УДК 004.415.5
DOI https://doi.org/10.32689/maup.it.2025.3.16

Борис ПАНАСЮК
аспірант спеціальності «Інженерія програмного забезпечення»,
Вінницький національний технічний університет,
boris.panasyuk@gmail.com
***ORCID:** 0009-0007-2064-9121*

Наталя БАБЮК
кандидат технічних наук, доцент кафедри програмного забезпечення,
Вінницький національний технічний університет,
babiuk@vntu.edu.ua
***ORCID:** 0000-0003-0607-6340*

КОНТРАКТНО-ОРІЄНТОВАНИЙ ЦИФРОВИЙ ДВІЙНИК МІКРОСЕРВІСНОЇ СИСТЕМИ: МОДЕЛЬ, МЕТАМОДЕЛЬ, АРТЕФАКТИ OPENAPI/ASYNCAPI

Анотація. *Метою дослідження є створення контрактно-орієнтованого цифрового двійника мікросервісної системи, що базується на моделі та метамоделі взаємодії сервісів через їх API-контракти. Для досягнення мети використано підхід API-first: формальні специфікації сервісів (OpenAPI для REST API та AsyncAPI для асинхронних API) слугують артефактами, на основі яких автоматично побудовано модель цифрового двійника.*

Методологія. *Застосовано аналіз та узагальнення сучасних підходів до цифрових двійників, моделювання мікросервісної архітектури із використанням формальних описів інтерфейсів, а також виконано порівняльний аналіз з існуючими моделями цифрових двійників.*

Наукова новизна. *Запропоновано концепцію «контрактно-орієнтованого» цифрового двійника, що вперше фокусує цифрову модель системи на її API-контрактах, забезпечуючи автоматизоване отримання та актуалізацію двійника з артефактів OpenAPI/AsyncAPI, тим самим поєднуючи процес документування API з підтримкою віртуальної копії системи.*

Висновки. *Контрактно-орієнтований підхід дозволяє підтримувати цифровий двійник актуальним при еволюції мікросервісів, спрощує тестування сумісності сервісів і аналіз поведінки системи без впливу на продуктивне середовище. Запропонований підхід апробовано на прикладі спрощеної мікросервісної системи; результати підтверджують можливість автоматичного формування двійника та ефективність його використання для інтеграційного тестування нових версій сервісів. Отримані результати можуть бути впроваджені у практику DevOps для автоматизації регресійного тестування мікросервісів та контролю відповідності їх реалізації заявленим контрактам. В цілому, використання контрактно-орієнтованого двійника сприяє підвищенню якості та надійності мікросервісних програмних комплексів та скорочує час, необхідний на інтеграційне тестування.*

Ключові слова**:** *контракт-орієнтоване моделювання, інформаційна система, цифровий двійник, мікросервісна архітектура, API-контракт, моделювання, автоматизація.*

Borys PANASIUK, Natalia BABIUK. CONTRACT-ORIENTED DIGITAL TWIN OF A MICROSERVICE SYSTEM: MODEL, METAMODEL, OPENAPI/ASYNCAPI ARTIFACTS

Abstract. The study aims *to develop a contract-oriented digital twin of a microservice-based system, grounded in a model and meta-model of service interactions defined by their API contracts.*

Methodology. *The study adopts an API-first approach: formal service interface specifications (OpenAPI for REST APIs and AsyncAPI for event-driven APIs) are used as artifacts to automatically construct the digital twin model. Additionally, a comparative analysis with existing approaches was conducted.*

Scientific novelty. *The concept of a «contract-oriented» digital twin is proposed, focusing the virtual model of the system on its API contracts; this approach is novel in enabling automated generation and updating of the twin from OpenAPI/AsyncAPI artifacts. This effectively merges the API documentation process with the maintenance of a live system model.*

Conclusions. *The contract-oriented approach ensures the digital twin remains up-to-date as microservices evolve, and it simplifies compatibility testing and behavioral analysis of the system without impacting the production environment. The proposed approach was validated on a simplified microservice scenario; the results confirm the feasibility of automatic twin generation and its effectiveness for integration testing of new service versions. The outcomes can be applied in DevOps practice to automate regression testing of microservices and ensure that their implementations conform to specified contracts. Overall, using a contract-oriented twin helps improve the quality and reliability of microservice-based software systems and reduces the time required for integration testing. A simplified prototype of the digital twin was implemented to demonstrate the approach, which showed its viability in a realistic scenario.*

Key words: *contract-oriented modelling, information system, digital twin, microservice architecture, API contract, modelling, automation.*

Постановка проблеми. Мікросервісна архітектура стала широко розповсюдженою для побудови масштабованих розподілених програмних систем. В той же час вона ускладнює забезпечення надійності та узгодженості системи, оскільки велика кількість сервісів з чітко визначеними інтерфейсами постійно змінюється та взаємодіє між собою. Цифровий двійник – віртуальна модель реального об'єкта чи системи – давно використовується в промисловості для моніторингу та прогнозування стану фізичних активів [9]. Багато з цих можливостей можуть бути корисними і для програмних систем на основі мікросервісів. Проте пряме застосування концепції цифрового двійника, спочатку орієнтованої на фізичні системи, до програмної архітектури стикається з проблемою відсутності фізичних параметрів та необхідністю моделювати поведінку сервісів і їх взаємодію.

Традиційно, цифровий двійник визначається як віртуальне представлення фізичного об'єкта або процесу, що тісно пов'язане з реальним прототипом і синхронізується з ним у режимі, близькому до реального часу [4]. Розрізняють поняття цифрової моделі, цифрової тіні та власне цифрового двійника: цифрова модель – це точна віртуальна копія об'єкта без автоматичного зв'язку з ним; цифрова тінь має однобічний зв'язок (дані надходять від реального об'єкта до моделі); цифровий двійник же передбачає двонапрямний обмін даними між фізичним і віртуальним об'єктом [5]. У випадку програмної системи «фізичним» об'єктом є сам програмний сервіс або сукупність сервісів, а роль даних виконують, зокрема, повідомлення та виклики API, які можна перехоплювати й аналізувати.

Проблемою є відсутність методів і моделей, що дозволяють створити цифровий двійник саме для програмної мікросервісної системи та підтримувати його актуальність у контексті частих змін сервісів. Існуючі підходи до моніторингу і тестування мікросервісів (наприклад, системи централізованого логування, трасування, contract testing) лише частково реалізують концепцію цифрового двійника і, як правило, не інтегровані в єдину модель системи. Відтак постає завдання розробити підхід, який дозволить формувати віртуальну модель мікросервісної системи, що відображає її структуру та поведінку, на основі артефактів, які вже існують у процесі розробки – формальних описів API. Такий контрактно-орієнтований цифровий двійник має слугувати інструментом для моделювання взаємодії сервісів, тестування змін та забезпечення відповідності між реалізацією сервісів і їхніми контрактами.

Аналіз останніх досліджень і публікацій. Концепція цифрового двійника була вперше запропонована у сфері керування життєвим циклом продукту М. Грівзом і Дж. Вікерсом (NASA) та формалізована у 2014 році [4]. З того часу цифрові двійники набули широкого застосування в різних галузях: виробництві, енергетиці, транспорті тощо [9], де вони використовуються для віддаленого моніторингу стану обладнання, прогнозування несправностей та оптимізації роботи систем. Наприклад, детальний огляд технологій цифрових двійників в контексті IoT представлений в роботі Minerva et al. (2020) [7], де розглядаються технічні характеристики та архітектурні моделі реалізації двійників для промислових пристроїв. В огляді Rasheed et al. (2020) особлива увага приділяється проблемам моделювання при створенні цифрових двійників та підкреслюється важливість адекватних інформаційних моделей [10].

Останніми роками з'явилися роботи, присвячені застосуванню концепції цифрових двійників до хмарних та програмних систем. Зокрема, у роботі Raghunandan et al. (2023) запропоновано цифровий двійник для архітектури мікросервісів у Kubernetes, який відслідковує використання ресурсів кластера в реальному часі та дозволяє виявляти аномалії [9]. Інший підхід – KubeKlone (Bhardwaj, Benson, 2022) – являє собою програмний симулятор (digital twin) для хмарно-периферійних мікросервісних застосувань, призначений для експериментів з алгоритмами автоматичного керування інфраструктурою на основі AI [3]. Bellavista et al. (2024) описують масштабовану мікросервісну платформу цифрових двійників для сценаріїв «хмара-край», яка використовує безсерверні обчислення для гнучкого розгортання компонентів двійника [2]. Таким чином, тенденція останніх досліджень – перехід від суто фізичних систем до програмно-орієнтованих двійників, зокрема мікросервісних архітектур.

Окремо варто відзначити роботи, де сама архітектура цифрового двійника реалізована з використанням мікросервісів. Так, Loboda, Starovit (2022) запропонували програмну архітектуру цифрового двійника для об'єкта «Новий безпечний конфайнмент» (укриття на ЧАЕС), в якій функціональні модулі двійника (візуалізація, прогнозування, аналіз) реалізовані як окремі мікросервіси, що взаємодіють через захищені протоколи [6]. Це підтверджує доцільність принципів мікросервісності у побудові складних цифрових двійників.

Аналіз публікацій показує, що хоча елементи концепції цифрового двійника вже застосовуються для мікросервісних систем, відсутні рішення, які б явно використовували контрактно-орієнтований підхід. Більшість існуючих робіт фокусуються або на моніторингу ресурсів і станів (як у Raghunandan et al.), або на засобах симуляції навантаження (як у KubeKlone), або на загальних питаннях архітектури. Натомість наш підхід пропонує будувати модель двійника безпосередньо зі специфікацій

ISSN 2786-5460 (Print), ISSN 2786-5479 (Online)

інтерфейсів сервісів (контрактів API), що вже наявні. Такий підхід відповідає сучасній практиці контрактно-орієнтованої (API-first) розробки мікросервісів [8], проте досі не був формалізований у вигляді цілісної концепції програмного цифрового двійника.

Мета статті. Метою цієї роботи є розробка моделі та метамоделі контрактно-орієнтованого цифрового двійника мікросервісної системи, а також методичних засад автоматизованого отримання відповідних артефактів (формальних описів інтерфейсів OpenAPI/AsyncAPI) і використання їх для побудови та підтримки цифрового двійника. Іншими словами, ставиться завдання формалізувати структуру цифрового двійника, що ґрунтується на контрактах сервісів, та показати, як на основі цієї структури можна автоматично генерувати допоміжні артефакти: симуляції сервісів, тестові сценарії, моніторингові компоненти тощо.

Для досягнення зазначеної мети потрібно вирішити такі підзадачі:

– Розробка метамоделі – описати основні сутності мікросервісної системи та їх взаємозв'язки в контексті API-контрактів (як REST, так і асинхронних).

– Формування моделі двійника – на основі метамоделі створити узагальнену модель цифрового двійника, яка включає представлення кожного сервісу та їхніх взаємодій.

– Автоматизація побудови – визначити процес автоматизованого отримання моделі двійника з артефактів OpenAPI/AsyncAPI та механізми синхронізації моделі з реальними сервісами (наприклад, через перехоплення викликів або оновлення при зміні версій API).

– Оцінка переваг – порівняти запропонований підхід із традиційними методами (централізований моніторинг, інтеграційне тестування тощо) щодо забезпечення цілісності системи та прискорення циклу розробки.

Модель та метамодель контрактно-орієнтованого цифрового двійника. Метамодель цифрового двійника мікросервісної системи визначає ключові елементи, необхідні для опису структури та поведінки системи на рівні її контрактів. На (рис. 1) (умовно) зображено основні класи метамоделі та зв'язки між ними. Центральним елементом є клас Microservice (Мікросервіс), який характеризується набором контрактів APIContract. Кожен APIContract може бути двох підтипів:

– RESTContract – описує синхронний веб-сервіс (HTTP REST API) з набором ресурсів та операцій.

– EventContract – описує асинхронний сервіс (напр., обмін повідомленнями через черги або топіки) з визначенням каналів публікації/підписки.

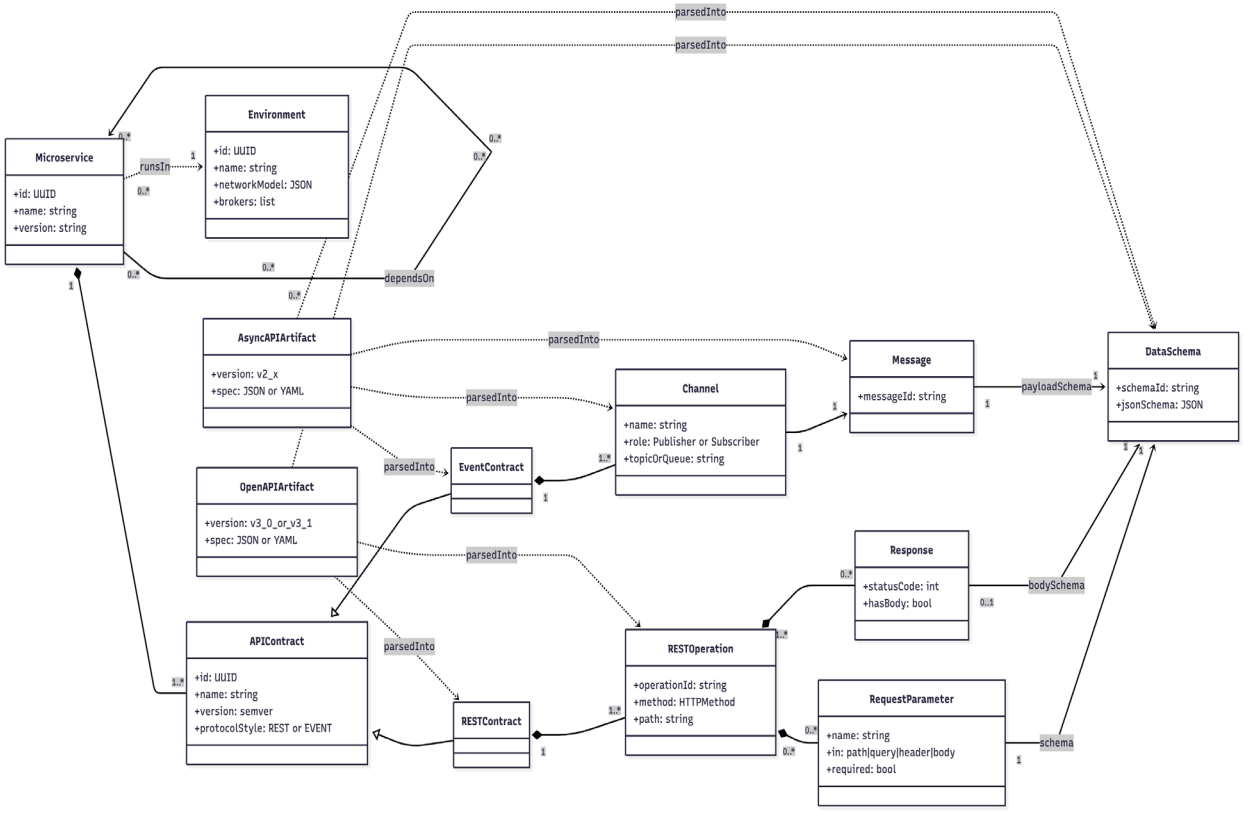

Рис. 1. Метамодель контрактно-орієнтованого цифрового двійника мікросервісної системи

Клас RESTContract містить колекцію об'єктів типу RESTOperation – кожна операція відповідає одному кінцевому endpoint REST API з конкретним HTTP-методом (GET, POST тощо) та шляхом. Для RESTOperation фіксується:

– набір RequestParameter (параметри запиту: path, query, header, body), кожен із зазначенням типу даних;

– структура Response(ів) – для різних кодів відповіді визначаються типи даних тіла відповіді (або вказується, що тіло відсутнє).

Аналогічно, клас EventContract містить колекцію Channel, кожен з яких має дві можливі ролі: Publisher (генерує події) або Subscriber (споживає події). Кожен Channel характеризує тип повідомлення (Message) з певною схемою даних.

Для узагальнення вводиться клас DataSchema (Схема даних), який описує структуру даних (об'єктів, масивів, примітивів) у форматі, сумісному з JSON Schema. Об'єкти RequestParameter, Response та Message містять посилання на відповідний DataSchema, що визначає формат даних.

Таким чином, APIContract (незалежно від типу REST чи Event) складається з описів операцій/каналів та пов'язаних із ними схем даних. Ці описи безпосередньо відповідають структурам стандартів OpenAPI та AsyncAPI, що забезпечує можливість автоматичного перетворення. Зокрема, артефакт OpenAPI версії 3.0/3.1 може бути розпарсований [8] у набір об'єктів RESTContract, RESTOperation, DataSchema, а артефакт AsyncAPI – у відповідні об'єкти EventContract, Channel, Message і DataSchema [1].

На рівні системи метамодель передбачає також відношення між мікросервісами залежності за використанням чужих API. Для цього вводиться зв'язок типу Microservice → Microservice («depends on»), який означає, що один сервіс виступає клієнтом API іншого. Цей зв'язок може бути заданий вручну (на основі знання архітектури), або виявлений автоматично шляхом аналізу конфігурацій викликів (наприклад, шляхом пошуку URL звернень у коді). У моделі така залежність використовується для побудови графа взаємодії сервісів.

Загальна модель цифрового двійника мікросервісної системи складається з:

– сукупності об'єктів Microservice, кожен з яких має власні контракти API (RESTContract і/або EventContract);

– визначених залежностей між цими Microservice;

– спільного компонента Environment (середовище), що моделює середовище виконання – мережеві умови, брокери повідомлень тощо (за потреби модель може бути розширена на рівень інфраструктури).

Автоматизація побудови та оновлення двійника. Однією з ключових переваг контрактно-орієнтованого підходу є можливість автоматизувати процес створення цифрового двійника. У типовому циклі розробки мікросервісів прийнято підтримувати актуальну документацію API у вигляді специфікацій OpenAPI/AsyncAPI (файл формату YAML/JSON) – такий підхід забезпечує єдине джерело правди про інтерфейси сервісів. Отже, побудова двійника може бути реалізована як конвеєр з кількох етапів:

1. Збирання контрактів. Із репозиторіїв коду або з централізованого реєстру збираються актуальні файли OpenAPI (для кожного REST-сервісу) та AsyncAPI (для сервісів, що спілкуються через повідомлення). Кожен файл контракту описує структуру API окремого сервісу.

2. Парсинг і трансформація. Кожна специфікація парсується за допомогою відповідних бібліотек (наприклад, Swagger-parser для OpenAPI або AsyncAPI parser) у внутрішню об'єктну модель. На цьому етапі забезпечується відповідність елементів: визначення шляхів і методів переводяться в об'єкти RESTOperation, специфікації схем даних зі складу контракту – у об'єкти DataSchema тощо. Аналогічно, AsyncAPI-специфікація перетворюється на об'єкти Channel та Message зі зв'язком до DataSchema.

3. Формування моделі системи. На основі метамоделі створюються екземпляри Microservice з заповненням їх APIContract відповідно до отриманих даних. Якщо контракти містять посилання на зовнішні сервіси чи топіки (наприклад, AsyncAPI може декларувати підписку на топік, який публікується іншим сервісом), ці відомості використовуються для встановлення зв'язків «depends on» між Microservice. За відсутності явних вказівок залежності можуть бути визначені експертно або шляхом аналізу конфігурацій (не є частиною контрактів, але можуть бути додані вручну для повноти моделі).

4. Розгортання цифрового двійника. Отримана модель може бути застосована кількома способами. По-перше, на її основі генеруються mock-сервіси – спрощені реалізації сервісів, що відповідають їх контрактам. Для цього використовуються наявні інструменти генерації коду: наприклад, на основі OpenAPI можна згенерувати каркас веб-сервера, який повертає фіксовані відповіді (або echo-відповіді) відповідно до контракту; на основі AsyncAPI – налаштувати тестовий брокер, що імітує необхідні канали та повідомлення. По-друге, модель слугує схемою для налаштування моніторингу: для кожного зафіксованого в моделі endpoint автоматично формується тестовий запит та перевірка

очікуваного формату відповіді (т.зв. контрактне тестування). Таким чином цифровий двійник виконує роль середовища віртуального тестування – запити до двійника обробляються mock-сервісами згідно з контрактами і дозволяють перевіряти інтеграцію без підняття реальних сервісів.

Автоматичне оновлення двійника відбувається при зміні контрактів. Якщо розробник змінює OpenAPI- або AsyncAPI-специфікацію сервісу (наприклад, додає новий метод або модифікує структуру даних), конвеєр перетворення запускається повторно та оновлює відповідні частини моделі двійника. Таким чином цифровий двійник завжди відображає актуальний стан інтерфейсів системи. Для порівняння, у традиційних підходах часто страждає актуальність документації API (вона може «гнити» – відставати від реалізації); у нашому ж підході ця проблема мінімізується, оскільки двійник безпосередньо генерується з тих самих артефактів, що й код сервісів (у контракт-орієнтованій розробці контракт створюється перед написанням коду).

Приклад та порівняння з існуючими рішеннями. Розглянемо спрощений приклад. Система складається з трьох мікросервісів: Order Service, Payment Service та Notification Service. Order Service надає REST API для створення замовлень (POST /orders) та отримання їх статусу (GET /orders/{id}); Payment Service надає REST API для опрацювання платежів (POST /payments); Notification Service підписується на подію OrderCreated (через брокер повідомлень, напр. RabbitMQ) та надсилає email з підтвердженням замовлення. Для цих сервісів існують специфікації API: OpenAPI для перших двох та AsyncAPI для останнього.

На основі цих специфікацій наш підхід формує модель двійника. Створюються три об'єкти Microservice – по одному на кожен сервіс – і для кожного заповнюється контракт: у Order Service це RESTContract з двома операціями (POST /orders, GET /orders/{id}) та відповідними схемами даних (наприклад, OrderRequest, OrderResponse); у Payment Service – RESTContract з однією операцією (POST /payments) та схемою PaymentRequest; у Notification Service – EventContract з каналом order.created (роль Subscriber) та повідомленням OrderCreatedMessage (містить, наприклад, ID замовлення та email клієнта). У модель додаються залежності: Notification Service залежить від Order Service (споживає його події), Order Service – від Payment Service (може викликати його API при створенні замовлення).

Для тестування такої системи традиційно потрібне розгортання всіх трьох сервісів у тестовому середовищі або створення заглушок вручну. Контрактно-орієнтований двійник дозволяє здійснити це автоматично. На основі OpenAPI для Order і Payment Service генеруються mock-сервери, що реалізують відповідні endpoints та повертають типові відповіді (наприклад, за допомогою Swagger Codegen). На основі AsyncAPI налаштовується тестовий брокер, який відправляє і приймає повідомлення OrderCreated. Далі інтегруємо ці компоненти: надсилаємо HTTP-запит POST /orders на mock Order Service – той повертає попередньо визначену відповідь (наприклад, 201 Created з деяким orderId). У реальній системі Order Service після створення замовлення опублікував би подію; у тестовому середовищі ми імітуємо це вручну, відправивши повідомлення OrderCreated на Notification Service (у нашому двійнику). Перевіряємо, що mock Notification Service отримав повідомлення і згенерував, скажімо, відповідний лог або виклик (в реальній реалізації – email). Попри спрощеність, такий експеримент дозволяє на ранніх етапах виявити невідповідності у форматі даних або послідовності викликів між сервісами.

Порівняємо підхід з існуючими рішеннями. Raghunandan et al. (2023) та деякі інші роботи фокусуються передусім на симуляції низькорівневих аспектів (навантаження, використання ресурсів) для моніторингу і оптимізації, тоді як наш підхід спрямований на перевірку функціональної сумісності – відповідності контрактам і інтеграції. Він не заміняє моделі типу KubeKlone (що емулюють продуктивність системи), а доповнює їх, забезпечуючи автоматизовану валідацію правильності інтерфейсів та взаємодій. Фактично, контрактно-орієнтований двійник поєднує ідеї контрактного тестування та середовища симуляції в межах єдиної моделі.

Висновки. У роботі представлено підхід до побудови цифрового двійника мікросервісної системи на основі її контрактів (специфікацій API). Розроблено формальну метамодель, яка описує мікросервіс, його RESTful і подієві інтерфейси, а також взаємозв'язки між сервісами. Показано, що використання існуючих артефактів OpenAPI та AsyncAPI дає змогу автоматизувати процес створення моделі двійника та генерування на її основі допоміжних компонентів (mock-сервісів, емуляторів повідомлень, тестових сценаріїв). Контрактно-орієнтований цифровий двійник підтримує актуальність моделі системи при внесенні змін до API, оскільки оновлення специфікацій автоматично відображається на моделі двійника.

Наукова новизна роботи полягає в поєднанні концепції цифрового двійника з методологією контрактно-орієнтованої розробки програмного забезпечення. Це дозволило створити інструмент,

який не лише відображає структуру системи, але й інтегрується у процес розробки, слугуючи «живою» документацією та середовищем для експериментів. Практична цінність підходу полягає у спрощенні інтеграційного тестування: команди розробників можуть перевіряти сумісність сервісів із двійником до їхнього розгортання в спільному середовищі, що знижує ризики збоїв при інтеграції.

Перспективи подальших досліджень включають розширення можливостей цифрового двійника для моделювання динаміки системи під навантаженням (шляхом інтеграції з інструментами на кшталт KubeKlone) та реалізацію двостороннього зв'язку з реальними сервісами (тобто перехід від цифрової тіні до повноцінного цифрового двійника програмної системи). Доцільним є також оцінювання ефективності запропонованого підходу на реальних кейсах і розробка рекомендацій щодо впровадження контрактно-орієнтованих двійників у практику DevOps.

Список використаних джерел:

1. AsyncAPI Initiative. AsyncAPI Specification (Version 2.3.0), 2022. URL: https://www.asyncapi.com (дата звернення: 24.09.2025).

2. Bellavista P., Bicocchi N., Fogli M., Giannelli C., Mamei M., Picone M. Exploiting microservices and serverless for Digital Twins in the cloud-to-edge continuum. *Future Generation Computer Systems*, 2024, pp. 275–287. DOI: 10.1016/j.future.2024.03.052.

3. Bhardwaj A., Benson T.A. KubeKlone: A Digital Twin for Simulating Edge and Cloud Microservices. In: Proc. 6th Asia-Pacific Workshop on Networking (APNet **2022),** *ACM*, 2022, 7 p. DOI: 10.1145/3542637.3542642.

4. Grieves M. Digital Twin: Manufacturing Excellence through Virtual Factory Replication. White Paper, 2014, 7 p.

5. Kritzinger W., Karner M., Traar G., Henjes J., Sihn W. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 2018, 51(11), pp. 1016–1022. DOI: 10.1016/j.ifacol.2018.08.474.

6. Лобода П. П., Старовіт І. С. Розробка архітектури програмного забезпечення прогнозування і управління термогазодинамічними процесами і радіаційним станом нового безпечного конфайнменту ЧАЕС на основі технології цифрових двійників. Вісник ХНТУ, 2022, № 4(83), с. 67–73. DOI: 10.35546/kntu2078-4481.2022.4.9.

7. Minerva R., Lee G. M., Crespi N. Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models. *Proceedings of the IEEE*, 2020, 108(6), pp. 1785–1824. DOI: 10.1109/JPROC.2020.2998530.

8. OpenAPI Initiative. OpenAPI Specification (Version 3.1.0), 2021. URL: https://spec.openapis.org/oas/v3.1.0 (дата звернення: 24.09.2025).

9. Raghunandan A., Kalasapura D., Caesar M. Digital Twinning for Microservice Architectures. In: Proc. IEEE Int. Conf. on Communications (ICC 2023), 2023, pp. 3018–3023. DOI: 10.1109/ICC45041.2023.10279802.

10. Rasheed A., San **O.,** Kvamsdal T. Digital Twin: Values, Challenges and Enablers from a Modeling Perspective. *IEEE Access*, 2020, 8, pp. 21980–22012. DOI: 10.1109/ACCESS.2020.2970143.

Дата надходження статті: 25.09.2025
Дата прийняття статті: 20.10.2025
Опубліковано: 04.12.2025