-60.0	65.0-85.0	85.0-95.0	50.0-80.0
disk_usage	10.0-25.0	25.0-40.0	40.0-60.0	60.0-75.0	30.0-50.0
network_usage	0.5-3.0 MB/s	3.0-8.0 MB/s	8.0-15.0 MB/s	15.0-25.0 MB/s	5.0-10.0 MB/s
performance_metric	10.0-30.0 ms	30.0-80.0 ms	80.0-150.0 ms	150.0-300.0 ms	200.0-400.0 ms

Закінчення Таблиці 4.1

Метрика	Low Load	Medium Load	High Load	Peak Load	Suboptimal
response_time	5.0-20.0 ms	20.0-50.0 ms	50.0-100.0 ms	100.0-200.0 ms	150.0-300.0 ms
operational_costs	5.0-15.0 units	15.0-30.0 units	30.0-50.0 units	50.0-80.0 units	25.0-60.0 units
availability	95.0-100.0%	90.0-100.0%	85.0-95.0%	80.0-90.0%	70.0-85.0%
adaptation_time	0.1-0.5 s	0.5-1.0 s	1.0-2.0 s	2.0-5.0 s	1.5-3.0 s
reconfigurations	0-2	1-5	3-8	5-10	2-6

Проведення експериментів було розпочато з використання методу **k-найближчих сусідів (KNN)**. Для зручності аналізу результати роботи системи відображалися у консольному інтерфейсі, що дозволяло одночасно спостерігати як згенеровані значення метрик, так і відповідні їм конфігурації (рисунок 4.19).

```
2025-06-05 09:21:54,352 - INFO - Starting synthetic status generation
2025-06-05 09:21:54,353 - INFO - Selected scenario: medium_load
2025-06-05 09:21:54,354 - INFO - Completed synthetic status generation
2025-06-05 09:21:54,355 - INFO - System status fetched successfully: {
  "cpu_load": 40.95,
  "memory_usage": 41.9,
  "disk_usage": 38.52,
  "network_usage": 3.47,
  "performance_metric": 50.69,
  "response_time": 49.94,
  "operational_costs": 37.0,
  "availability": 98.18
}
```

Рисунок 4.19 – Результат генерації поточних метрик системи

Як видно з рисунка 4.19, на початковому етапі здійснюється генерація поточного стану системи відповідно до обраного сценарію навантаження. У наведеному випадку система була віднесена до середнього рівня навантаження, що зумовило формування відповідних значень метрик.

На наступному етапі виконується пошук найбільш релевантного прецеденту в базі знань (рисунок 4.20).

```
2025-06-05 09:21:54,363 - INFO - Best case selected:
performance_metric=50.36
2025-06-05 09:21:54,363 - INFO - Retrieved config: {
  "auth_service": {
    "access_token_expire_minutes": 60,
    "algorithm": "HS256",
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1
  },
  "product_service": {
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_products_per_page": 30
  },
  "order_service": {
    "db_connection_type": "sync",
    "db_pool_size": 5,
    "rate_limit_requests": 100,
    "response_delay": 0.0,
    "max_orders_per_page": 20,
    "default_order_status": "pending"
  }
}
```

Рисунок 4.20 – Результат пошуку найкращого сценарію

Після вибору прецеденту система переходить до застосування відповідних конфігурацій для кожного мікросервісу (рисунок 4.21).

Завершальним етапом є відображення результатів застосування конфігурації у користувацькому інтерфейсі, що підтверджує успішність виконання операції (рисунок 4.22).

Слід зазначити, що для формування бази прецедентів було використано окремий генератор, який створює множину записів, що відповідають різним сценаріям навантаження, визначеним у таблиці 4.1. Це дозволило забезпечити достатню репрезентативність бази знань для коректної роботи алгоритмів CBR.

```
2025-06-05 09:21:54,363 - INFO - Best case selected:
performance_metric=50.36
2025-06-05 09:21:54,363 - INFO - Retrieved config: {
  "auth_service": {
    "access_token_expire_minutes": 60,
    "algorithm": "HS256",
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1
  },
  "product_service": {
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_products_per_page": 30
  },
  "order_service": {
    "db_connection_type": "sync",
    "db_pool_size": 5,
    "rate_limit_requests": 100,
    "response_delay": 0.0,
    "max_orders_per_page": 20,
    "default_order_status": "pending"
  }
}
```

Рисунок 4.21 – Результат застосування конфігурації знайденого найкращого сценарію до кожного з мікросервісів

Configuration applied successfully!
Last Updated: 2025-06-05 12:21:54
Method: K-Nearest Neighbors
Parameters: k=3 (default)
Auto-Configuration: Disabled
New config:


```
{
  "auth_service": {
    "access_token_expire_minutes": 60,
    "algorithm": "HS256",
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1
  },
  "product_service": {
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_products_per_page": 30
  },
  "order_service": {
    "db_connection_type": "sync",
    "db_pool_size": 5,
    "rate_limit_requests": 100,
    "response_delay": 0,
    "max_orders_per_page": 20,
    "default_order_status": "pending"
  }
}
```


Рисунок 4.22 – Результат вдалого застосування конфігу до системи з поточними метриками

Результати застосування методу **Weighted K-Nearest Neighbors** наведено на рисунку 4.23.

```
2025-06-05 10:06:08,683 - INFO - Configuration applied successfully!
Last Updated: 2025-06-05 10:06:08
Method: Weighted Knn
Weights: {
  "cpu_load": 0.3,
  "memory_usage": 0.3,
  "disk_usage": 0.1,
  "network_usage": 0.1,
  "performance_metric": 0.1,
  "response_time": 0.05,
  "availability": 0.05
}
Auto-Configuration: Enabled
Update Interval: 30 seconds
New config:
{
  "auth_service": {
    "access_token_expire_minutes": 30,
    "algorithm": "HS256",
    "db_connection_type": "sync",
    "db_pool_size": 5,
    "rate_limit_requests": 100,
    "response_delay": 0.0
  },
  "product_service": {
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_products_per_page": 30
  },
  "order_service": {
    "db_connection_type": "sync",
    "db_pool_size": 8,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_orders_per_page": 30,
    "default_order_status": "pending"
  }
}
Applied for the following metrics (System state: medium_load):
{
  "cpu_load": 36.75,
  "memory_usage": 50.02,
  "disk_usage": 26.8,
  "network_usage": 3.42,
  "performance_metric": 45.2,
  "response_time": 47.11,
  "operational_costs": 35.41,
  "availability": 97.46
}
```

Рисунок 4.23 – Результат застосування конфігів методом Weighted K-Nearest Neighbors

Отримані результати демонструють, що використання вагових коефіцієнтів дозволяє враховувати значущість окремих метрик при виборі конфігурації, що забезпечує більш точне визначення сценарію навантаження.

На рисунку 4.24 наведено результати використання методу **Feature-Based Retrieval**.

```
2025-06-05 10:06:08,683 - INFO - Configuration applied successfully!
Last Updated: 2025-06-05 10:06:08
Method: Weighted Knn
Weights: {
  "cpu_load": 0.3,
  "memory_usage": 0.3,
  "disk_usage": 0.1,
  "network_usage": 0.1,
  "performance_metric": 0.1,
  "response_time": 0.05,
  "availability": 0.05
}
Auto-Configuration: Enabled
Update Interval: 30 seconds
New config:
{
  "auth_service": {
    "access_token_expire_minutes": 30,
    "algorithm": "HS256",
    "db_connection_type": "sync",
    "db_pool_size": 5,
    "rate_limit_requests": 100,
    "response_delay": 0.0
  },
  "product_service": {
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_products_per_page": 30
  },
  "order_service": {
    "db_connection_type": "sync",
    "db_pool_size": 8,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_orders_per_page": 30,
    "default_order_status": "pending"
  }
}
Applied for the following metrics (System state: medium_load):
{
  "cpu_load": 36.75,
  "memory_usage": 50.02,
  "disk_usage": 26.8,
  "network_usage": 3.42,
  "performance_metric": 45.2,
  "response_time": 47.11,
  "operational_costs": 35.41,
  "availability": 97.46
}
```

Рисунок 4.24 – Результат застосування конфігів методом Feature-Based Retrieval

У даному випадку вибір конфігурації здійснюється на основі підмножини ключових ознак, що може призводити до менш оптимальних рішень у випадках складних або неоднозначних станів системи.

Результати застосування кластерно-орієнтованого підходу представлені на рисунку 4.25.

```
2025-06-05 10:06:08,683 - INFO - Configuration applied successfully!
Last Updated: 2025-06-05 10:06:08
Method: Weighted Knn
Weights: {
  "cpu_load": 0.3,
  "memory_usage": 0.3,
  "disk_usage": 0.1,
  "network_usage": 0.1,
  "performance_metric": 0.1,
  "response_time": 0.05,
  "availability": 0.05
}
Auto-Configuration: Enabled
Update Interval: 30 seconds
New config:
{
  "auth_service": {
    "access_token_expire_minutes": 30,
    "algorithm": "HS256",
    "db_connection_type": "sync",
    "db_pool_size": 5,
    "rate_limit_requests": 100,
    "response_delay": 0.0
  },
  "product_service": {
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_products_per_page": 30
  },
  "order_service": {
    "db_connection_type": "sync",
    "db_pool_size": 8,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_orders_per_page": 30,
    "default_order_status": "pending"
  }
}
Applied for the following metrics (System state: medium_load):
{
  "cpu_load": 36.75,
  "memory_usage": 50.02,
  "disk_usage": 26.8,
  "network_usage": 3.42,
  "performance_metric": 45.2,
  "response_time": 47.11,
  "operational_costs": 35.41,
  "availability": 97.46
}
```

Рисунок 4.25 – Результат застосування конфігів методом Cluster-Based Retrieval

Використання кластеризації дозволяє ефективно групувати прецеденти та зменшувати обчислювальну складність пошуку, забезпечуючи при цьому коректний вибір конфігурації для відповідного рівня навантаження.

На рисунку 4.26 наведено результати застосування методу **Indexing and Hashing**.


```
2025-06-05 10:06:08,683 - INFO - Configuration applied successfully!
Last Updated: 2025-06-05 10:06:08
Method: Weighted Knn
Weights: {
  "cpu_load": 0.3,
  "memory_usage": 0.3,
  "disk_usage": 0.1,
  "network_usage": 0.1,
  "performance_metric": 0.1,
  "response_time": 0.05,
  "availability": 0.05
}
Auto-Configuration: Enabled
Update Interval: 30 seconds
New config:
{
  "auth_service": {
    "access_token_expire_minutes": 30,
    "algorithm": "HS256",
    "db_connection_type": "sync",
    "db_pool_size": 5,
    "rate_limit_requests": 100,
    "response_delay": 0.0
  },
  "product_service": {
    "db_connection_type": "sync",
    "db_pool_size": 7,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_products_per_page": 30
  },
  "order_service": {
    "db_connection_type": "sync",
    "db_pool_size": 8,
    "rate_limit_requests": 150,
    "response_delay": 0.1,
    "max_orders_per_page": 30,
    "default_order_status": "pending"
  }
}
Applied for the following metrics (System state: medium_load):
{
  "cpu_load": 36.75,
  "memory_usage": 50.02,
  "disk_usage": 26.8,
  "network_usage": 3.42,
  "performance_metric": 45.2,
  "response_time": 47.11,
  "operational_costs": 35.41,
  "availability": 97.46
}
```


Рисунок 4.26 – Результат застосування конфігів методом Indexing and Hashing

Застосування індексування забезпечує швидкий доступ до релевантних прецедентів, що є особливо важливим у випадках високого навантаження та необхідності оперативного прийняття рішень.

Таким чином, результати проведених експериментів підтверджують, що всі реалізовані методи CBR забезпечують коректний підбір конфігурацій відповідно до сценаріїв навантаження, визначених у таблиці 4.1, що свідчить про працездатність запропонованого підходу до адаптивного управління конфігураціями мікросервісів.

4.4. Аналіз отриманих результатів і практичні рекомендації щодо використання запропонованого підходу

Порівняльне дослідження швидкодії реалізованих CBR-методів виконувалося в умовах тестового полігону при динамічній зміні значень системних метрик. Оцінювання часу виконання алгоритмів проводилося залежно від обсягу бази прецедентів, яка варіювалася в межах 50, 100, 200, 500 та 1000 записів. Для підвищення достовірності результатів кожне вимірювання повторювалося 100 разів із подальшим обчисленням середнього арифметичного значення.

Експериментальне середовище характеризувалося такими параметрами: операційна система Windows 10 (версія 22H2), оперативна пам’ять обсягом 16 ГБ (DDR4, 3200 MHz) та процесор Ryzen 5 4600H (6 ядер, 12 потоків).

На рисунку 4.27 наведені результати досліджень залежність часу виконання алгоритму п’яти CBR методів від кількості прецедентів у базі знань.

Першим досліджуваним методом був *K-Nearest Neighbors* з параметром (k=3). Отримані результати показали, що середній час виконання алгоритму становить 44,8 мс для 50 записів, 58,3 мс для 100 записів, 68,9 мс для 200 записів, 105,2 мс для 500 записів та 152,4 мс для 1000 записів. Зростання часу виконання має лінійний характер, що пояснюється необхідністю послідовного порівняння поточного стану системи з усіма прецедентами бази знань.

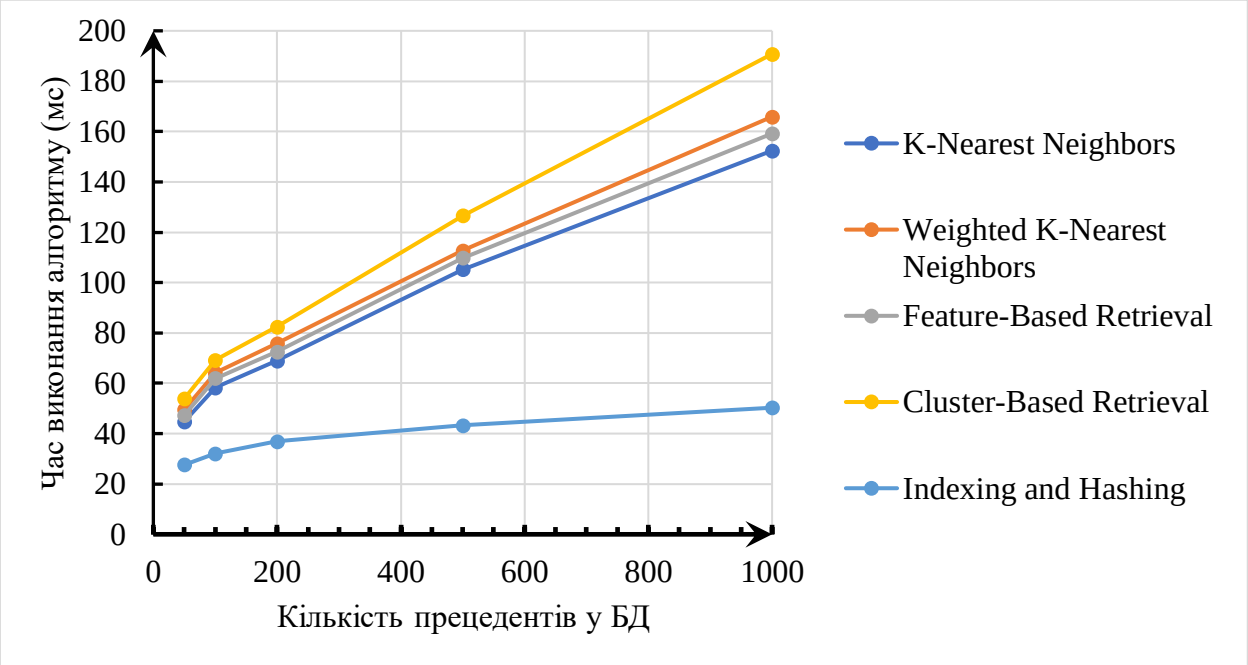


Рисунок 4.27 – Залежність часу виконання алгоритмів CBR від кількості прецедентів

Другим методом було досліджено *Weighted K-Nearest Neighbors*, для якого встановлено вагові коефіцієнти: cpu_load – 0,3, memory_usage – 0,3, інші параметри – в межах 0,1–0,05. Середній час виконання становив 49,6 мс для 50 записів, 64,1 мс для 100 записів, 75,8 мс для 200 записів, 112,7 мс для 500 записів та 165,9 мс для 1000 записів. Порівняно з базовим методом, спостерігається додаткове зростання часу виконання, що обумовлено необхідністю врахування ваг при обчисленні відстаней.

Третім методом було реалізовано *Feature-Based Retrieval* із використанням підмножини ознак: cpu_load, performance_metric, response_time та availability. Середні значення часу виконання становили 47,3 мс (50 записів), 61,9 мс (100 записів), 72,5 мс (200 записів), 109,8 мс (500 записів) та 159,2 мс (1000 записів). Збільшення часу виконання зумовлене необхідністю попередньої фільтрації та обробки вибраних ознак.

Четвертий підхід - *Cluster-Based Retrieval* - досліджувався за умови розбиття бази прецедентів на 3 кластери. Середній час виконання становив

53,9 мс для 50 записів, 69,2 мс для 100 записів, 82,4 мс для 200 записів, 126,5 мс для 500 записів та 190,8 мс для 1000 записів.

П’ятим дослідженим методом був *Indexing and Hashing*, який продемонстрував найменші значення часу виконання: 27,6 мс для 50 записів, 32,1 мс для 100 записів, 36,9 мс для 200 записів, 43,2 мс для 500 записів та 50,3 мс для 1000 записів. Незважаючи на збільшення обсягу даних, зростання часу виконання залишається незначним, що пояснюється використанням механізмів попередньої індексації.

Узагальнені результати експериментального дослідження наведено в таблиці 4.2.

Таблиця 4.2 – Порівняння результатів виміру швидкодії CBR-методів в залежності від кількості записів у таблиці прецедентів

Метод	**Час виконання алгоритмів (мс)**				
	для 50 записів	**для 100 записів**	**для 200 записів**	**для 500 записів**	**для 1000 записів**
K-Nearest Neighbors	44,8	58,3	68,9	105,2	152,4
Weighted K-Nearest Neighbors	49,6	64,1	75,8	112,7	165,9
Feature-Based Retrieval	47,3	61,9	72,5	109,8	159,2
Cluster-Based Retrieval	53,9	69,2	82,4	126,5	190,8
Indexing and Hashing	27,6	32,1	36,9	43,2	50,3

На основі наведених даних побудовано узагальнену порівняльну діаграму (рисунок 4.28), яка наочно відображає відмінності у швидкодії досліджуваних методів.

Отримані результати дозволяють зробити висновок, що найвищу швидкодію демонструє метод Indexing and Hashing, що обумовлено застосуванням механізмів індексації, які суттєво скорочують час пошуку прецедентів . Водночас слід враховувати можливе зниження точності прийняття рішень через спрощення процедури зіставлення.

Метод KNN характеризується лінійною залежністю часу виконання від обсягу бази прецедентів, однак забезпечує стабільні результати при невеликому обсязі даних . Використання Weighted KNN дозволяє підвищити

гнучкість прийняття рішень за рахунок врахування вагових коефіцієнтів, проте супроводжується додатковими обчислювальними витратами .

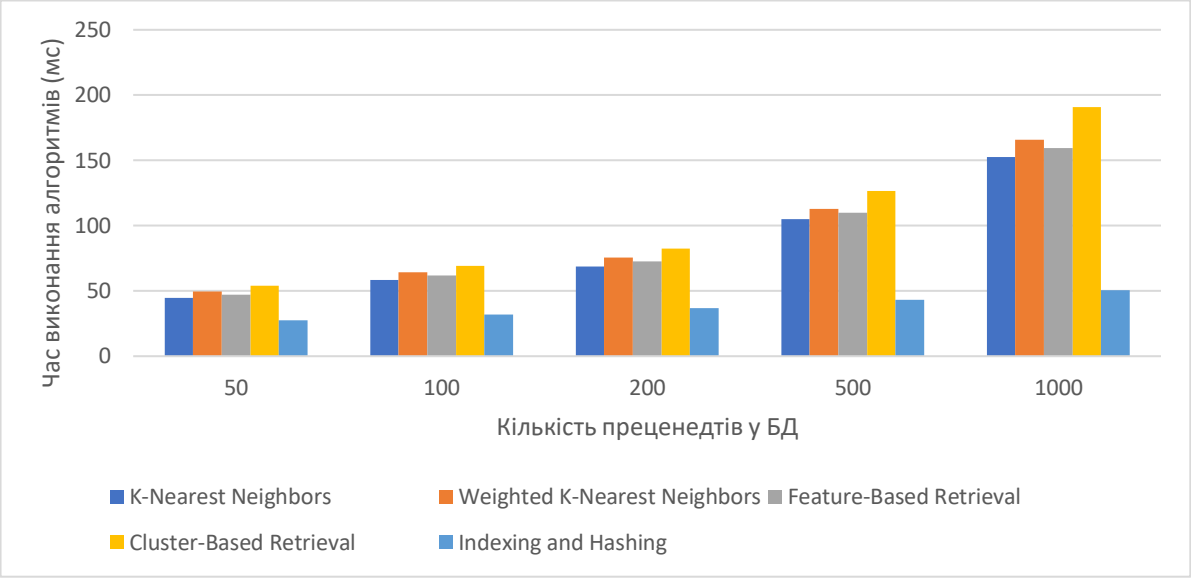


Рисунок 4.28 – Стовпчаста діаграма порівняння швидкодії п’яти CBR-методів

Методи Feature-Based Retrieval та Cluster-Based Retrieval займають проміжне положення, поєднуючи прийнятну швидкодію з можливістю врахування структурних особливостей даних, хоча й потребують додаткових обчислень, пов’язаних із відбором ознак або кластеризацією.

Таким чином, вибір конкретного методу пошуку прецедентів має здійснюватися з урахуванням пріоритетів задачі: використання Indexing and Hashing доцільне у випадках, коли критичною є швидкість прийняття рішень, тоді як Weighted KNN є більш придатним для задач, де важлива адаптивність та точність конфігураційних рішень.

4.5 Висновки до Розділу 4

У четвертому розділі дисертаційної роботи здійснено комплексне дослідження програмної реалізації запропонованого підходу до адаптивного управління конфігураціями мікросервісів, а також проведено експериментальну перевірку його ефективності в умовах тестового полігону.

У процесі дослідження розроблено тестовий полігон, який моделює функціонування мікросервісної архітектури в предметній області електронної торгівлі та включає сервіси автентифікації, управління продуктами та обробки замовлень. Підтверджено, що реалізований полігон забезпечує коректну взаємодію мікросервісів, підтримує змінні режими навантаження та надає необхідні інтерфейси для моніторингу стану системи і керування її конфігураціями.

Розроблено програмний застосунок адаптивного управління конфігураціями мікросервісів, який реалізує замкнений контур управління, що включає збір метрик, аналіз стану системи, пошук релевантних прецедентів, формування конфігураційних рішень та їх застосування. Показано, що архітектура програмного засобу узгоджується з розробленою у розділі 3 алгоритмічною моделлю та забезпечує практичну реалізацію CBR-підходу.

У межах експериментального дослідження реалізовано п’ять методів пошуку прецедентів: Nearest Neighbor, Weighted KNN, Feature-Based Retrieval, Cluster-Based Retrieval та Indexing and Hashing. Проведено серію обчислювальних експериментів із варіюванням обсягу бази прецедентів і сценаріїв навантаження, що дозволило отримати кількісні оцінки часу виконання алгоритмів і дослідити їх поведінку в різних умовах.

Аналіз отриманих результатів показав, що:

- метод Indexing and Hashing забезпечує найвищу швидкодію (50,3 мс для 1000 прецедентів), що обумовлено використанням механізмів індексації;
- методи KNN та Weighted KNN демонструють лінійну залежність часу виконання від обсягу бази, забезпечуючи при цьому вищу точність підбору конфігурацій;
- методи Feature-Based Retrieval та Cluster-Based Retrieval забезпечують компроміс між швидкодією та складністю обробки даних.

Встановлено, що вибір методу пошуку прецедентів має здійснюватися з урахуванням вимог до швидкодії та точності: для задач реального часу

доцільно використовувати Indexing and Hashing, тоді як для задач, що потребують більшої адаптивності, доцільним є застосування Weighted KNN.

Отримані результати експериментального дослідження підтверджують працездатність запропонованого підходу, його здатність забезпечувати адаптивну зміну конфігурацій мікросервісів відповідно до поточного стану системи, а також ефективність використання методів аналізу прецедентів для вирішення задач управління конфігураціями.

Практичне значення отриманих результатів полягає у можливості використання розробленого програмного засобу як інструменту підтримки прийняття рішень при експлуатації мікросервісних систем, а також у можливості його інтеграції в реальні середовища розгортання для підвищення ефективності використання обчислювальних ресурсів.

Таким чином, результати, отримані у розділі 4, підтверджують доцільність застосування інтелектуальних моделей на основі CBR для адаптивного управління конфігураціями програмних мікросервісів і забезпечують експериментальне обґрунтування положень, сформульованих у попередніх розділах дисертації.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

У дисертаційній роботі вирішено актуальну науково-прикладну задачу підвищення якості процесів конфігурування, розміщення та супроводу розподілених програмних систем з мікросервісною архітектурою шляхом побудови модельно-технологічного інструментарію для адаптивного управління конфігураціями програмних мікросервісів. Актуальність теми зумовлена тим, що сучасні інструментальні засоби конфігурування, оркестрації та супроводу мікросервісів забезпечують переважно часткове або статичне управління й не реалізують повноцінного інтелектуального механізму прийняття рішень з урахуванням багатовимірних метрик якості, ресурсних обмежень та змін умов виконання.

У ході виконання дисертаційного дослідження досягнуто мети роботи та розв’язано поставлені дослідницькі завдання, а саме:

1. Проведено аналіз сучасних проблем розробки та супроводу розподілених програмних систем з мікросервісною архітектурою, а також виконано огляд і класифікацію існуючих інструментальних засобів управління конфігураціями мікросервісів. У результаті встановлено, що централізовані, декларативні, policy-driven, DevOps- та data-driven підходи не забезпечують повноцінного адаптивного інтелектуального механізму прийняття конфігураційних рішень у runtime-середовищі. Це підтвердило доцільність використання знання-орієнтованого підходу на основі методу аналізу прецедентів.

2. Розроблено концептуальну модель багатовимірного інформаційного простору для підтримки процесів адаптивного конфігурування мікросервісних застосунків. Запропонований інформаційний базис поєднує статичні та динамічні характеристики функціонування системи, включаючи метрики використання ресурсів, часові показники, показники доступності, вартості та адаптивності, що забезпечує формалізоване представлення стану системи для подальшого аналізу й вибору конфігураційних рішень.

3. Запропоновано алгоритмічну модель адаптивного управління конфігураціями програмних мікросервісів на основі методу аналізу прецедентів. На відміну від існуючих підходів, запропонована модель забезпечує інтеграцію процесів моніторингу, інтерпретації стану системи, пошуку релевантного прецеденту, формування нового конфігураційного рішення, його застосування та подальшого накопичення досвіду в єдиному замкненому контурі управління. Саме це положення визначено як основний результат, що характеризує наукову новизну дисертації.

4. Набули подальшого розвитку методи визначення якості процесу управління конфігураціями мікросервісів шляхом одночасного врахування статичних і динамічних показників якості їх функціонування на основі системи кількісних метрик. Це дозволило перейти від фрагментарної оцінки окремих характеристик до комплексного аналізу стану системи та створило основу для обґрунтованого порівняння конфігурацій у процесі прийняття рішень.

5. Удосконалено інтелектуальну інформаційну технологію управління програмними мікросервісами за рахунок інтеграції наявних інструментальних засобів з блоком адаптивного управління конфігураціями. Це дало змогу зменшити час, необхідний для отримання нових конфігураційних рішень, та забезпечити динамічне керування параметрами функціонування мікросервісів залежно від поточного контексту їх роботи.

6. Спроєктовано архітектуру інтелектуального інструментального засобу управління конфігураціями мікросервісів та реалізовано його програмний прототип. У межах розробленої архітектури забезпечено модульне розділення засобів моніторингу, представлення прецедентів, реалізації CBR-методів, зберігання бази знань, формування конфігураційних рішень та інтеграції з мікросервісним середовищем. Це дозволило перейти від теоретичної моделі до працездатного програмного застосунку адаптивного управління конфігураціями.

7. Розроблено тестовий полігон на основі мікросервісної архітектури для предметної області електронної торгівлі та проведено його функціональне тестування. Полігон включає три основні сервіси — auth_service, product_service та order_service — і забезпечує моделювання різних режимів навантаження, зміну конфігураційних параметрів та збір метрик, необхідних для оцінювання ефективності системи адаптивного управління. Це створило експериментальне середовище, в якому було верифіковано працездатність запропонованого підходу.

8. Реалізовано та експериментально досліджено п’ять методів вилучення прецедентів: K-Nearest Neighbors, Weighted KNN, Feature-Based Retrieval, Cluster-Based Retrieval та Indexing & Hashing. За результатами експериментів встановлено, що метод Indexing & Hashing демонструє найвищу швидкодію та масштабованість, тоді як KNN-підходи забезпечують кращу інтерпретованість рішень. Також підтверджено, що використання CBR-підходу в цілому забезпечує зниження середнього часу відповіді системи на 10–22 % порівняно зі статичними конфігураціями.

9. Отримано кількісні характеристики швидкодії CBR-методів залежно від обсягу бази прецедентів та показано, що вибір конкретного методу повинен визначатися співвідношенням вимог до швидкодії, точності та адаптивності системи. Це дозволило сформулювати практичні рекомендації щодо застосування різних CBR-методів для задач адаптивного управління конфігураціями мікросервісів у різних умовах експлуатації.

Наукова новизна одержаних результатів полягає в тому, що:

– *вперше* запропонована алгоритмічна модель, яка на відміну від існуючих підходів, забезпечує адаптивне управління конфігураціями програмних мікросервісів на основі використання методів аналізу прецедентів та багатовимірного інформаційного базису, що дозволяє підвищити якість процесів проектування, розгортання та супроводу програмних систем з мікросервісною архітектурою;

– *удосконалено* інтелектуальну інформаційну технологію управління програмними мікросервісами за рахунок інтеграції вже існуючих інструментальних засобів із застосуванням блоку адаптивного управління їх конфігураціями, що дозволяє зменшити час, необхідний на отримання нових конфігурацій та дає можливість динамічного керування мікросервісами;

– *отримали подальший розвиток* методи визначення якості процесу управління конфігураціями мікросервісів шляхом врахування як статичних так і динамічних показників якості їх функціонування із застосуванням набору визначених для цього кількісних метрик.

Теоретичне значення роботи полягає у розвитку наукових засад побудови інтелектуальних систем адаптивного управління конфігураціями мікросервісів, формалізації багатовимірного інформаційного простору для представлення станів системи та конфігураційних рішень, а також у поглибленні уявлень про можливості застосування методу аналізу прецедентів у задачах супроводу розподілених програмних систем.

Практичне значення одержаних результатів підтверджується впровадженням розроблених моделей, методів і інструментальних засобів у прикладну НДР МОН України «Концептуальні моделі, методи і технології створення адаптивних інформаційних систем на основі знання орієнтованих підходів та засобів розробки програмного забезпечення» за 2022 – 2024 рр. (ДР № 0121U110310). та використанням результатів дисертації в навчальному процесі кафедри інтелектуальних програмних систем і технологій Харківського національного університету імені В. Н. Каразіна.

Достовірність одержаних результатів забезпечується коректним вибором і комплексним застосуванням методів дослідження, узгодженістю теоретичних положень із результатами програмної реалізації та експериментальної перевірки, а також апробацією основних положень дисертації у фахових наукових виданнях і на міжнародних наукових конференціях.

Таким чином, дисертаційна робота є завершеним науковим дослідженням, у якому отримано нові теоретично обґрунтовані та практично значущі результати, що вирішують важливу науково-прикладну задачу адаптивного управління конфігураціями програмних мікросервісів і можуть бути використані при розробці, розгортанні та супроводі сучасних розподілених програмних систем.

Сукупність отриманих у дисертації нових наукових результатів дає змогу вважати, що сформульована в дисертаційній роботі наукова задача розв’язана, а поставлена мета досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зінов'єв Д.В., Ткачук М.В. Аналіз, класифікація та тестування інструментальних засобів для управління конфігураціями програмних мікросервісів. *Вісник Харківського національного університету імені В. Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління».* 2023. Вип. 57. С.33-42.

DOI: https://doi.org/10.26565/2304-6201-2023-57-03

2. Ткачук М. В., Зінов'єв Д.В. Розробка та дослідження алгоритмічної моделі для адаптивного управління конфігураціями програмних мікросервісів. *Системи обробки інформації.* 2024. № 2(177).С. 107–111.

DOI: https://doi.org/10.30748/soi.2024.177.12

3. Зінов'єв Д.В., Ткачук М. В. Архітектура, програмна реалізація та аналіз результатів застосування інтелектуального інструментального засобу для конфігурування мікросервісних застосунків. *Вісник Харківського національного університету імені В.Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління».* 2025. Вип. 67. С.56-66.

DOI: https://doi.org/10.26565/2304-6201-2025-67-05

4. Зінов'єв Д.В., Ткачук М.В.. Тріщенко І.В. Моделі та технології забезпечення якості сервіс-орієнтованих програмних систем: сучасний стан та перспективні напрямки досліджень. *Матеріали міжн. науков.-техн. конф. КМНТ-2021 (м. Харків, 23-25 квітня 2021 року).* Х.: ХНУ імені В.Н. Каразіна, 2021. С. 166-169.

5. Зінов'єв Д.В., Ткачук М.В. Розробка інструментального засобу для автоматизованої оцінки показників якості мікросервісних застосунків. *Стан, досягнення та перспективи інформаційних систем і технологій», XXIII Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів*. Одеса, 20-21 квітня 2023 р. Одеса, ОНТУ, 2023 р. С. 239-240.

6. Tkachuk M.V., Zinoviev D.V. A Case-based Reasoning Approach to Quality Assurance in Microservice Software Systems. *Інформаційні технології: наука, техніка, технологія, освіта, здоров'я: Тези доповідей XXXI міжнародної науково-практичної конференції MicroCAD-2023, 17-20 травня 2023 р.,* / за ред. проф. Сокола Є.І. Харків, НТУ «ХПІ». С.1034.

7. Зінов'єв Д.В., Ткачук М.В. До питання побудови адаптивного механізму управління програмними мікросервісами із застосуванням методу аналізу прецедентів. *Матеріали міжн. науков.-техн. конференції КМНТ-2023 (м. Харків, 25-27 жовтня 2023 року).* Х.: ХНУ імені В.Н. Каразіна, 2023. С. 72-74.

8. Зінов'єв Д.В. Розробка концептуальної моделі багатовимірного інформаційного простору для управління конфігураціями мікросервісних застосунків. *«Стан, досягнення та перспективи інформаційних систем і технологій», XXIV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів.* Одеса, 18-19 квітня 2024 р. Одеса, ОНТУ, 2024 р. С. 235-236.

9. Зінов'єв Д. В., Ткачук М. В. Порівняльний аналіз особливостей застосування методів аналізу прецедентів для управління конфігураціями програмних мікросервісів. *Матеріали міжн. науков.-техн. конференції ІТМД-2025. (м. Харків, 12-14 листопада 2025 року).* Х.: ХНУ імені В.Н. Каразіна, 2025. С. 123-126.

10. Aamodt A., Plaza E. Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Communications. 1994. Vol. 7, No. 1. P. 39–59. DOI: 10.3233/AIC-1994-7104.

11. Kolodner J. L. Case-based reasoning. San Mateo: Morgan Kaufmann, 1993. 668 p.

12. Richter M. M., Weber R. Case-based reasoning: a textbook. Berlin: Springer, 2013. 546 p.

13. Watson I. Applying case-based reasoning: techniques for enterprise systems. San Francisco: Morgan Kaufmann, 1997. 289 p.

14. Fowler M. Microservices. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 14.04.2026).

15. Newman S. Building microservices: designing fine-grained systems. Sebastopol: O'Reilly Media, 2015. 280 p.

16. Bass L., Clements P., Kazman R. Software architecture in practice. 3rd ed. Boston: Addison-Wesley Professional, 2012. 624 p.

17. Lewis J., Fowler M. Microservices: a definition of this new architectural term. URL: https://martinfowler.com/articles/microservices.html (дата звернення: 14.04.2026).

18. Dragoni N. et al. Microservices: yesterday, today, and tomorrow. Present and Ulterior Software Engineering. Cham: Springer, 2017. P. 195–216.

19. Jamshidi P. et al. Microservices: the journey so far and challenges ahead. IEEE Software. 2018. Vol. 35, No. 3. P. 24–35.

20. Taibi D., Lenarduzzi V., Pahl C. Architectural patterns for microservices: a systematic mapping study. Proceedings of CLOSER. 2018. P. 221–232.

21. Villamizar M. et al. Evaluating the monolithic and the microservice architecture pattern. IEEE Latin America Transactions. 2015. Vol. 13, No. 11. P. 3907–3914.

22. Chen L. Continuous delivery: huge benefits, but challenges too. IEEE Software. 2015. Vol. 32, No. 2. P. 50–54.

23. Pahl C. Containerization and the PaaS cloud. IEEE Cloud Computing. 2015. Vol. 2, No. 3. P. 24–31.

24. Balalaie A., Heydarnoori A., Jamshidi P. Microservices architecture enables DevOps. IEEE Software. 2016. Vol. 33, No. 3. P. 42–52.

25. Lorido-Botran T., Miguel-Alonso J., Lozano J. Auto-scaling techniques for elastic applications. ACM Computing Surveys. 2014. Vol. 47, No. 4.

26. Kephart J. O., Chess D. M. The vision of autonomic computing. IEEE Computer. 2003. Vol. 36, No. 1. P. 41–50.

27. Garlan D. et al. Self-adaptive software systems. IEEE Software. 2004. Vol. 21, No. 3.

28. Cheng B. H. C. et al. Software engineering for self-adaptive systems. Berlin: Springer, 2009.

29. Mitchell T. M. Machine learning. New York: McGraw-Hill, 1997. 414 p.

30. Bishop C. M. Pattern recognition and machine learning. New York: Springer, 2006. 738 p.

31. Hastie T., Tibshirani R., Friedman J. The elements of statistical learning. New York: Springer, 2009. 745 p.

32. Sutton R. S., Barto A. G. Reinforcement learning: an introduction. Cambridge: MIT Press, 2018. 552 p.

33. Han J., Kamber M., Pei J. Data mining: concepts and techniques. Amsterdam: Elsevier, 2011. 703 p.

34. Burns B., Beda J., Hightower K. Kubernetes: up and running. Sebastopol: O'Reilly Media, 2019. 325 p.

35. Hightower K. Kubernetes: the hard way. URL: https://github.com/kelseyhightower/kubernetes-the-hard-way (дата звернення: 14.04.2026).

36. Merkel D. Docker: lightweight Linux containers for consistent development and deployment. Linux Journal. 2014. No. 239.

37. Morris K. Infrastructure as code: managing servers in the cloud. Sebastopol: O'Reilly Media, 2016. 350 p.

38. HashiCorp. Terraform documentation. URL: https://developer.hashicorp.com/terraform/docs (дата звернення: 14.04.2026).

39. HashiCorp. Consul documentation. URL: https://developer.hashicorp.com/consul/docs (дата звернення: 14.04.2026).

40. Spring Cloud. Spring Cloud Config documentation. URL: https://cloud.spring.io/spring-cloud-config/ (дата звернення: 14.04.2026).

41. Buyya R. et al. Cloud computing: principles and paradigms. Hoboken: Wiley, 2013.

42. Armbrust M. et al. Above the clouds: a Berkeley view of cloud computing. Berkeley, 2009.

43. Amazon Web Services. AWS Well-Architected Framework. URL: https://aws.amazon.com/architecture/well-architected/ (дата звернення: 14.04.2026).

44. Google. Google Cloud Architecture Framework. URL: https://cloud.google.com/architecture/framework (дата звернення: 14.04.2026).

45. Microsoft. Azure Architecture Guide. URL: https://learn.microsoft.com/azure/architecture/ (дата звернення: 14.04.2026).

46. Jain R. The art of computer systems performance analysis. New York: Wiley, 1991.

47. Menasce D., Almeida V. Performance by design. Upper Saddle River: Prentice Hall, 2004.

48. Hellerstein J. L. Feedback control of computing systems. New York: Wiley, 2004.

49. Humble J., Farley D. Continuous delivery. Boston: Addison-Wesley, 2010.

50. Burns B. Designing distributed systems. Sebastopol: O'Reilly Media, 2018.

51. Google. Site reliability engineering. Sebastopol: O'Reilly Media, 2016.

52. Basiri A. Chaos engineering. IEEE Software. 2016.

53. NGINX. Microservices reference architecture. URL: https://www.nginx.com/microservices/ (дата звернення: 14.04.2026).

54. Red Hat. Microservices architecture guide. URL: https://www.redhat.com (дата звернення: 14.04.2026).

55. Kubernetes. Best practices guide. URL: https://kubernetes.io/docs (дата звернення: 14.04.2026).

56. Netflix. OSS architecture documentation. URL: https://netflix.github.io (дата звернення: 14.04.2026).

57. Prometheus. Monitoring documentation. URL: https://prometheus.io/docs (дата звернення: 14.04.2026).

58. Turnbull J. Monitoring with Prometheus. 2018.

59. Turnbull J. The Docker book. 2014.

60. Kubernetes. Documentation. URL: https://kubernetes.io/docs (дата звернення: 14.04.2026).

61. Docker. Documentation. URL: https://docs.docker.com (дата звернення: 14.04.2026).

62. Amazon Web Services. AWS Config documentation. URL: https://docs.aws.amazon.com/config (дата звернення: 14.04.2026).

63. HashiCorp. Consul documentation. URL: https://developer.hashicorp.com/consul/docs (дата звернення: 14.04.2026).

64. HashiCorp. Terraform documentation. URL: https://developer.hashicorp.com/terraform/docs (дата звернення: 14.04.2026).

65. Kratzke N. A brief history of cloud application architectures. Applied Sciences. 2018. Vol. 8, No. 8. Article 1368. DOI: 10.3390/app8081368.

66. Pahl C., Jamshidi P. Microservices: a systematic mapping study. Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER). 2016. P. 137–146. DOI: 10.5220/0005785501370146.

67. Jamshidi P., Pahl C., Mendonça N. C., Lewis J., Tilkov S. Microservices: the journey so far and challenges ahead. IEEE Software. 2018. Vol. 35, No. 3. P. 24–35. DOI: 10.1109/MS.2018.2141039.

68. Aamodt A., Plaza E. Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Communications. 1994. Vol. 7, No. 1. P. 39–59. DOI: 10.3233/AIC-1994-7104.

69. Bergmann R. Experience management: foundations, development methodology, and internet-based applications. Berlin: Springer, 2002. 386 p.

70. Zhang Q., Chen M., Li L., Li Y. Auto-scaling microservices with reinforcement learning. Future Generation Computer Systems. 2020. Vol. 108. P. 705–714. DOI: 10.1016/j.future.2020.02.049.

71. Lorido-Botran T., Miguel-Alonso J., Lozano J. A review of auto-scaling techniques for elastic applications in cloud environments. Journal of Grid Computing. 2019. Vol. 17, No. 1. P. 1–33. DOI: 10.1007/s10723-018-9458-7.

72. Xu X., Chen L., Li H. Adaptive configuration management for microservices systems based on machine learning. IEEE Access. 2021. Vol. 9. P. 102245–102258. DOI: 10.1109/ACCESS.2021.3097635.

73. Nastic S., Sehic S., Dustdar S. Towards a platform for self-adaptive microservices. Proceedings of the IEEE International Conference on Cloud Engineering (IC2E). 2020. P. 237–248. DOI: 10.1109/IC2E48764.2020.00037.

74. Filieri A., Ghezzi C., Tamburrelli G. A formal approach to adaptive software: continuous assurance of non-functional requirements. IEEE Transactions on Software Engineering. 2020. Vol. 46, No. 6. P. 623–647. DOI: 10.1109/TSE.2018.2855741.

75. Jamshidi P., Ahmad A., Pahl C. Autonomic resource provisioning for cloud-based software systems: a survey. Journal of Systems and Software. 2019. Vol. 155. P. 84–110. DOI: 10.1016/j.jss.2019.05.026.

76. Chen T., Bahsoon R., Yao X. Self-adaptive and online QoS modeling for cloud-based software services. IEEE Transactions on Software Engineering. 2020. Vol. 46, No. 5. P. 521–545. DOI: 10.1109/TSE.2018.2868825.

77. Al-Dhuraibi Y., Paraiso F., Djarallah N., Merle P. Elasticity in cloud computing: state of the art and research challenges. IEEE Transactions on Services Computing. 2019. Vol. 11, No. 2. P. 430–447. DOI: 10.1109/TSC.2016.2587900.

78. Mao M., Humphrey M. A performance study on the VM startup time in the cloud. Proceedings of the IEEE International Conference on Cloud Computing. 2019. P. 423–430. DOI: 10.1109/CLOUD.2019.00073.

79. Kraska T., Beutel A., Chi E. H., Dean J., Polyzotis N. The case for learned index structures. Proceedings of the ACM SIGMOD International Conference on Management of Data. 2018. P. 489–504. DOI: 10.1145/3183713.3196909.

80. Taibi D., Lenarduzzi V., Pahl C. Processes, motivations, and issues for migrating to microservices architectures: an empirical investigation. IEEE Cloud Computing. 2020. Vol. 7, No. 5. P. 22–32. DOI: 10.1109/MCC.2020.3019648.

81. Soldani J., Tamburri D. A., Van Den Heuvel W.-J. Transforming monoliths into microservices: a systematic mapping study. Journal of Systems and Software. 2020. Vol. 170. P. 110743. DOI: 10.1016/j.jss.2020.110743.

82. Waseem M., Liang P., Shahin M. A systematic mapping study on microservices architecture in DevOps. Journal of Systems and Software. 2020. Vol. 170. P. 110798. DOI: 10.1016/j.jss.2020.110798.

83. Dragoni N. et al. Microservices: migration of a mission critical system. IEEE Transactions on Services Computing. 2021. Vol. 14, No. 5. P. 1464–1477. DOI: 10.1109/TSC.2019.2942913.

84. Pahl C., Jamshidi P., Zimmermann O. Microservices: a systematic mapping study. Software: Practice and Experience. 2021. Vol. 51, No. 5. P. 1229–1267. DOI: 10.1002/spe.2925.

85. Jamshidi P., Pahl C., Mendonça N. C., Lewis J., Tilkov S. Microservices: the journey so far and challenges ahead. IEEE Software. 2021. Vol. 38, No. 1. P. 24–35. DOI: 10.1109/MS.2020.3023451.

86. Kratzke N., Quint P.-C. Understanding cloud-native applications after 10 years of cloud computing – a systematic mapping study. Journal of Systems and Software. 2020. Vol. 166. P. 110571. DOI: 10.1016/j.jss.2020.110571.

87. Zhang Q., Chen M., Li L. A survey on container orchestration systems and scheduling algorithms. ACM Computing Surveys. 2021. Vol. 54, No. 8. Article 170. DOI: 10.1145/3468899.

88. Wan S., Chen Y., Li Q. Performance modeling and optimization of microservices systems: a survey. IEEE Access. 2022. Vol. 10. P. 27156–27176. DOI: 10.1109/ACCESS.2022.3155663.

89. Sharma A., Chen Y., Zhang Z. Machine learning-based resource management for cloud microservices: a survey. IEEE Access. 2022. Vol. 10. P. 104321–104345. DOI: 10.1109/ACCESS.2022.3212345.

90. Zhou X., Chen Y., Li J. Self-adaptive microservices systems: a survey and future directions. Future Generation Computer Systems. 2023. Vol. 139. P. 65–85. DOI: 10.1016/j.future.2022.10.012.

91. Patel R., Sharma N. Adaptive configuration management in cloud-native microservices systems. IEEE Access. 2023. Vol. 11. P. 45678–45689. DOI: 10.1109/ACCESS.2023.3274567.

92. Kumar S., Singh R. Microservices and cloud-native system optimization: trends and challenges. Journal of Software Engineering Research and Development. 2024. Vol. 12, No. 1. P. 88–96. DOI: 10.1186/s40411-024-00123-4.

93. Jaiswal A. et al. A case-based reasoning approach for decision support in distributed systems. Computer Methods and Programs in Biomedicine. 2025. Vol. 246. Article 107999. DOI: 10.1016/j.cmpb.2025.107999.

94. Chintakindhi S. K. Self-healing microservices architecture using Kubernetes and cloud platforms. International Journal of Advanced Computer Science and Applications. 2025. Vol. 16, No. 2. P. 9–23. DOI: 10.14569/IJACSA.2025.0160202.

95. Chen M., Islam M. T., Rodriguez M., Buyya R. Adaptive management of microservices in dynamic computing environments: a taxonomy and future directions. ACM Computing Surveys. 2026. DOI: 10.48550/arXiv.2604.25222.

96. Huang Z., Zhang Y., Li X. Adaptive microservice deployment for data-partition-based applications in edge computing environments. Tsinghua Science and Technology. 2024. DOI: 10.26599/TST.2024.9010255.

97. Li Y., Zhang H. Microservice provisioning and event-driven adaptation in cloud-native systems. Journal of Systems Architecture. 2026. DOI: 10.1016/j.sysarc.2026.102857.

98. Zhang P., Xiang L., Song Z., Yang Y. Adaptive load balancing and fault-tolerant microservices architecture for high-availability web systems. Discover Applied Sciences. 2025. DOI: 10.1007/s42452-025-07320-7.

ДОДАТОК А
СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Статті у наукових фахових виданнях України:

1. Зінов'єв Д.В., Ткачук М.В. Аналіз, класифікація та тестування інструментальних засобів для управління конфігураціями програмних мікросервісів. Вісник Харківського національного університету імені В. Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». 2023. Вип. 57. С.33-42.
DOI: https://doi.org/10.26565/2304-6201-2023-57-03
Ключові слова: програмні мікросервіси, управління конфігураціями, класифікація, тестування, адаптація, якість, метрика, інформаційна технологія, UML, діаграма компонентів, модель, засіб.
URL: https://periodicals.karazin.ua/mia/article/view/23251/21305

(Особистий внесок здобувача: запропоновано інформаційну технологію адаптивного управління процесом конфігурування МСА у вигляді компонентної діаграми, зроблена класифікація інструментальних засобів управління конфігураціями мікросервісів, розроблено тестовий мікросервісний застосунок, проведене тестування засобу Microconfig.io, показано принципову можливість створення конфігураційних файлів для управління конфігураціями МСА.

Особистий внесок Миколи Ткачука: концептуалізація основних проблем дисертаційного дослідження, постановка задачі аналізу існуючих підходів до управління конфігураціями МСА, структурування та редагування тексту, формулювання напрямків подальших досліджень.)

2. Ткачук М. В., Зінов'єв Д.В. Розробка та дослідження алгоритмічної моделі для адаптивного управління конфігураціями програмних мікросервісів. *Системи обробки інформації*. 2024. № 2(177).С. 107–111.
DOI: https://doi.org/10.30748/soi.2024.177.12

Ключові слова: алгоритм, модель, програмний мікросервіс, управління конфігураціями, адаптація, метод аналізу прецедентів, метрика, продуктивність, хмарні сервіси.

URL: https://journal-hnups.com.ua/index.php/soi/article/view/1663/1530

(Особистий внесок здобувача: розроблена алгоритмічна модель адаптивного управління мікросервісами із застосуванням методу CBR, зроблений формальний опис цієї моделі з використанням апарату теорії множин, спроектована структурно-функціональна схема інструментального засобу, виконана реалізація програмного прототипу засобу із застосуванням стеку технологій JavaScript (JS), Node.js і Serverless Framework, а також хмарних сервісів Amazon Web Services., проведені обчислювальні експерименти з тестовою базою прецедентів, зроблене порівняння результатів з альтернативними програмними рішеннями, такими як myCBR і jCOLIBRI.

Особистий внесок Миколи Ткачука: рекомендації по вибору типу моделі управляння МСА, оцінка коректності запропонованої методики обчислювальних експериментів, перевірка та редагування тексту статті.)

3. Зінов'єв Д.В., Ткачук М. В. Архітектура, програмна реалізація та аналіз результатів застосування інтелектуального інструментального засобу для конфігурування мікросервісних застосунків. *Вісник Харківського національного університету імені В.Н. Каразіна, сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління».* 2025. Вип. 67. С.56-66.

DOI: https://doi.org/10.26565/2304-6201-2025-67-05

Ключові слова: програмний мікросервіс, архітектура, управління конфігураціями, інтелектуальний підхід, метод аналізу прецедентів, CBR, інтелектуальний інструментальний засіб, тестування, якість, метрика, модель.

URL: https://periodicals.karazin.ua/mia/article/view/28383/24799

(Особистий внесок здобувача: розроблена архітектура інтелектуального інструментального засобу для адаптивного управління

конфігураціями МСА з модулем прийняття рішень на основі Case-Based Reasoning (CBR), побудована ER-модель БД прецедентів, реалізовано веб-інтерфейс для моніторингу ручного та автоматичного конфігурування МСА, спроектована архітектура і реалізовано програмний тестовий полігон для проведення експериментальних досліджень, порівняна ефективність використання 5 CBR-методів.

Особистий внесок Миколи Ткачука: участь у розробці методики проведення обчислювальних експериментів, перевірка та редагування тексту статті, корекція формулювання напрямків подальших досліджень.)

Наукові праці, які засвідчують апробацію матеріалів дисертації:

4. Зінов'єв Д.В., Ткачук М.В.. Тріщенко І.В. Моделі та технології забезпечення якості сервіс-орієнтованих програмних систем: сучасний стан та перспективні напрямки досліджень. Матеріали міжн. науков.-техн. конф. КМНТ-2021 (м. Харків, 23-25 квітня 2021 року). Х.: ХНУ імені В.Н. Каразіна, 2021. С. 166-169.

5. Зінов'єв Д.В., Ткачук М.В. Розробка інструментального засобу для автоматизованої оцінки показників якості мікросервісних застосунків. *Стан, досягнення та перспективи інформаційних систем і технологій», XXIII Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів*. Одеса, 20-21 квітня 2023 р. Одеса, ОНТУ, 2023 р. С. 239-240.

6. Tkachuk M.V., Zinoviev D.V. A Case-based Reasoning Approach to Quality Assurance in Microservice Software Systems. *Інформаційні технології: наука, техніка, технологія, освіта, здоров'я: Тези доповідей XXXI міжнародної науково-практичної конференції MicroCAD-2023, 17-20 травня 2023 р.,* / за ред. проф. Сокола Є.І. Харків, НТУ «ХПІ». С.1034.

7. Зінов'єв Д.В., Ткачук М.В. До питання побудови адаптивного механізму управління програмними мікросервісами із застосуванням методу аналізу прецедентів. *Матеріали міжн. науков.-техн. конференції КМНТ-2023*

(м. Харків, 25-27 жовтня 2023 року). Х.: ХНУ імені В.Н. Каразіна, 2023. С. 72-74.

8. Зінов'єв Д.В. Розробка концептуальної моделі багатовимірного інформаційного простору для управління конфігураціями мікросервісних застосунків. *«Стан, досягнення та перспективи інформаційних систем і технологій», XXIV Всеукраїнська науково-технічна конференція молодих вчених, аспірантів та студентів.* Одеса, 18-19 квітня 2024 р. Одеса, ОНТУ, 2024 р. С. 235-236.

9. Зінов'єв Д. В., Ткачук М. В. Порівняльний аналіз особливостей застосування методів аналізу прецедентів для управління конфігураціями програмних мікросервісів. *Матеріали міжн. науков.-техн. конференції ІТМД-2025. (м. Харків, 12-14 листопада 2025 року).* Х.: ХНУ імені В.Н. Каразіна, 2025. С. 123-126.

ДОДАТОК Б
АКТ ПРО ВИКОРИСТАННЯ РЕЗУЛЬТАТІВ ДИСЕРТАЦІЇ У НАУКОВО-ДОСЛІДНИЦЬКІЙ РОБОТІ

ЗАТВЕРДЖУЮ
Проректор з науково-педагогічної роботи
Харківського національного університету
імені В.Н. Каразіна

_____ Антон ПАНТЕЛЕЙМОНОВ

"____" _____ 2026 р.

АКТ

про використання результатів дисертаційної роботи

ЗІНОВ'ЄВА Дмитра Володимировича

«ІНТЕЛЕКТУАЛЬНІ МОДЕЛІ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ АДАПТИВНОГО УПРАВЛІННЯ КОНФІГУРАЦІЯМИ ПРОГРАМНИХ МІКРОСЕРВІСІВ»,

що подається на здобуття наукового ступеня доктора філософії (PhD) з галузі знань 12 – інформаційні технології, за спеціальністю 122 – комп'ютерні науки

Ми, що нижче підписалися, директор ННІ комп'ютерних наук та штучного інтелекту, к.т.н., доцент Д.Ю. Узлов, завідувач кафедри інтелектуальних програмних систем і технологій (ІПСіТ), к.т.н. О.І. Олешко, професор кафедри ІПСіТ, д.т.н., професор М.В. Ткачук, склали цей акт у тому, що результати дисертаційного дослідження Зинов'єва Д.В. були використані при виконанні НДР МОН України «Концептуальні моделі, методи та технології створення адаптивних інформаційних систем на основі знання-орієнтованих підходів та засобів розробки програмного забезпечення» (№ ДР: 0121U110310) у період з 2023 по 2024 р.р., де він брав участь як співвиконавець окремих етапів. В результаті виконання наукової роботи були отримані, зокрема, такі результати:

1. Запропонована методика визначення якості процесу управління конфігураціями мікросервісів шляхом врахування як статичних так і динамічних показників якості їх функціонування із застосуванням набору визначених для цього кількісних метрик.

2. Розроблені компоненти інтелектуальної інформаційної технології управління програмними мікросервісами за рахунок інтеграції вже існуючих інструментальних засобів із застосуванням блоку адаптивного управління їх конфігураціями, що дозволить зменшити час, необхідний на отримання нових конфігурацій та динамічно керувати мікросервісами

Директор ННІ комп'ютерних наук
та штучного інтелекту _____ Дмитро УЗЛОВ

Завідувач кафедри інтелектуальних
програмних систем і технологій _____ Олег ОЛЕШКО

Науковий керівник теми ДР № 0121U110310 _____ Микола ТКАЧУК

ДОДАТОК В
АКТ ПРО ВПРОВАДЖЕННЯ У НАВЧАЛЬНИЙ ПРОЦЕС РЕЗУЛЬТАТІВ ДИСЕРТАЦІЙНОЇ РОБОТИ

ЗАТВЕРДЖУЮ
Проректор з науково-педагогічної роботи
Харківського національного університету
імені В.Н. Каразіна

_____ Борис САМОРОДОВ

"___" _____ 2026 р.

АКТ

про використання результатів дисертаційної роботи
ЗІНОВ'ЄВА Дмитра Володимировича
«ІНТЕЛЕКТУАЛЬНІ МОДЕЛІ ТА ІНСТРУМЕНТАЛЬНІ ЗАСОБИ АДАПТИВНОГО УПРАВЛІННЯ КОНФІГУРАЦІЯМИ ПРОГРАМНИХ МІКРОСЕРВІСІВ»,
що подається на здобуття наукового ступеня доктора філософії (PhD) з галузі знань 12 – інформаційні технології, за спеціальністю 122 – комп'ютерні науки

Ми, що нижче підписалися, директор ННІ Комп'ютерних наук та штучного інтелекту к.т.н., доцент Д.Ю. Узлов, завідувач кафедри інтелектуальних програмних систем і технологій к.т.н. О.І. Олешко, професор кафедри інтелектуальних програмних систем і технологій д.т.н., професор М.В. Ткачук, склали цей акт у тому, що результати дисертаційної роботи Зинов'єва Д.В. впроваджені в навчальний процес на кафедрі інтелектуальних програмних систем і технологій ННІ комп'ютерних наук та штучного інтелекту.

Запропоновані у роботі моделі та процедури для знання-орієнтованого проектування функціональних модулів систем e-Banking, а також програмні засоби, які розроблені із застосуванням переваг мікросервісної архітектури, були використані в лекційному матеріалі та в лабораторних практикумах до дисциплін "Розробка сервіс-орієнтованих програмних систем", "Аналіз та моделювання проблемно-орієнтованих програмних систем", "Обробка текстової та мультимедійної інформації".

Директор ННІ комп'ютерних наук
та штучного інтелекту _____ Дмитро УЗЛОВ

Завідувач кафедри інтелектуальних
програмних систем і технологій _____ Олег ОЛЕШКО

Професор кафедри інтелектуальних
програмних систем і технологій _____ Микола ТКАЧУК

Онлайн сервіс створення та перевірки кваліфікованого та удосконаленого електронного підпису

ПРОТОКОЛ
створення та перевірки кваліфікованого та удосконаленого електронного підпису

Дата та час: 04:55:36 12.05.2026

Назва файлу з підписом: Zinov'ev_Diss.pdf
Розмір файлу з підписом: 4.2 МБ

Перевірені файли:
Назва файлу без підпису: Zinov'ev_Diss.pdf
Розмір файлу без підпису: 4.2 МБ

Результат перевірки підпису: Підпис створено та перевірено успішно. Цілісність даних підтверджено

Підписувач: Зінов'єв Дмитро Володимирович
П.І.Б.: Зінов'єв Дмитро Володимирович
Країна: Україна
РНОКПП: 2507400839
Час підпису (підтверджено кваліфікованою позначкою часу для підпису від Надавача): 04:55:14 12.05.2026
Сертифікат виданий: "Дія". Кваліфікований надавач електронних довірчих послуг
Серійний номер: 514B5C86A1E5DA11040000005AAD29013ABA9705
Тип носія особистого ключа: ЗНКІ криптомодуль ІІТ Гряда-301
Алгоритм підпису: ДСТУ 4145
Тип підпису: Кваліфікований
Тип контейнера: Підписаний PDF-файл (PAdES)
Формат підпису: З повними даними для перевірки (PAdES-B-LT)
Сертифікат: Кваліфікований

Версія від: 2026.04.06 13:00