МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ВОЗНЮК АНДРІЙ ВІКТОРОВИЧ

Допускається до захисту:
завідувач кафедри
прикладної математики
та кібербезпеки
д-р. філос. з математики
_____ А. В. Луценко
“____”_____20__ р.

РОЗРОБКА БАГАТОАГЕНТНОЇ АУТИНТИФІКАЦІЇ ЦИФРОВИХ ДВІЙНИКІВ В КІБЕРФІЗИЧНИХ СИСТЕМАХ МЕТОДАМИ СИСТЕМНОГО АНАЛІЗУ ТА АЛГЕБРИ КОРТЕЖІВ

Спеціальність 113 Прикладна математика
Кваліфікаційна (магістерська) робота

Науковий керівник:
Л. В. Загоруйко, к.т.н., доцент,
доцент кафедри прикладної
математики та кібербезпеки
_____
(підпис)

Оцінка ____/____/____
(бали за шкалою ЄКТС/за національною шкалою)
Голова ЕК:_____
(підпис)

Вінниця 2024

АНОТАЦІЯ

Вознюк А. В. Розробка багатоагентної аутинтифікації цифрових двійників в кіберфізичних системах методами системного аналізу та алгебри кортежів. Спеціальність 113 “Прикладна математика”, Освітня програма “Прикладна математика”. Донецький національний університет імені Василя Стуса, Вінниця, 2024.

У кваліфікаційній роботі досліджено аутентифікацію цифрових двійників у багатоагентних кіберфізичних системах. Встановлено сутності, що приймають участь в даному процесі та їх характеристики. Визначено закони, що описують відношення між виявленими сутностями. Створено програмне забезпечення, що надає можливість задавати характеристики мульти-агентної системи та, базуючись на створеній мета-моделі, автоматизувати процес створення, реалізації аутентифікації та розгортання цифрових двійників.

Ключові слова: цифровий двійник, аутентифікація, багатоагентна система, мульти-агентна система, алгебра кортежів.

52 с., 1 табл., 16 рис., 1 дод., 58 джерел

Vozniuk A. Development of a multi-agent authentication of digital doubles in cyber-physical system by using methods of systems analysis and tuple calculus. Specialty 113 “Applied mathematics”, Programme “Applied mathematics”. Vasyl’ Stus Donetsk National University, Vinnytsia, 2024.

The qualification paper investigates the authentication of digital twins in multi-agent cyber-physical systems. Entities participating in this process and their characteristics have been established. The laws describing the relationship between the identified entities are defined. Software has been created that provides an

opportunity to set the characteristics of a multi-agent system and, based on the created meta-model, to automate the process of creating, implementing authentication and deploying digital twins.

Keywords: digital twin, digital doubles, authentication, multi-agent system, tuple calculus.

Tabl. 1. Fig. 16. Bibliography: 58 items.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IoT — інтернет речей (internet of things)

OSI — мережева модель OSI (Open Systems Interconnection Basic Reference Model)

CA — центр сертифікації (Certificate authority)

PKI — інфраструктура відкритих ключів (Public key infrastructure)

TOTP — одноразовий пароль на основі часу (Time-based One-Time Password Algorithm)

ВСТУП

У технологічному світі, що постійно розвивається, концепція цифрових двійників стала рушійною силою, що змінює форму галузей і розкриває безпрецедентний потенціал. Цифровий двійник — це віртуальне представлення фізичного об’єкта або системи, що використовує дані в реальному часі для моделювання, моніторингу та аналізу його поведінки. Актуальність використання цифрових двійників поширюється на різні сектори, від виробництва та охорони здоров’я до міського планування та інших, сприяючи інноваціям та ефективності.

Цифрові двійники революціонізують виробничі процеси, надаючи цілісне уявлення про виробничі системи. Виробники можуть створювати віртуальні копії свого обладнання та цілих виробничих ліній, оптимізуючи продуктивність, прогнозуючи потреби в обслуговуванні та мінімізуючи час простою. Їх інтеграція з індустрією 4.0 забезпечує безперебійний зв’язок між фізичною та цифровою сферами, сприяючи адаптивному та чутливому виробничому середовищу.

У сфері охорони здоров’я цифрові двійники досягають значних успіхів, особливо в персоналізованій медицині. Створення цифрових копій органів або навіть цілих біологічних систем дозволяє проводити точне моделювання, допомагаючи зрозуміти хвороби та розробити цілеспрямовані методи лікування. Лікарі можуть використовувати цифрових двійників для тестування різних сценаріїв лікування та прогнозування відповідей пацієнтів, що в кінцевому підсумку підвищує точність діагностики та покращує результати лікування.

Розгортання цифрових двійників у міському плануванні змінює спосіб проектування та управління містами. Створюючи віртуальні моделі цілих міських ландшафтів, містобудівники можуть симулювати вплив різних рішень, від змін інфраструктури до впровадження розумних технологій. Цей

проактивний підхід покращує сталість, управління ресурсами та загальну якість життя мешканців.

Поширення пристроїв інтернету речей (IoT) і передових сенсорних технологій стало рушійною силою популярності цифрових двійників. Ці технології дозволяють безперебійно збирати дані в режимі реального часу з фізичних активів, дозволяючи створювати та підтримувати точні цифрові копії. У міру того, як датчики стають доступнішими та доступнішими, організації знаходять все більш можливим впровадження цифрових двійників для широкого спектру додатків.

Експоненціальне зростання обчислювальної потужності та широке впровадження хмарних технологій забезпечили інфраструктуру, необхідну для обробки складних симуляцій та обробки даних, пов’язаних із цифровими двійниками. Хмара дозволяє зберігати й аналізувати величезні обсяги даних у режимі реального часу, дозволяючи організаціям розгортати та керувати цифровими двійниками в масштабі. Цей перехід до хмарних рішень демократизував доступ до технологій цифрових двійників, зробивши їх більш доступними для широкого кола галузей.

Аутентифікація відіграє ключову роль у забезпеченні безпеки та цілісності кіберфізичних систем. Ці системи, що інтегрують цифрові технології з фізичними процесами, все більше поширені в критичній інфраструктурі, промислових умовах, охороні здоров’я та транспорті. Важливість аутентифікації полягає в її здатності перевіряти ідентичність користувачів або пристроїв, які взаємодіють із системою. Це гарантує, що лише авторизовані особи або компоненти можуть отримати доступ до цих систем і контролювати їх, запобігаючи несанкціонованому доступу, втручанню та потенційним збоям.

Аутентифікація має першочергове значення в контексті цифрових двійників, оскільки ці віртуальні копії фізичних активів або систем стають все більш важливими в різних галузях промисловості, включаючи виробництво, охорону здоров’я та управління інфраструктурою. Цифрові двійники збирають,

відстежують і аналізують дані реального світу, а інформація, яку вони обробляють, часто є конфіденційною, цінною або критично важливою. Тому забезпечення безпеки та надійності цих віртуальних агентів має вирішальне значення. Аутентифікація відіграє ключову роль у перевірці ідентичності користувачів, пристроїв і програм, які взаємодіють із цифровими двійниками, запобігаючи несанкціонованому доступу та потенційному витоку даних.

Крім того, аутентифікація в контексті цифрових двійників підвищує їхню стійкість проти кібератак. Оскільки ці віртуальні копії стають все більш складними та взаємопов’язаними, вони стають привабливими цілями для зловмисників, які прагнуть порушити їх роботу або викрасти цінну інформацію. Надійні механізми аутентифікації можуть діяти як перша лінія захисту, перешкоджаючи несанкціонованому доступу та обмежуючи потенційну шкоду від кібератак.

Таким чином, аутентифікація є основоположним елементом безпеки та ефективності цифрових двійників. Він забезпечує цілісність даних, підтримує довіру та відповідність вимогам, а також підвищує стійкість до кіберзагроз. Перевіряючи ідентичність користувачів, пристроїв і програм, аутентифікація допомагає захистити конфіденційність, цілісність і доступність даних у цифрових двійниках, зрештою захищаючи фізичні активи, підвищуючи ефективність роботи та зменшуючи ризики в різних галузях.

Об’єктом дослідження в даній роботі є цифрові двійники в кіберфізичних системах.

Предметом дослідження є аутентифікація цифрових двійників у багатоагентних кіберфізичних системах.

Метою дослідження є розробка підходу та створення програмного забезпечення для організації аутентифікації цифрових двійників в мультиагентній кіберфізичній системі.

Завдання дослідження:

1. проаналізувати існуючі підходи щодо організації процесу аутентифікації;

2. проаналізувати життєвий цикл цифрового двійника та визначити етапи, що потребують різних підходів щодо впровадження аутентифікації;
3. виявити елементи мультиагентної кіберфізичної системи, з якими взаємодіє цифровий двійник;
4. виявити сутності, що приймають участь в аутентифікації цифрового двійника, використовуючи модель сутність-зв'язок;
5. дослідити залежності між виявленими сутностями;
6. визначити обмеження, що мають місце в процесі впровадження аутентифікації на кожному етапі життєвого циклу цифрового двійника;
7. розробити підхід для організації аутентифікації цифрових двійників в мультиагентній кіберфізичній системі;
8. створити програмне забезпечення, що реалізує розроблений підхід.

Методами дослідження обрано системний аналіз та алгебру кортежів. Системний аналіз забезпечує розуміння вимог, визначення меж системи, можливість моделювання даних та процесів, виявлення обмежень та аналіз створеного прототипу чи системи. Алгебра кортежів забезпечує формальний і точний спосіб представлення та маніпулювання зв’язками між сутностями. У контексті багатоагентних систем, які включають множину автономних об’єктів, що взаємодіють один з одним, алгебра відношень виявляється особливо вигідною з кількох причин:

- формальне представлення відношень — алгебра кортежів дозволяє формалізувати зв’язки між агентами в структурований та однозначний спосіб;
- композиційність — алгебра кортежів підтримує композиційність, тобто складні зв’язки можна побудувати з простіших. Це корисно для моделювання складної мережі взаємодій у багатоагентних системах, де агенти можуть брати участь у різних відносинах одночасно;
- послідовність і узгодженість — покладаючись на формальну математичну структуру, таку як алгебра кортежів, легше забезпечити послідовність і

узгодженість моделей, що представляють зв’язки. Це має вирішальне значення для уникнення неоднозначності та протиріч у специфікаціях багатоагентних систем;

- забезпечує міцну основу для аналізу та міркувань про відношення. Поведінку, залежності та взаємодію агентів можна систематично аналізувати, дозволяючи ідентифікувати потенційні проблеми та перевіряти властивості системи;
- масштабованість — багатоагентні системи часто охоплюють велику кількість агентів із різноманітними відносинами. Алгебра кортежів є масштабованою, що робить можливим моделювання та аналіз складних зв’язків у системах із значною кількістю агентів;
- пристосованість до змін — багатоагентні системи за своєю суттю є динамічними, відносини яких розвиваються з часом. Здатність алгебри кортежів адаптуватися до змін у відносинах є перевагою для моделювання динамічної природи взаємодій у багатоагентних системах.

РОЗДІЛ 1

1.1 Аутентифікація в багатоагентних системах

Згідно з [1,2,3] багатоагентною системою є система, що складається з кількох автономних агентів, які взаємодіють один з одним і своїм середовищем для досягнення конкретних цілей або завдань. Кожен агент у багатоагентній системі є автономною сутністю зі своїм власним набором можливостей, знань і цілей. Ці агенти можуть працювати разом, конкурувати або діяти незалежно.

У багатоагентній системі агенти мають кілька важливих характеристик [4]:

- автономність: агенти, хоча б частково, незалежні;
- обмеженість уявлення: у жодного з агентів немає уявлення про всю систему, або система занадто складна, щоб знання про неї мало практичне застосування для агента;
- децентралізація: немає агентів, що керують усією системою [5].

У контексті аутентифікації, розглядаючи дані характеристики, можна виділити наступні обмеження, що мають місце у будь-який момент часу для кожного агенту:

- агент має скінченну множину каналів зв'язку з іншими об'єктами з якими він взаємодіє;
- агент може не володіти знаннями щодо того, яким чином буде використана інформація, яку отримують від нього інші об'єкти системи;
- кожна взаємодія має протокол передачі даних, що відомий усім агентам, які приймають у ній участь, адже, в зворотному випадку, комунікація не могла б відбутися.

Зважаючи на наведені вище характеристики та обмеження, для побудови системи аутентифікації доцільно оперувати такими сутностями як канал взаємодії та протокол передачі даних, що використовується для конкретного каналу зв'язку. При цьому, під протоколом взаємодії, тут і надалі, мається на увазі лише та частина, що виконує задачу аутентифікації. Варто зазначити деякі особливості даних сутностей:

- усі агенти, що приймають участь у певній взаємодії, використовують однаковий протокол передачі даних;
- кожна взаємодія визначається множиною агентів, що приймають у ній участь та протоколами передачі даних, що ними використовуються. Якщо, зважаючи на динамічну природу системи, деякі з цих параметрів змінилися, то така взаємодія розглядається, як інша, незважаючи на те чи її задачі у контексті предметної області залишилися сталими;
- якщо під час певної взаємодії не передаються дані, що використовуються для аутентифікації, то вважаємо, що використовується пуста множина протоколів передачі даних;
- канал взаємодії може містити у собі інші канали (наприклад, як у випадку з різними рівнями моделі OSI).

Загальні кроки автентифікації згідно з [7, 8, 9] є такими:

1. Початковий крок: заявник не аутентифікований.
2. Етап підключення: заявник вимагає від сервісу ідентифікації використання функції, яка потребується для аутентифікації. Сервіс ідентифікації запитує монітор для аутентифікації заявника.
3. Етап аутентифікації: заявник аутентифікується та відкривається сесія. Сервіс ідентифікації забезпечує користувачеві необхідні функції.

4. Етап відключення: користувач під’єднується або від’єднується від монітора і стан повертається до початкового кроку. Цей крок може бути ініційовано за тайм-аутом або дією користувача.

1.2 Аутентифікація цифрових двійників

Згідно з [10, 11, 12] виділяють такі основні види цифрових двійників:

- двійники продукту (Product Twins)
- двійники процесу (Process Twins)
- двійники активів (Asset Twins)
- двійники систем (System Twins)

У контексті аутентифікації, для цифрових двійників, що складаються з великої кількості компонентів, можна провести їх декомпозицію та розглядати як декілька цифрових двійників.

Можна виділити два види обмежень, що впливають на вибір способу аутентифікації. Перші можна описати категорійною змінною та множинною значень, які вона може або не може приймати. Наприклад, деякі об'єкти кіберфізичної системи можуть підтримувати лише протокол HTTP, що виключає можливість застосування способів аутентифікації, які створені для інших протоколів прикладного рівня. З іншої сторони існують обмеження, які оцінюються величинами, що є неперервними за своєю природою. Прикладом такого може бути обмежена обчислювальна спроможність пристрою чи швидкість передачі даних каналів, що можуть використовуватись конкретним об'єктом системи.

Згідно з [13, 14, 15] взаємодію цифрових двійників між собою та з іншими об'єктами кіберфізичної системи можна схематично зобразити так, як показано на рисунку 1.1.

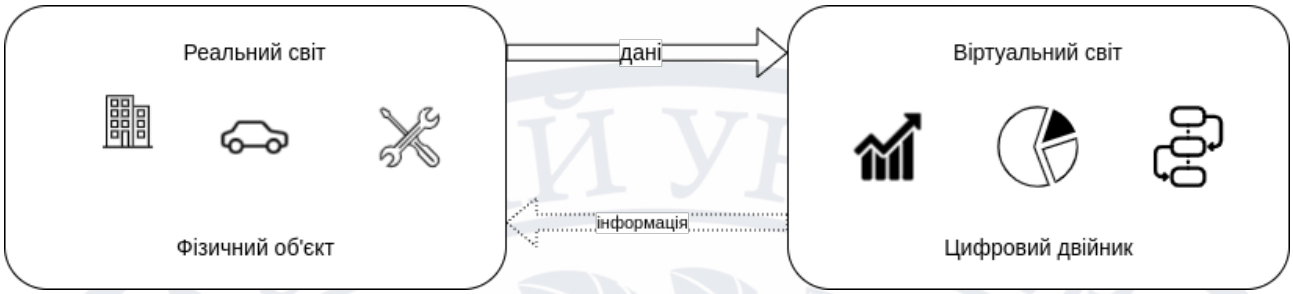

Рисунок 1.1 — Взаємодія цифрового двійника з реальним світом

У системах, що містять цифрових двійників, зв’язок охоплює кілька ключових типів, щоб полегшити обмін інформацією та сприяти симбіотичному зв’язку між фізичною сутністю та її цифровим аналогом. Передача даних є основоположним аспектом, де сенсорні дані з реальної системи постійно збираються датчиками та передаються на цифровий двійник. Ці дані включають різноманітні параметри, забезпечуючи цифровому двійнику представлення в реальному часі стану та поведінки фізичної системи.

Крім того, комунікація команд і зворотного зв'язку відіграє ключову роль у забезпеченні динамічної взаємодії між фізичною та віртуальною сферами. Статті, прогнози та стратегії оптимізації, створені цифровим двійником, можуть бути передані у фізичну систему, щоб впливати на її роботу. Цей двонапрямлений потік інформації забезпечує адаптивне керування, коли аналізи цифрового двійника інформують процеси прийняття рішень у реальному світі та можуть запускати коригування чи оптимізацію у фізичній системі. Інтеграція цих типів зв’язку гарантує, що цифрові двійники сприяють не тільки моніторингу та аналізу, але й активно формують і покращують продуктивність пов’язаних фізичних систем.

Загалом можна виділити наступні види взаємодій:

1. взаємодія реальних об'єктів між собою
2. взаємодія реальних об'єктів з цифровими двійниками
3. взаємодія цифрових двійників між собою

Кожен з наведених видів взаємодії має притаманну їм множину обмежень. Різниця між зв’язком від реального об’єкта до його цифрового двійника та зв’язком між реальними об’єктами полягає в природі та меті взаємодії. У контексті зв’язку між реальним об’єктом і цифровим подвійним зв’язком основна увага приділяється передачі даних і зворотному зв’язку між фізичною сутністю та її віртуальним представленням. Датчики на реальному об’єкті постійно збирають інформацію про його стан і оточення, і ці дані передаються на цифровий двійник для аналізу, моніторингу та моделювання. Комунікація характеризується одностороннім потоком інформації, причому цифровий двійник служить дзеркалом, яке відображає та всебічно представляє аналог у реальному світі.

І навпаки, у спілкуванні між реальними об’єктами взаємодія відбувається між фізичними об’єктами без посередництва цифрового двійника. Цей тип спілкування часто характеризується прямим, двонапрямленим обміном між двома чи більше матеріальними об’єктами. На відміну від зв’язку між реальним об’єктом і цифровим подвійником, який передбачає віртуальне представлення, зв’язок між реальним об’єктом безпосередньо впливає на поведінку та координацію фізичних об’єктів у найближчому оточенні. Обидві форми зв'язку відіграють важливу роль у формуванні майбутнього взаємопов'язаних систем.

1.3 Фактори аутентифікації та цифрові двійники

Фактори аутентифікації — це елементи або фрагменти інформації, які використовуються для перевірки суб'єкту, який намагається отримати доступ до системи, пристрою чи цифрового ресурсу. Ці фактори використовуються в процесах аутентифікації, щоб переконатися, що суб'єкт, який запитує доступ, є тим, за кого себе видає. Зазвичай виділяють три основних типи факторів аутентифікації [16, 17, 18]:

1. фактор знання — цей фактор включає інформацію або знання, якими повинен володіти лише законний користувач;

2. фактор володіння — цей фактор включає фізичний об’єкт або токен, яким володіє користувач і який може використовувати для аутентифікації;
3. фактор властивості — цей фактор базується на фізичних або поведінкових характеристиках притаманних користувачеві.

Існує велика кількість реалізацій [19, 20, 21] для кожного з них і їх множина постійно розширюється. Деякі з них вказані на рисунку 1.2.

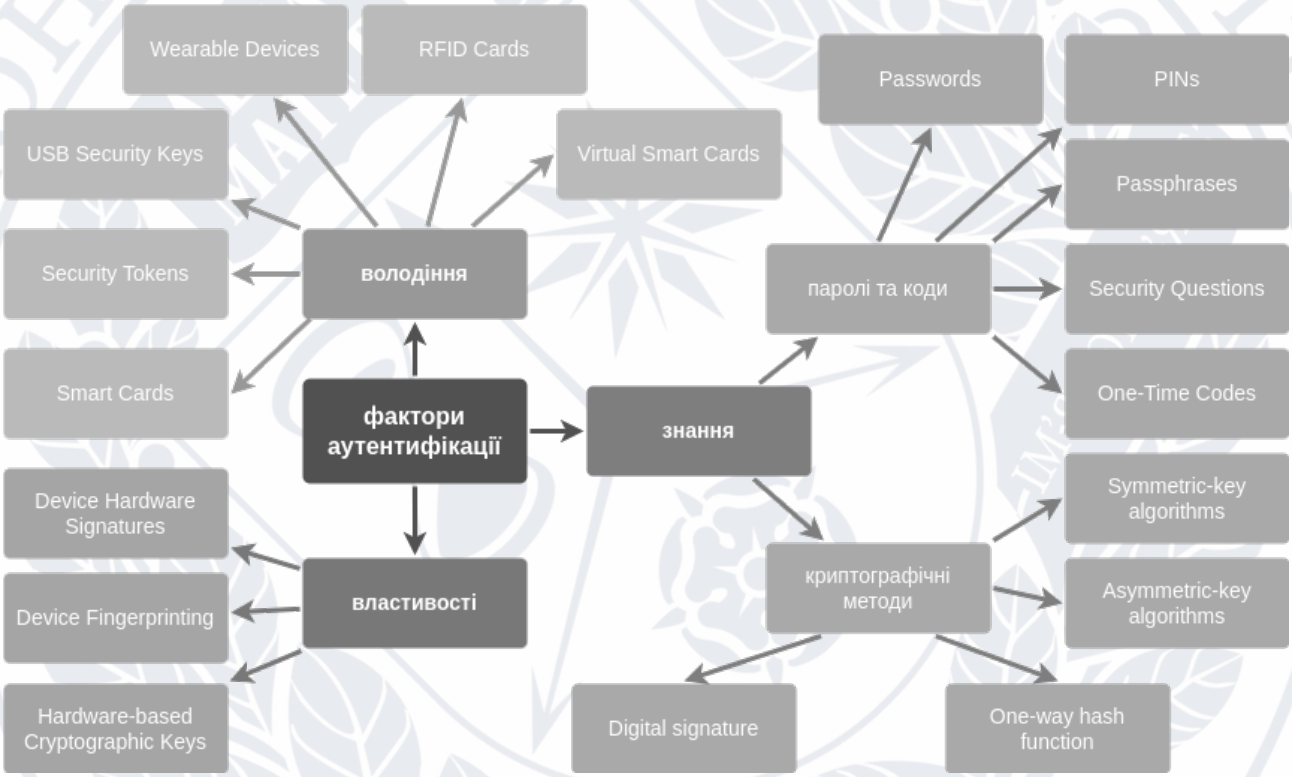

Рисунок 1.2 — Приклади факторів аутентифікації

Кожен з наведених факторів вимагає окремих підходів для реалізації системи аутентифікації.

1.3.1 Вхідні дані для фактору знання у цифрових двійників

Згідно з [22, 23, 24, 25] найбільш використовуваними є наступні фактори знання для аутентифікації:

- пароль;
- одноразовий пароль;

- криптографічні підходи, що базуються на публічних ключах;
- підтвердження з нульовим знанням (“zero-knowledge proofs” підходи);
- цифрові підписи.

Розглянемо кожен з варіантів реалізації фактору знання окремо.

В [26, 27] розглядаються такі недоліки паролю як фактору знання. У комп’ютерних системах із віддаленим доступом користувач ідентифікує себе в системі, надсилаючи секретний пароль. Зловмисник може дізнатися про секретний пароль, а потім імітувати його під час взаємодії з системою користувача трьома способами:

1. отримавши доступ до інформації, що зберігається всередині системи, наприклад, читанням системного файлу паролів;
2. перехоплюючи спілкування користувача з системою, наприклад, прослуховування лінії з’єднання терміналу користувача з системою або спостереження за виконання програми перевірки пароля.
3. через ненавмисне розкриття користувачем свого пароля, наприклад, вибрати пароль, який легко вгадати.

Третю можливість ніхто не може запобігти, тому зазвичай така можливість розкриття паролю не розглядається.

Щоб уникнути проблем, пов’язаних із повторним використанням пароля, були розроблені одноразові паролі [28, 29, 30, 31]. Існують два типи одноразових паролів, пароль запиту-відповіді і список паролів. Пароль запиту-відповіді відповідає значенням запиту після отримання ідентифікатора користувача. Відповідь обчислюється на основі значення відповіді (що базується на основі деякого значення) або вибирається з таблиці на основі відповіді користувача. Зазвичай такі знання є рекурсивними та генеруються на основі попередніх значень відповідей серверу чи клієнту.

Автентифікація з відкритим ключем — це надійний і безпечний метод, який широко використовується в різних цифрових системах для автентифікації користувачів і забезпечення безпечного доступу [32, 33, 34]. За своєю суттю цей механізм автентифікації покладається на пару криптографічних ключів — відкритий і закритий ключ. Відкритий ключ, як випливає з назви, надається відкрито та служить унікальним ідентифікатором, пов’язаним з користувачем. При цьому закритий ключ залишається конфіденційним і надійно зберігається на пристрої користувача. Коли користувач намагається отримати доступ до системи, система надає йому запит, представляючи криптографічну головоломку, яку можна вирішити лише за допомогою його закритого ключа. Відповідний відкритий ключ, уже збережений у системі, підтверджує особу користувача. Цей процес підвищує безпеку, усуваючи потребу в традиційних паролях, які вразливі до різноманітних кіберзагроз.

Однією з помітних переваг автентифікації з відкритим ключем є її здатність сприяти безпечному спілкуванню через незахищені мережі. Оскільки відкритий ключ є відкритим, він може вільно поширюватися та використовуватися будь-ким для шифрування повідомлень або даних, призначених для користувача, пов’язаного з цим ключем. Однак тільки власник відповідного закритого ключа може розшифрувати інформацію та отримати доступ до неї. Така асиметрія забезпечує конфіденційність і цілісність конфіденційних даних під час передачі. Крім того, автентифікація відкритих ключів підтримує відкликання та ротацію ключів, що дозволяє організаціям негайно відкликати доступ у разі порушення безпеки або регулярно змінювати ключі для посилення заходів безпеки.

Незважаючи на сильні сторони, реалізація автентифікації відкритих ключів вимагає ретельного розгляду методів керування ключами. Організації повинні встановити безпечні механізми для створення, розповсюдження та зберігання ключів, щоб запобігти несанкціонованому доступу або

компрометації. Оскільки технологія продовжує розвиватися, автентифікація з відкритим ключем залишається наріжним каменем у створенні безпечних, масштабованих і ефективних систем автентифікації в широкому діапазоні програм і галузей.

Самопідписані сертифікати в контексті автентифікації мають як переваги, так і недоліки [35, 36, 37, 38]. Позитивним моментом є те, що однією з помітних переваг використання самопідписаних сертифікатів є простота й економічність впровадження. На відміну від сертифікатів, отриманих від центрів сертифікації (CA), які часто передбачають грошові витрати та більш складний процес перевірки, самопідписані сертифікати можна створити та розгорнути без зовнішніх залежностей. Це робить їх зручним вибором для невеликих програм або внутрішніх систем, де бюджет на заходи безпеки може бути обмеженим.

Однак самопідписані сертифікати мають властиві недоліки, одним із яких є відсутність сторонньої перевірки. На відміну від сертифікатів, виданих авторитетними CA, самопідписані сертифікати не користуються довірою, встановленою загальновизнаними органами. Отже, коли користувач стикається з самопідписаним сертифікатом, існує більша ймовірність зіткнутися з попереджувальними повідомленнями чи підказками, що може призвести до потенційної плутанини та відчутного браку безпеки. Крім того, відсутність зовнішньої перевірки означає, що користувач повністю відповідає за перевірку автентичності сертифіката, створюючи потенційний шлях для атак типу "людина посередині", якщо з ним не поводитися обережно. Таким чином, хоча самопідписані сертифікати пропонують швидке та економічне рішення, ретельний розгляд і відповідні заходи безпеки є вирішальними для пом’якшення пов’язаних ризиків.

Підтвердження з нульовим знанням — це криптографічний метод, який використовується в аутентифікації та інших протоколах безпеки для підтвердження автентичності запиту без розкриття будь-якої конкретної

інформації про сам запит. У контексті аутентифікації підтвердження з нульовим знанням дозволяють користувачеві продемонструвати своє знання секрету (наприклад, пароля), фактично не розкриваючи сам секрет. Це забезпечує високий рівень безпеки та конфіденційності, оскільки верифікатор може бути впевнений в ідентичності користувача, не передаючи чи отримуючи жодної конфіденційної інформації.

Даний підхід все ширше використовується в IoT через свої унікальні властивості, які відповідають головним обмеження IoT пристроїв. У традиційних схемах аутентифікації пристрої IoT повинні зберігати великі обсяги даних аутентифікації, що може поставити ці дані під загрозу. Однак у zero knowlefge proof підході верифікатор чи інша особа нічого не дізнається з інтерактивної частини, що може ефективно захистити конфіденційність даних. Це дозволяє реалізувати наступні дві властивості [39]:

- анонімність. Прагнучи приховати справжні особи користувачів під час спілкування, IoT повинен забезпечити повну та взаємну анонімність для кожного вузла. Однак звичайні методи на основі проксі не підтримують аутентифікацію. Нульове знання, природно, забезпечує анонімність і підтримує аутентифікацію;
- опір атаці "Man-in-the-middle". В атаці “людина посередині” існує зловмисник, який може встановлювати незалежні з’єднання та може довільно отримувати доступ, змінювати та передавати повідомлення між двома сторонами, при цьому сторони комунікації не знають, що зв’язки між ними зламано. Підтвердження з нульовим знанням прив’язує інформацію користувача та дані обміну ключами. Під час прив’язки доказу будь-які спроби змінити повідомлення не можуть пройти перевірку законних користувачів. Згідно з цією ідеєю, схеми на основі підтвердження з нульовим знанням також можуть уникнути підробок, атак на паролі, атак із повторами, відстеження тощо.

Також у підході з підтвердженням з нульовим знанням порівняно низькі накладні витрати. Наприклад, якщо симетрична криптографія використовується для встановлення безпечного зв’язку через незахищені канали, накладні витрати на безпечну передачу симетричних ключів є дуже високими, оскільки кількість пристроїв IoT дуже велика. З іншого боку, інфраструктура відкритих ключів (PKI) вимагає високої обчислювальної складності. Це тому, що для середовища IoT більшість криптосистем із відкритим ключем недостатньо легкі. Підтвердження з нульовим знанням може ефективно зменшити обчислювальну складність і комунікаційну складність, поєднуючи їх з іншими технологіями. Як наслідок отримавши зменшення обчислювальної складності та зниження складності передачі даних.

Існують підходи до реалізації підтвердженням з нульовим знанням на базі технології blockchain, як у [40, 41, 42]. В такій роботі як [43] розглядається можливість реалізації згаданого підходу на основі хаотичних мап.

Якщо розглядати реалізацію аутентифікації без прив'язки до конкретного способу її реалізації, то, у випадку цифрових двійників, фактор знання може реалізовуватися двома шляхами:

- безпосереднє володіння даною інформацією;
- отримання необхідної інформації від інших суб'єктів кіберфізичної системи.

Розглядаючи дані підходи, слід мати на увазі швидкодію та кількість ресурсів, що затрачаються на генерацію необхідної інформації. З іншої сторони слід враховувати ріст складності системи, так як виникає необхідність вирішення усіх задач пов'язаних з життєвим циклом управління інформацією, що використовується для цього фактору аутентифікації. Наприклад, виникає необхідність реалізації або інтеграції такого широко вживаного підходу [26, 27] як ротація пароля. При цьому менеджмент ротації пароля може бути

реалізовано цифровим двійником або вимагатиме введення додаткового суб'єкту, що виконуватиме дану функцію. Переваги та недоліки кожного з підходів наведені у таблиці:

	Реалізація на стороні агента	Реалізація з застосуванням додаткового суб'єкту
Необхідність введення додаткових агентів у систему	Ні	Так
Необхідність реалізації інтеграції	Ні	Так
Необхідність додаткових каналів зв'язку	Ні	Так
Необхідність додаткових обчислювальних ресурсів	Так	Ні
Централізований менеджмент паролів	Ні	Так
Необхідність реалізації менеджменту ротації/зміни пароля	Так	Ні

З наведеної таблиці видно, що кожен з підходів має як переваги так і недоліки. Тому, зважаючи на параметри кіберфізичної системи та агентів, що входять до її складу, може бути застосовано як один підхід до усіх цифрових двійників так і обидва одночасно.

1.3.2 Вхідні дані для факторів володіння і властивості у цифрових двійників

Фактори володіння для цифрових двійників відіграють вирішальну роль у забезпеченні безпечної автентифікації та контролю доступу [44, 45, 46]. Ці

фактори допомагають перевірити, чи пристрій, який намагається отримати доступ до системи чи мережі, знаходиться у власності авторизованого користувача [47, 48]. Ось список факторів володіння, які можуть використовуватися пристроями та цифровими двійниками:

- апаратні токени - включення фізичних апаратних токенів, таких як ключі безпеки USB або модулі апаратної автентифікації, які потрібно фізично підключити до пристрою для автентифікації;
- радіочастотна ідентифікація (RFID) - використання технології RFID для автентифікації пристроїв. RFID-мітки або картки можна використовувати для надання доступу до певних пристроїв на основі їх унікальних ідентифікаторів;
- комунікація ближнього поля (NFC) - використання технології NFC для встановлення безпечних з’єднань між пристроями;
- унікальні ключі пристрою - створення унікальних криптографічних ключів для кожного пристрою, які надійно зберігаються на пристрої;
- аутентифікація Bluetooth - використання технології Bluetooth для автентифікації пристрою. Сполучення пристроїв за допомогою з’єднань Bluetooth може гарантувати, що лише сполучені та авторизовані пристрої зможуть спілкуватися;
- IMEI або серійні номери - перевірка міжнародної ідентифікації мобільного обладнання (IMEI) або унікальних серійних номерів для кожного пристрою.

Зважаючи на природу цих факторів та природу цифрових двійників, можливі два варіанти:

1. цифровий двійник відтворює об'єкт у мірі достатній для проходження аутентифікації за цим фактором;

2. цифровий двійник не має достатньо інформації для проходження аутентифікації за цим фактором.

Якщо у першому випадку достатньо лише реалізувати збір та передачу необхідних даних, то у другому випадку вводяться додаткові кроки, адже цифровий двійник не володіє такими даними.

В загальному вигляді задача забезпечення цифрового двійника необхідною інформацією вирішується створенням додаткового цифрового двійника для відповідного фактору. Він повинен бути двійником процесу передачі даних між об'єктом для якого створюється двійник та об'єктом, що виконує аутентифікацію.

В контексті аутентифікації цифрових двійників дещо виділяється використання алгоритму одноразового пароля на основі часу (TOTP). Одноразові паролі на основі часу (TOTP) пропонують надійний підхід до підвищення безпеки автентифікації шляхом впровадження динамічних, чутливих до часу кодів. У цьому методі користувачі зазвичай встановлюють програму автентифікації або використовують спеціальний апаратний маркер. Під час доступу до захищеної системи чи служби користувач надає другий фактор, крім свого статичного пароля. Цей другий фактор є кодом, згенерованим алгоритмом TOTP, і він змінюється через певний інтервал часу. Часовий характер кодів значно підвищує безпеку, оскільки зловмисникам стає значно складніше скомпрометувати облікові дані. Простота та легкість впровадження роблять TOTP популярним вибором [49, 50, 51], не створюючи зайвої складності як для розробників, так і для кінцевих користувачів. Відкриті стандарти, що лежать в основі TOTP, як визначено в RFC 6238, забезпечують сумісність між різними платформами, сприяючи взаємодії та широкому застосуванню.

Однак використання TOTP має певні недоліки [52, 53]. Користувачі повинні мати здатність генерувати динамічні коди, наприклад апаратний

маркер. Проблеми синхронізації між пристроєм користувача та годинником сервера іноді можуть призвести до проблем автентифікації. Крім того, користувачам необхідно встановити механізми резервного копіювання на випадок втрати або недоступності пристрою. Незважаючи на ці міркування, TOTP залишається практичним і ефективним методом двофакторної автентифікації, встановлюючи баланс між безпекою та зручним впровадженням у різноманітних програмах і службах.

Зважаючи на наведені переваги та недоліки, реалізація TOTP цифровими двійниками може значно збільшити як потребу у часі та ресурсах на їх реалізацію так і збільшення використання ресурсів самим цифровим двійником.

В даному розділі було проведено огляд існуючих рішень щодо впровадження аутентифікації в мульти-агентних системах. Також було досліджено особливості аутентифікації, що відносяться безпосередньо до цифрових двійників. Розглянуто фактори аутентифікації та відомі методи і алгоритми їх реалізації в умовах, що накладаються природою цифрових двійників як агентів системи. Отримані результати дають змогу перейти до формалізації задачі.

РОЗДІЛ 2

2.1 Аутентифікація цифрових двійників при наперед заданих обмеженнях

Розглянемо фрагмент мультиагентної системи на якому представлено як цифрові двійники так і реальні об'єкти кібер-фізичної системи (рисунок 2.1):

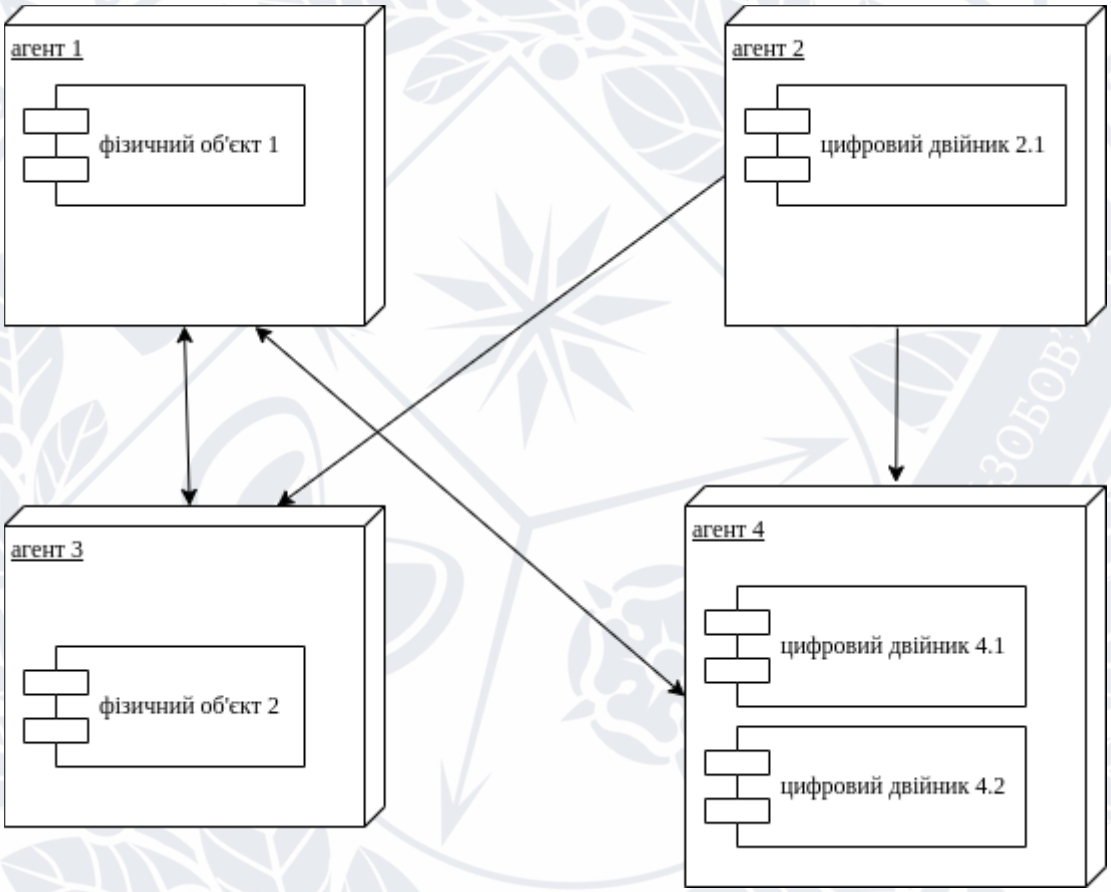

Рисунок 2.1 — Фрагмент мультиагентної системи

Агент може бути представлений:

- фізичним об'єктом;
- цифровим двійником;
- групою цифрових двійників (що, зазвичай, трапляється у випадках, коли різні аспекти фізичного об'єкту відтворюються окремими цифровими двійниками).

У контексті аутентифікації, взаємодія агентів складається з відправлення даних та власне їх перевірки, а тому може бути як одно-направленою так і багато-направленою.

ER-діаграма взаємодій в мультиагентній системі зображена на рисунку 2.2.

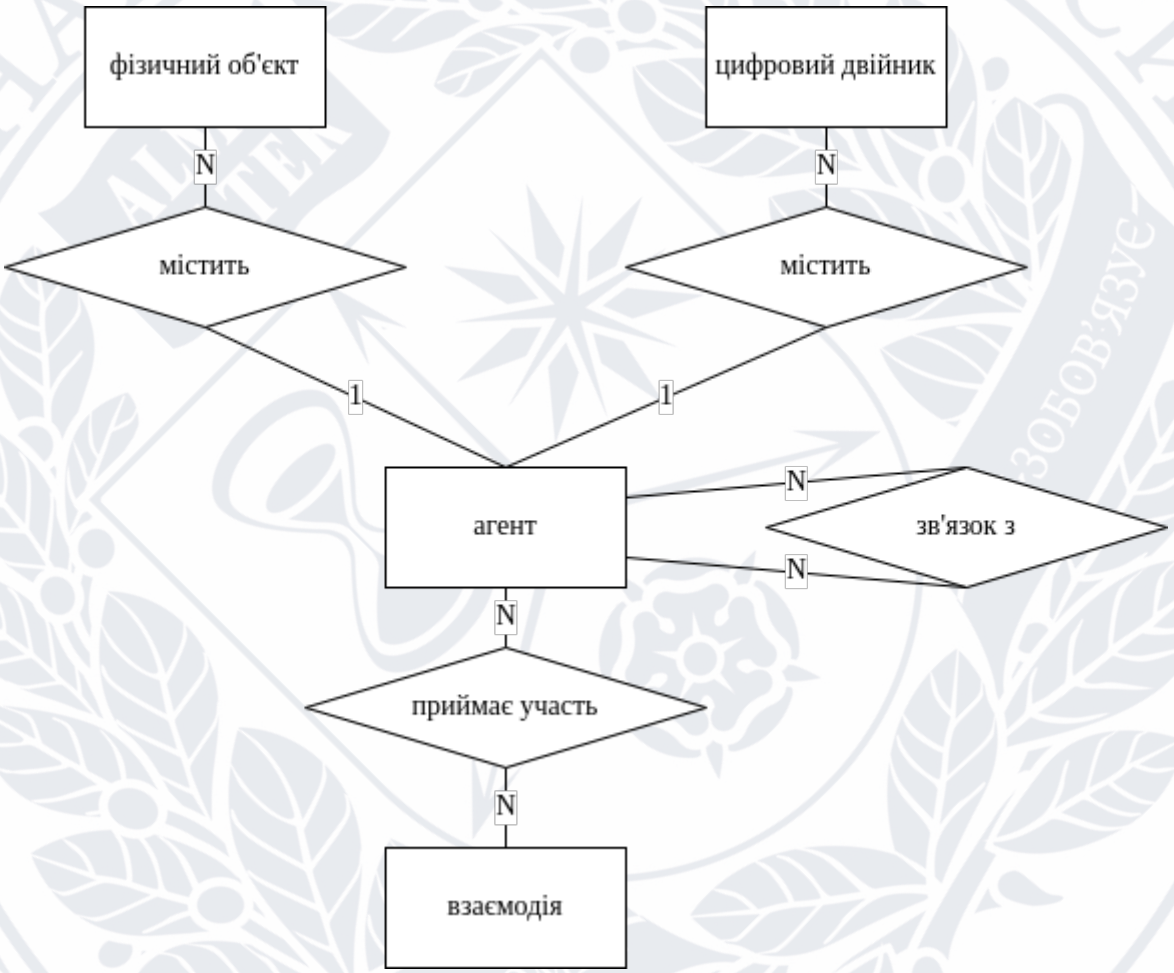

Рисунок 2.2 — ER-діаграма взаємодій в мультиагентній системі

Під час взаємодії агентів між собою, створюється множина каналів зв'язку, кожен з яких використовує певні протоколи зв'язку. Для підтримки механізму аутентифікації протокол визначає елементи даних, що передаються між агентами (пароль, токен, тощо) або мають бути в наявності у декого з них (сертифікат, приватний ключ, публічний ключ, тощо).

Складові окремо взятої взаємодії зображені на рисунку 2.3:

Рисунок 2.3 — Складові взаємодії між агентами

При цьому можна зазначити декілька властивостей наведених сутностей:

- кожна частина даних для аутентифікації відноситься до певного типу (наприклад пароль, authentication token, refresh token, сертифікат);
- кожен протокол може використовувати декілька частин даних для аутентифікації (як одного так і різних типів, наприклад вид хеш-функції та її вхідні параметри);
- різні протоколи можуть використовувати ті ж самі дані для аутентифікації (наприклад приватний чи публічний ключ);
- кожна взаємодія вимагає підтримку певної множини протоколів усіма сторонами, що приймають участь у комунікації.

Зважаючи на приведені вище властивості, можна визначити, що взаємодія та протокол, а також протокол та дані для аутентифікації знаходяться у відношенні багато до багатьох. При цьому тип даних та дані для аутентифікації знаходяться у відношенні один до багатьох.

ER-діаграма зображена на рисунку 2.4.

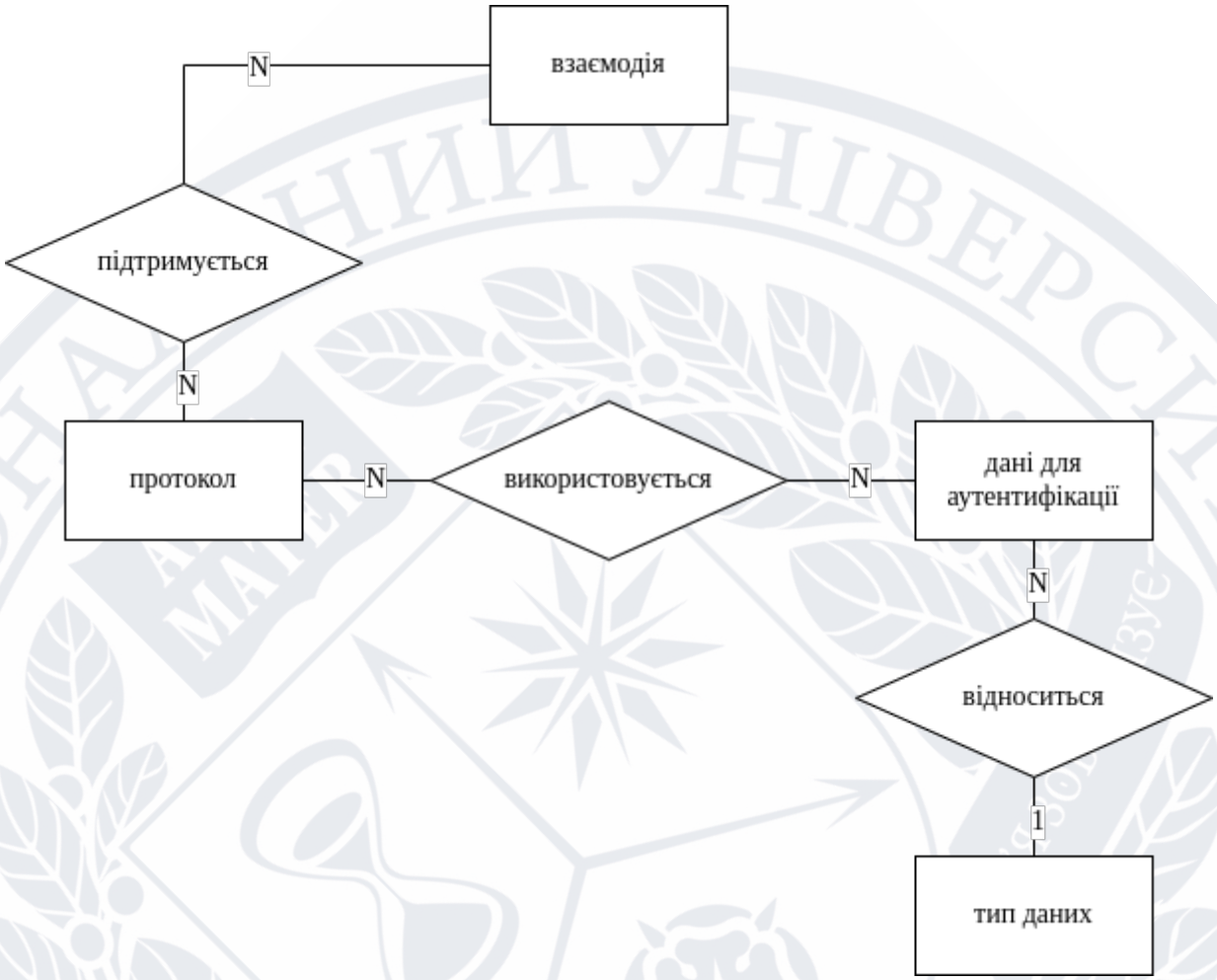

Рисунок 2.4 — ER-діаграма складових взаємодії агенту

Розглянемо цифрового двійника у контексті виконуваних ним задач. В даному контексті варто виділити наступні етапи його життєвого циклу [54]:

1. проектування;
2. створення;
3. розгортання;
4. існування як агента мультиагентної системи.

Детальніше життєвий цикл цифрового двійника зображено на рисунку 2.5.

Рисунок 2.5 — Життєвий цикл цифрового двійника

Для виконання будь-якої задачі цифровому двійнику потрібно прийняти участь у певній множині взаємодій з іншими двійниками чи об’єктами кібер-фізичної системи. Дана множина не є статичною, а змінюється при переході між етапами життєвого циклу двійника.

На етапах проектування та створення цифрового двійника існує можливість, що певну задачу можна виконати різними способами. Тому на цьому проміжку можна визначити множину множин взаємодій, кожна з яких відповідає окремому способу вирішення задачі.

На етапі функціонування цифрового двійника існує можливість того, що, в залежності від результатів виконання певної підзадачі, буде відбуватися перехід до різних станів системи. Як наслідок, для виконання однієї і тієї ж задачі, в різних випадках, цифровому двійнику необхідно приймати участь у різних взаємодіях.

Незалежно від того, які переходи між станами системи будуть відбуватися під час функціонування агенту, етап розгортання вимагає, щоб усі можливі взаємодії підтримувалися цифровим двійником. В протилежному випадку поставлена задача не зможе бути виконаною, що означатиме неможливість переходу до етапів проектування та створення.

Взаємовідношення між конкретною задачею, що виконується цифровим двійником, та взаємодіями, необхідними на різних етапах його існування схематично зображено на рисунку 2.5

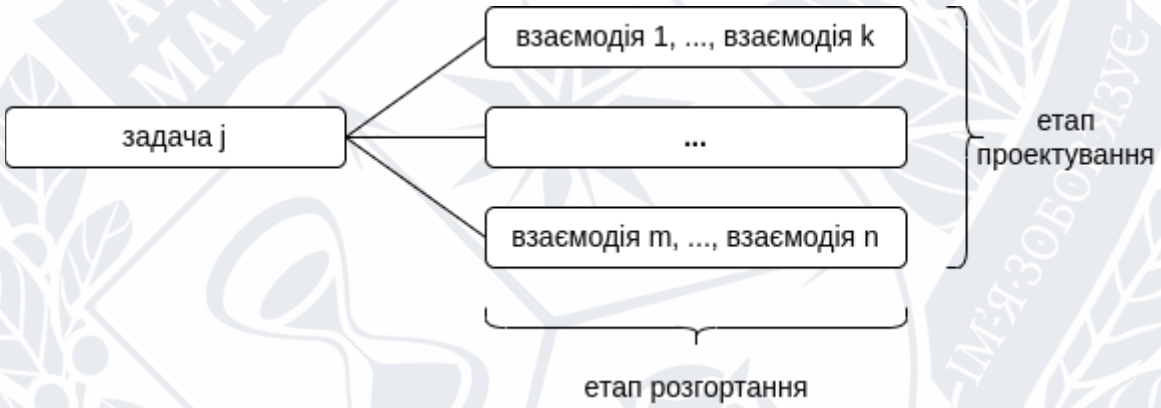

Рисунок 2.6 — Залежність задача-взаємодія від етапу життєвого циклу агенту

Кожна задача потребує певних ресурсів, щоб бути виконаною. При цьому деякі з типів ресурсів мають більше значення на початкових етапах (наприклад час, який необхідно витратити на реалізацію можливості виконання задачі агентом), а інші мають більше значення після розгортання цифрового двійника (наприклад об'єм оперативної пам'яті чи мінімальна пропускна спроможність каналу передачі даних). Звідси слідують такі фактори:

- кожному ресурсу можна поставити у відповідність певний тип ресурсу, кожен з яких оцінюється притаманними йому одиницями виміру;
- для кожного об'єкту існує обмеження зверху для кожного типу ресурсу;
- для кожного процесу існує обмеження зверху для кожного типу ресурсу;

- оцінка кількості використаних ресурсів проводиться з урахуванням обраного способу вирішення задачі;
- оцінка доцільності використання кожного типу ресурсу залежить від етапу життєвого циклу цифрового двійника.

Для того, щоб абстрагуватися від конкретних задач, що виконуються цифровим двійником, роботу будь-якого агенту кібер-фізичної системи можна розглядати як виконання ланцюжка обробників підзадач. Серед них можна виділити окремий ланцюжок обробників процесу аутентифікації.

В загальному вигляді створення системи аутентифікації цифрових двійників у багатоагентній кіберфізичній системі виглядає наступним чином:

1. визначення об'єктів та процесів, для яких створюватимуться цифрові двійники;
2. визначення агентів, що складають кібер-фізичну систему;
3. для кожного цифрового двійника визначення взаємодій в яких йому необхідно прийняти участь для виконання поставлених задач;
4. для кожної взаємодії, визначення каналів зв'язку, необхідних для її підтримки та протоколів передачі даних, що забезпечують ці канали;
5. визначення інформації, якою повинен володіти кожен цифровий двійник для проходження аутентифікації;
6. реалізація процесу зберігання усієї наведеної вище інформації;
7. реалізація процесу передачі усієї наведеної вище інформації;
8. реалізація протоколів передачі даних в межах цифрових двійників;
9. реалізація протоколів передачі даних в межах додаткових цифрових двійників, що виступають у ролі медіаторів.

Для можливості проходження перелічених кроків, необхідно визначити закони за якими функціонує мульти-агентна система у контексті наведених в цьому розділу сутностей та наявні обмеження.

2.1.1 Вибір способу аутентифікації для окремого цифрового двійника

Зважаючи на сутності, виділені у попередньому розділі, та необхідності можливості формалізації відношення між ними, введемо наступні позначення:

- A (Agent) — множина агентів кіберфізичної системи;
- T (Task) — множина задач, що виконуються агентами;
- TT (Task template) — варіант реалізації вирішення задачі на етапі проектування;
- DT (Digital twin) — множина цифрових двійників кіберфізичної системи;
- I (Interaction) — множина взаємодій між агентами;
- C (Connection) — множина каналів зв'язку;
- P (Protocol) — множина протоколів передачі даних для кожного каналу;
- AD (Authentication data) — множина елементів інформації для кожного протоколу, що використовуються для процесу аутентифікації;
- ADT (Authentication data type) — типи елементів інформації;
- R (Resource) — множина ресурсів, що використовується для певної взаємодії.
- RT (Resource type) — множина типів ресурсів;
- H (Handler) — множина обробників підзадач.

Розглянемо функціонування окремо взятого цифрового двійника як об'єкту мульти-агентної кібер-фізичної системи. В цьому випадку наперед відомі задачі, що мають бути вирішені цифровим двійником. Як наслідок, відома множина агентів з якими необхідно комунікувати для виконання кожної з

них. Також відомі принципи існування та параметри взаємодії реального об'єкту.

На етапах проектування та розробки цифровому двійнику можна поставити у відповідність множину множин взаємодій у яких потрібно прийняти участь для вирішення конкретної задачі. Її можна визначити наступним чином:

$$I|_{A_x} = A_x \bowtie_{x=T.aid} T_i \bowtie_{i=TT.tid} TT_j \bowtie_{j=I.ttid} I$$

де $i \in \left[0, n\left(A_x \bowtie_{x=T.aid} T_i\right)\right]$,

$j \in \left[0, n\left(A_x \bowtie_{x=T.aid} T_i \bowtie_{i=TT.tid} TT_j\right)\right]$

Перед переходом до наступних етапів проводиться пошук таких значень j, щоб множина агентів знаходилися у бієктивному відображенні з множинами взаємодій.

Будь-який цифровий двійник є частиною певного фізичного об'єкту. Як наслідок, існують обмеження щодо максимально можливих об'ємів використання ресурсів кожного типу. Аналогічні обмеження застосовуються до каналів, що використовуються цифровим двійником під час взаємодії з іншими агентами.

Так, як обмеження визначаються в залежності від типу ресурсу, то в загальному випадку дане обмеження можна описати наступним чином:

$$\forall i, RT_i\big|_{A_x} < RT_i^{max}\big|_{A_x}$$

Як зазначено вище, ліва частина містить дві складові. Загальний об'єм використаних агентом ресурсів конкретного типу можна визначити за формулою:

$$RT_j\big|_{A_i} = \sum_{k=1}^{n} I|_{A_j} \bowtie P \bowtie H \bowtie R_j \bowtie RT$$

Загальний об'єм використаних взаємодіями, в яких приймає участь конкретний агент, ресурсів конкретного типу можна визначити за формулою:

$$RT_j\big|_{A_i} = \sum_{k=1}^{n} I|_{A_j} \bowtie P \bowtie AD \bowtie H \bowtie R_j \bowtie RT$$

Звідси можна визначити загальну кількість ресурсів, що використовується конкретним агентом:

$$RT_j\big|_{A_i} = \sum_{k=1}^{n} I\big|_{A_j} \bowtie P \bowtie H \bowtie R_j \bowtie RT + \sum_{k=1}^{n} I\big|_{A_j} \bowtie P \bowtie AD \bowtie H \bowtie R_j \bowtie RT$$

На різних етапах життєвого циклу цифрового агенту різні типи ресурсів мають різне значення в оцінці доцільності використання кожної взаємодії. Для того, щоб проводити таку оцінку, необхідно ввести два додаткових вектори коефіцієнтів αта β, кожному елементу яких можна поставити у відповідність певний тип ресурсу. Значення коефіцієнту регулюватиме значимість типу ресурсу на кожному з етапів під час оцінки об'єму використаних ресурсів. Це дасть змогу позбутися від залежності від одиниць виміру різних видів ресурсів.

В результаті виводиться два обмеження. Перше для ресурсів на етапах проектування та розробки:

$$\sum_{j=1}^{n} \alpha_j \times R_j\big|_A \Rightarrow min$$

Друге для ресурсів на етапах функціонування:

$$\sum_{j=1}^{n} \beta_j \times R_j\big|_A \Rightarrow min$$

Кожен цифровий двійник складається з статичних та динамічних елементів. В загальному випадку, перші є незмінними з початку етапу функціонування агенту, а їх модифікацію можна вважати додатковим етапом розгортання. До таких елементів можна віднести протоколи, обробники задач, статичні елементи даних (наприклад сертифікат, приватний ключ, функція чи тип алгоритму, що використовується для кодування).

З точки зору визначення сутностей, необхідних для розгортання цифрового двійника, достатньо визначити обробники задач та статичні елементи даних. Решта визначається на основі них та взаємодій в яких має брати участь агент.

Множина статичних елементів даних залежить від протоколів, що реалізуються цифровим двійником. Вони, в свою чергу, залежать від взаємодій, в яких він приймає участь. Тому:

$$AD_{static}\big|_A = I\big|_A \bowtie P \bowtie AD \bowtie_{AD.id = ADT.adid \wedge ADT.type = static} ADT$$

Множина обробників задач, які повинні бути реалізовані цифровим двійником складається з обробників задач, що виконуються ним та множини обробників задач аутентифікації. Друга множина, у свою чергу складається з обробників протоколів та елементів даних. Звідси маємо:

$$H\big|_A = H_{auth} + H_{nonauth}$$

де $H_{nonauth} = A \bowtie H$,

$$H_{auth} = I\big|_A \bowtie P \bowtie H + I\big|_A \bowtie P \bowtie AD \bowtie H$$

Тому:

$$H\big|_A = I\big|_A \bowtie P \bowtie H + I\big|_A \bowtie P \bowtie AD \bowtie H + A \bowtie H$$

З точки зору визначення сутностей, необхідних для функціонування цифрового двійника, достатньо визначити динамічні елементи даних. Адже спосіб визначення обробників задач вже наведено вище та є відомим на етапі розгортання агенту. Їх знаходження є повністю аналогічним пошуку статичних елементів даних, лише фільтрація відбувається за інших значенням атрибуту типу даних:

$$AD_{dynamic}\big|_A = I\big|_A \bowtie P \bowtie AD \bowtie_{AD.id = ADT.adid \wedge ADT.type \neq static} ADT$$

В даному підрозділі було розглянуто складові цифрових двійників на різних етапах їх життєвого циклу, визначено залежності між сутностями, що дозволяють визначати ці складові. Також визначено обмеження, що мають місце

для кожного окремого цифрового двійника як складової мульти-агентної системи.

2.1.2 Вибір способу аутентифікації для групи цифрових двійників

Розглядаючи аутентифікацію в мульти-агентній системі, необхідно приймати до уваги усю множину цифрових двійників, що приймають у ній участь. Адже вибір способу функціонування для кожного з них окремо може призвести до таких негативних наслідків:

1. Реалізація надлишкового функціоналу. Наприклад, замість реалізації двох різних способів аутентифікації різними цифровими двійниками можна може бути доцільним використання одного, вимогливішого за певними ресурсами, але більш ефективного за них двох сумарно.
2. Зведення оцінки ефективності до локального мінімуму замість глобального. Це може бути наслідком того, що частина контексту втрачається при розгляді кожного двійника окремо. Як наслідок частина параметрів системи не береться до уваги.

Задля уникнення наведених недоліків доцільно розглянути такі параметри, що відображають оцінку параметрів системи в цілому. Аналогічно до характеристик, що розглядалися у відношенні до окремого цифрового двійника, можна виділити такі їх групи:

- множина усіх взаємодій до етапу розгортання;
- множина усіх взаємодій після етапу розгортання;
- обмеження щодо використання кожного типу ресурсу окремими цифровими двійниками;
- обмеження щодо використання кожного типу ресурсу системою в цілому на кожному етапі.

Розглядаючи взаємодії, що мають місце між цифровими двійниками між собою та іншими кібер-фізичними об'єктами, доцільно розглянути усю їх множину у рамках мульти-агентної системи. В такому випадку, задача визначення взаємодій, в яких прийматимуть участь агенти, зводиться до пошуку такої множини взаємодій I_X, що:

$$\forall i \in [0, n(A)], I|_{A_i} \in I_X$$

Зважаючи на залежності між сутностями, виведені у попередньому розділі, дане відношення можна записати у вигляді:

$$\forall i \in [0, n(A)], \exists j, A_i \bowtie T \bowtie TT_j \bowtie I \in I_X$$

Тобто задача визначення усієї множини взаємодій, що матимуть місце в системі зводиться до пошуку множини, що задовільняє наведену вище умову. Виходячи з того, що усі елементи системи, що приймають участь в цій умові знаходяться у відношенні багато до багатьох, випливає можливість існування декількох I_X, що її задовільняють. В найкращому випадку їх існує $n(A)$, в найгіршому — $n(A) \times n(T) \times n(TT) \times n(I)$.

З іншої сторони, пошук відповідної множини I_Xнеобхідно провести до етапу розгортання цифрових двійників. Введення додаткових обмежень дозволить скоротити кількість елементів вихідної множини. Як було розглянуто раніше, кожен агент системи має фізичні обмеження, що застосовуються для кожного типу ресурсу. Дану умову можна записати у вигляді:

$$\forall i \in [0, n(A)], \forall j \in [0, n(RT)], RT_j|_{A_i} < RT_j^{max}\Big|_{A_i}$$

де спосіб визначення $RT_j|_{A_i}$наведено у попередньому розділі.

Наведена умова є більш загальною, ніж аналогічна умова для єдиного цифрового двійника, та включає у себе усі можливі варіанти, що покриті попередньою, тому в подальшому використовуватиметься замість попереднього обмеження.

Для цього необхідно розглянути умову мінімізації використання кожного типу ресурсу на рівні мульти-агентної системи в цілому. При цьому розділивши її на дві окремих, що залежатимуть від етапу життєвого циклу цифрового двійника.

Для етапу проектування та розгортання:

$$\forall i \in [0, n(A)], \forall j \in [0, n(RT)], \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_j \times R_j|_{A_i} \Rightarrow min$$

Для етапу розгортання цифрових двійників:

$$\forall i \in [0, n(A)], \forall j \in [0, n(RT)], \sum_{i=1}^{n} \sum_{j=1}^{n} \beta_j \times R_j|_{A_i} \Rightarrow min$$

Перелічені в даному розділі умови та дають змогу побудови бієктивного відображення між A та I. При цьому вирішується задача мінімізації використовуваних ресурсів, а також враховуються обмеження, що стосуються цифрових двійників як складових фізичних об'єктів.

2.2 Аутентифікація цифрових двійників при відсутності заданих обмежень

Існують випадки, коли при виборі способу аутентифікації обмеження відсутні або ними можна знехтувати. До них можна віднести:

- наявні обмеження за усіма типами ресурсів значно перевищують споживання кожного з можливих способів аутентифікації, що планується використовувати цифровим двійником;
- реалізація взаємодій виноситься на агент, що виконує роль медіатора та має фізичні характеристики, що значно перевищують характеристики цифрового двійника, який делегує виконання задач медіатору;
- реалізація аутентифікації проводиться для організації взаємодії в рамках групи цифрових двійників. В такому разі немає обмежень, які виникають як наслідок необхідності впровадження способів аутентифікації, що підтримуються існуючими фізичними об'єктами мульти-агентної системи.

В такому разі достатньо реалізувати аутентифікацію, що відповідає загальним вимогам до побудови таких систем. Кроки для вибору способу аутентифікації зображено на рисунку 2.7.

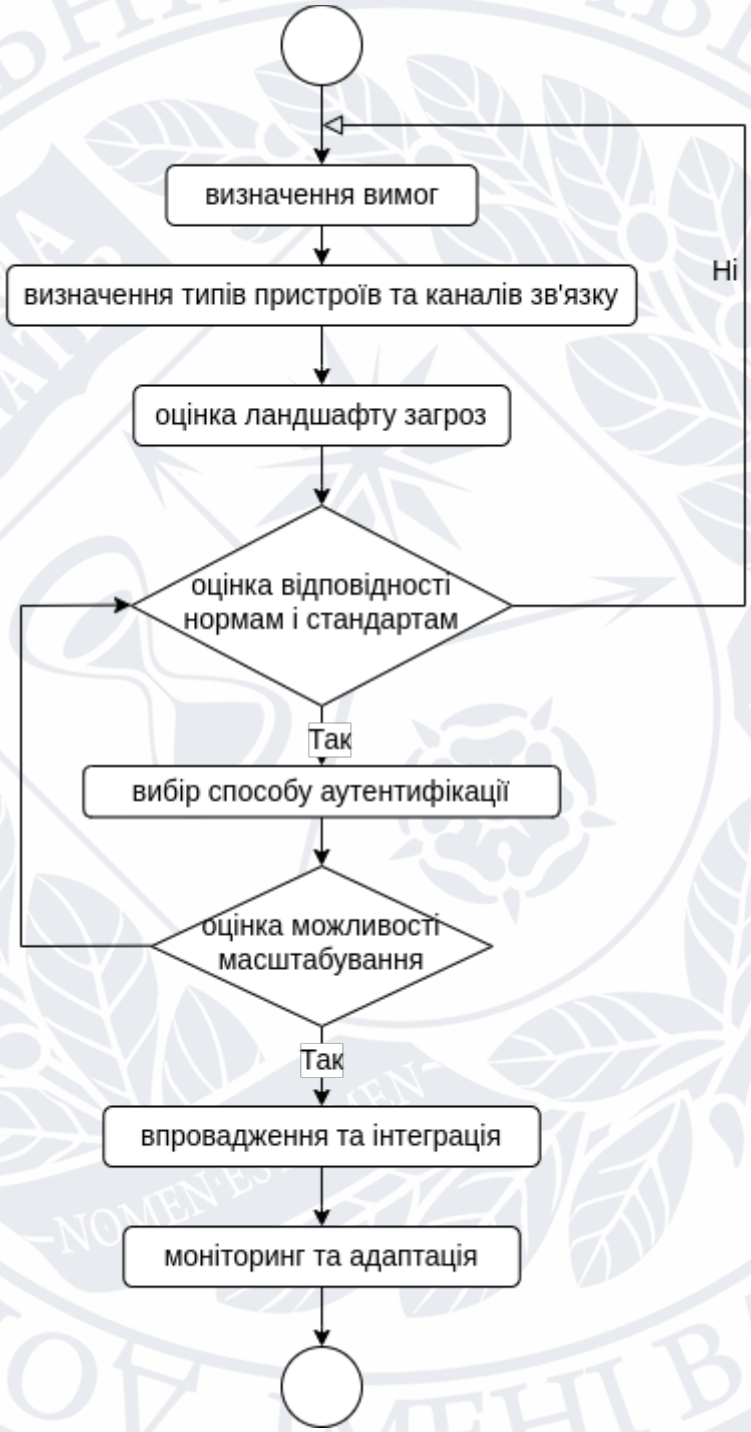

Рисунок 2.7 — Блок-схема кроків для вибору способу аутентифікації

Загалом, усі наведені кроки можна розбити на декілька етапів, кожен з яких складається з декілька кроків для досягнення певної цілі даного етапу.

На початковому етапі проводиться загальна оцінка системи, визначаються вимоги, типи пристроїв та канали зв'язку, що можуть ними використовуватися. Це необхідно для того, щоб визначити сімейства способів аутентифікації, які доцільно впроваджувати в даній мульти-агентній системі.

Наступним кроком здійснюється оцінка ландшафту загроз. Для цього необхідно оцінити потенційні загрози та вразливі місця, пов’язані з мульти-агентною системою. Отримані результати використовуються для розгляду впливу різних методів аутентифікації на пом’якшення цих загроз.

Результатом наведених кроків є множина методів аутентифікації, що покривають базові вимоги та потенційно можуть бути використані. Зважаючи на те, що цифрові двійники використовуються у великій кількості галузей, необхідно брати до уваги обмеження, що безпосередньо відносяться до кожної з них. Існують норми та стандарти, що регулюють спосіб аутентифікації, спосіб використання даних та інші аспекти операцій, що відбуваються як всередині агентів мульти-агентної системи, так і під час передачі інформації між ними. Наприклад, для оперування з медичними даними, цифровий двійник має відповідати стандарту HIPAA. Якщо ж він використовує особисті дані користувачів, то виникає необхідність відповідати стандарту GDPR. Існують інші стандарти для різних предметних областей. У кожному випадку вони мають розглядатися окремо, в залежності від того, якою інформацією оперують цифрові двійники та інші агенти системи.

Враховуючи те, що до складу мульти-агентної системи входить велика кількість цифрових двійників, доцільно провести оцінку можливості масштабування обраних методів аутентифікації. Це необхідно для вирішення наступних задач:

- переконатися, що обраний спосіб аутентифікації може масштабуватися відповідно до росту системи;
- зменшення зусиль щодо впровадження шляхом перевикористання реалізації різними агентами;
- оцінити вплив на продуктивність та системні ресурси.

Одним з останніх етапів є впровадження та інтеграція. Їх результатом є мульти-агентна система з підтримкою аутентифікації між будь-якими агентами, що можуть взаємодіяти між собою. Це дає можливість переходу до фінального етапу моніторингу та адаптації на якому, зважаючи на отримані метрики роботи системи, можливі подальші зміни для корегування впровадженого способу аутентифікації.

В цьому розділі було розглянуто процес побудови системи аутентифікації в мульти-агентній системі для окремого цифрового двійника та для їх множини. Проведено декомпозицію системи на її складові для подальшого її дослідження. Для кожного випадку виділено основні характеристики системи, що варто брати до уваги під час розробки, впровадження та функціонування цифрових двійників. Досліджено відношення між визначеними характеристиками та побудовано залежності між ними, що описують закони функціонування системи. Отримані залежності між елементами системи можуть бути використані для автоматизації вибору способу аутентифікації цифрових двійників.

РОЗДІЛ 3

3.1 Загальна архітектура програмної реалізації

Відокремлення метаданих, які використовуються для побудови системи, від даних, які належать системі, є фундаментальною практикою, яка підвищує гнучкість, зручність обслуговування та загальну ефективність системи. Метадані стосуються інформації, яка описує або визначає інші дані, і в контексті розробки програмного забезпечення вони включають деталі про структуру, поведінку та конфігурацію системи. Ізолюючи метадані від фактичних даних, можна досягти більшої модульності проектів. Це розділення дозволяє легше оновлювати та модифікувати структуру та поведінку системи без безпосереднього впливу на дані, якими вона керує.

Крім того, відокремлення метаданих від системних даних сприяє підвищенню цілісності та безпеки даних. Системні дані містять конфіденційну або критично важливу інформацію, і, відокремлюючи її від метаданих, можна установити більш надійний контроль доступу та застосувати заходи безпеки, спеціально розроблені для захисту даних. Таке розділення гарантує, що зміни в конфігурації або функціональності системи випадково не порушують цілісність і конфіденційність базових даних. Загалом чітке розмежування між метаданими та системними даними оптимізує процеси розробки, полегшує еволюцію системи та підвищує безпеку та надійність програмних додатків.

Також, практика відокремлення метаданих від даних має вирішальне значення для оптимізації продуктивності системи. Метадані часто включають інформацію про індексування, зв’язки та обмеження, які є критичними для пошуку та маніпулювання даними. Коли ці метадані зберігаються окремо, це дозволяє ефективно оптимізувати запити та стратегії індексування без прямого впливу на збережені дані. Такий поділ дозволяє точно налаштовувати характеристики продуктивності системи, не змінюючи цілісності або структури базових даних, створюючи програмне рішення з більшою швидкістю

реагування та масштабованістю. Таким чином, поділ метаданих і даних у розробці програмного забезпечення не тільки сприяє адаптації та співпраці, але також сприяє оптимізації продуктивності системи.

Враховуючи визначені в попередніх розділах сутності, що приймають участь в організації системи аутентифікації, можна виділити три основних компоненти системи, що зображені на рисунку 3.1.

Рисунок 3.1 — Основні компоненти системи

Зважаючи на життєвий цикл цифрових двійників та дії, що необхідно виконувати для реалізації їх аутентифікації в мульти-агентних системах, можна виділити наступні складові системи:

- User/Actor — безпосередньо користувач, що створює конфігурацію мульти-агентної системи та ініціює розгортання цифрових двійників;
- Configurer — сервіс, що відповідає за зберігання інформації про складові системи та їх параметри;
- Estimator — сервіс, що відповідає за обчислення параметрів цифрових двійників, що створюються;
- Deployer — сервіс, що відповідає за розгортання цифрових двійників та введення їх в мульту-агентну системи;
- Executor — множина реалізацій поведінки цифрових двійників (у тому числі процесу аутентифікації).

Sequence-діаграма процесу роботи системи зображена на рисунку 3.2.

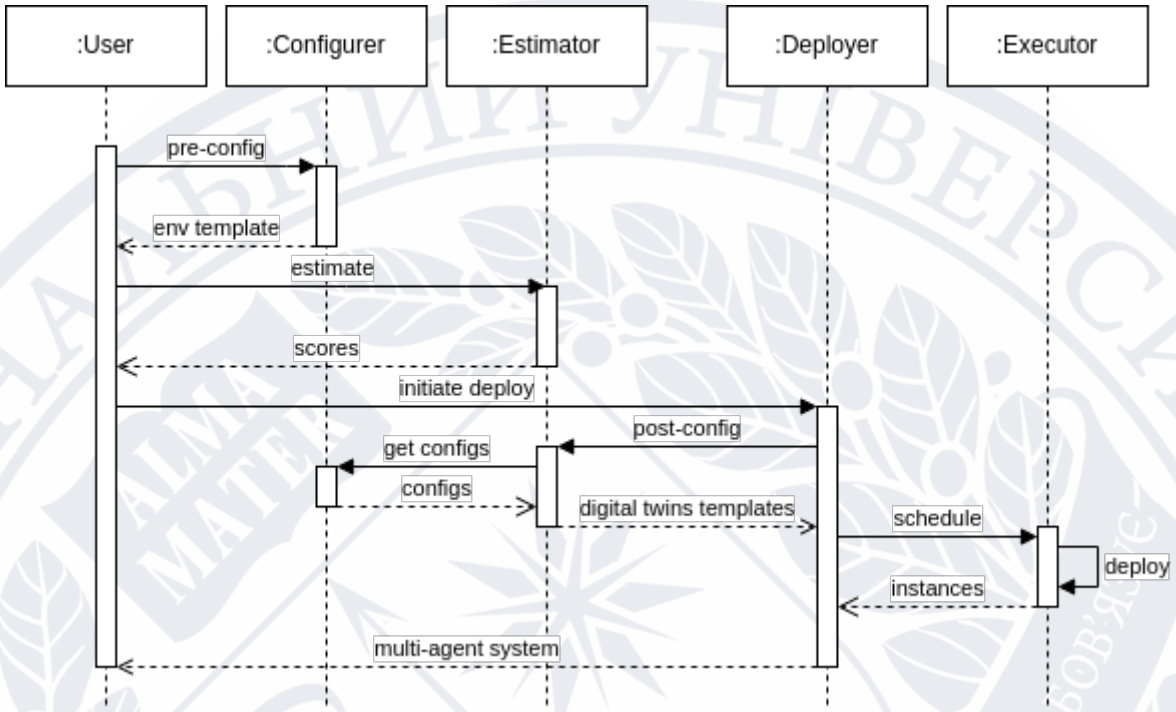

Рисунок 3.2 — Sequence-діаграма процесу роботи системи

Розглянемо роботу системи покроково. На початковому етапі відбувається попереднє налаштування параметрів системи:

1. задається множина цифрових двійників, які необхідно створити;
2. визначається множина задач, що має виконуватися кожним агентом;
3. для кожної задачі задаються множини взаємодій у яких необхідно прийняти участь для її виконання в залежності від обраного способу;
4. для кожної взаємодії задається множина протоколів, що нею підтримується;
5. для кожного протоколу задається множина елементів даних необхідних для проходження процесу аутентифікації;
6. для кожного елементу даних задається множина його параметрів.

За цей етап відповідає сервіс Configurer. Результатом його роботи є шаблон мульти-агентної системи.

Наступним кроком проводиться оцінка параметрів системи, визначення взаємодій, в яких прийматиме участь кожен цифровий двійник та параметрів усіх пов'язаних з ними сутностей (протоколів, елементів даних аутентифікації,

тощо). За цей етап відповідає сервіс Estimator. Результатом його роботи є множина шаблонів, кожен з яких відповідає сімейству однотипних цифрових двійників.

За наступний етап відповідає сервіс Deployer. На цьому етапі відбувається визначення параметрів, що необхідні для створення шаблонів конкретних екземплярів цифрових двійників.

Останнім кроком відбувається безпосередньо розгортання цифрових двійників на основні отриманих шаблонів, а також їх існування в якості агентів мульти-агентної системи. За цей етап відповідає сервіс Executor.

Відокремлення обробника аутентифікації від інших обробників є доцільною практикою, яка покращує модульність, зручність обслуговування та безпеку. Ізолюючи логіку аутентифікації у власний спеціальний обробник, можна отримати чисту та організовану кодову базу. Це розділення дозволяє створити більш модульну архітектуру, де проблеми аутентифікації інкапсульовані в окремому компоненті, який може бути перевикористаним різними цифровими двійниками. У результаті це зменшує ймовірність небажаних побічних ефектів під час модифікації або розширення інших частин системи.

Крім того, відокремлення обробника аутентифікації підвищує безпеку, надаючи централізовану спеціалізовану точку контролю для операцій, пов’язаних з аутентифікацією. Ця ізоляція мінімізує ризик уразливості безпеки та полегшує застосування узгоджених політик безпеки для всієї системи. У разі оновлень або змін у вимогах до аутентифікації можна вирішити ці проблеми за допомогою спеціального обробника аутентифікації, не впливаючи на функціональність інших компонентів, що сприяє створенню більш стійкої та безпечної архітектури програмного забезпечення.

Цифровий двійник з точки зору обробників задач, які ним використовуються, у загальному вигляді зображено на рисунку 3.3:

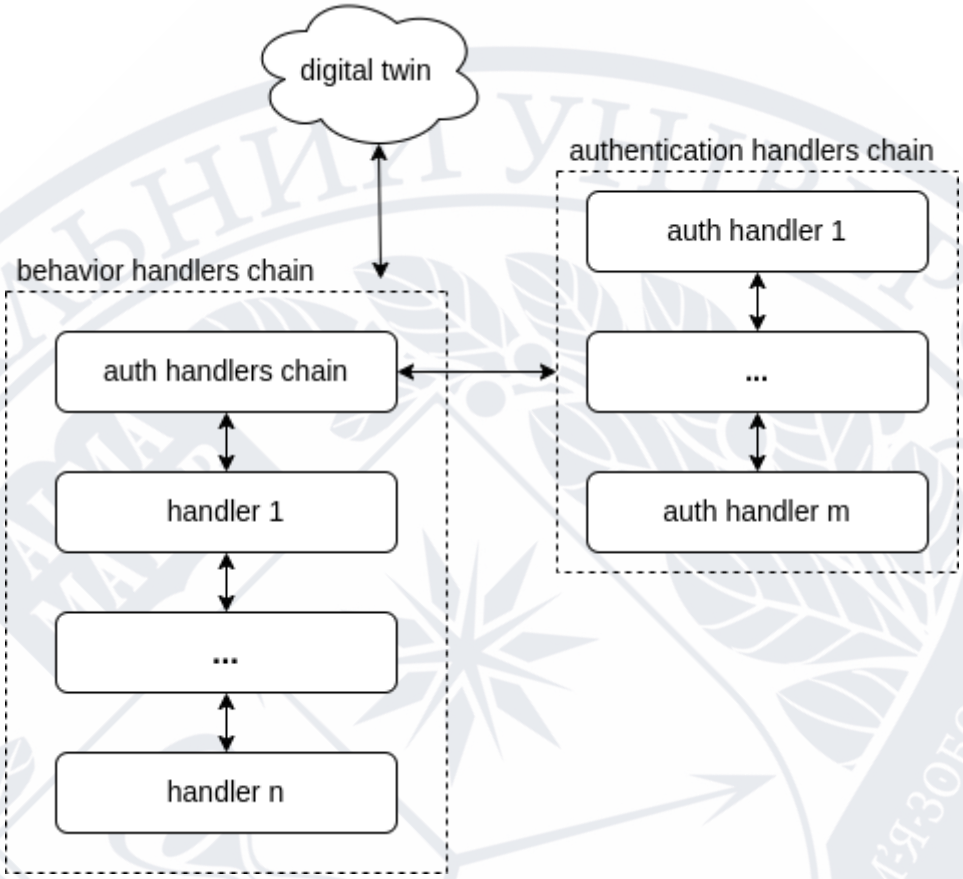

Рисунок 3.3 — Структура екземпляру цифрового двійника

Кожен екземпляр цифрового двійника складається з двох частин:

- ланцюжок обробників поведінки двійника;
- ланцюжок обробників процесу аутентифікації.

В даному підрозділі було визначено основні компоненти системи, сервіси з яких вони складаються та зони відповідальності кожного з них. Також визначено функціонал за який відповідає кожен сервіс. Наступним кроком доцільно розробити структуру сховища даних.

3.2 Розробка структури бази даних

Сутності предметної області та відношення між ними було розглянуто у другому розділі цієї роботи.

Діаграма структури бази даних наведена нижче (з метою покращення читабельності діаграми деякі атрибути на ній не відображені):

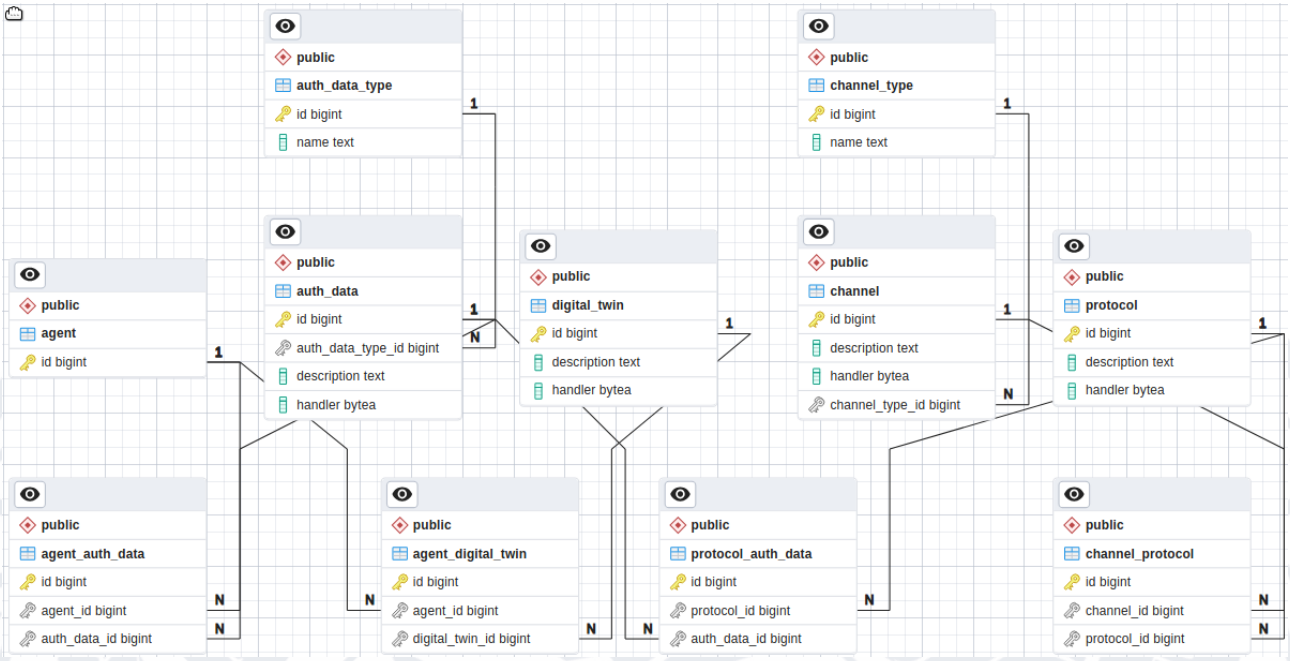

Рисунок 3.3 — ER-діаграма бази даних

Як видно з наведеної діаграми, для кожної сутності, визначеної вище, створюється окрема таблиця. Також вводяться додаткові таблиці для можливості реалізації відношення many-to-many.

Окрім атрибутів, що вводяться для опису системи, вводиться атрибут handler, що має безпосереднє відношення до програмної реалізації процесу аутентифікації та містить метадані, що визначають яким чином обробляється той чи інший процес кіберфізичної системи.

Інтеграція стовпця, який посилається на реалізацію обробника в розробці структури бази даних, дає значні переваги. Однією з ключових переваг є сприяння повторному використанню коду та зручності обслуговування. Використовуючи обробники, можна інкапсулювати певні функції або бізнес-логіку та легко повторно використовувати їх у різних частинах бази даних (для різних цифрових двійників). Ця модульна конструкція полегшує ефективне оновлення або модифікацію окремих компонентів без необхідності масштабних змін, підвищуючи загальну зручність обслуговування бази даних. Крім того, використання колонок обробників сприяє адаптації до вимог, що змінюються. Різні типи даних можуть вимагати унікальної обробки, а обробники

забезпечують гнучкий механізм для налаштування поведінки без зміни основної схеми бази даних.

В програмній реалізації, кожному значенню handler можна поставити у відповідність ієрархію класів, що відповідає за обробку конкретного типу процесу (наприклад визначення пари користувач/пароль для конкретної взаємодії конкретного цифрового двійника чи отримання токена для аутентифікації).

3.3 Розробка сервісів конфігурування та розгортання мультиагентної системи

Зоною відповідальності Configurer сервісу є інформація про складові системи та їх параметри. Також він забезпечує їх зберігання та підтримку операцій над ними.

Зважаючи на те, що метадані та системні дані були відокремленні, Configurer може бути реалізований як звичайний REST сервіс, що надає можливість виконання CRUD операцій над сутностями, що описують мульти-агентну систему. Цього достатньо, щоб покрити усі операції, що можуть бути необхідними для визначення початкової конфігурації системи. Для спрощення взаємодії з даним сервісом використано HAL Explorer, що забезпечує інтерфейс користувача. Приклад їх роботи зображено на рисунку 3.4.

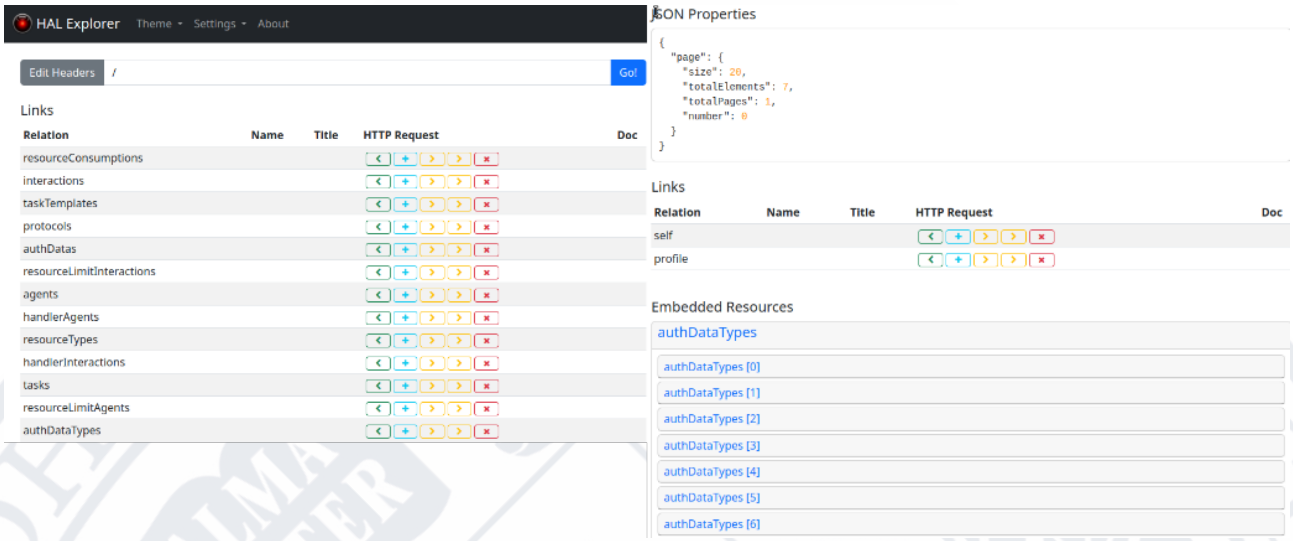

Рисунок 3.4 — Приклад роботи з Configurer сервісом

Робота сервісу Estimator повністю базується на залежностях, що були виведені в розділі 2. Як зазначено на рисунку 3.2, Estimator відповідає за обчислення параметрів цифрових двійників на етапах, що передують етапу розгортання. Як наслідок, він відноситься до середовища метаданих. З точки зору реалізації, Estimator складається з репозиторіїв, що забезпечують доступ для читання необхідних метаданих та об'єктів, що відповідають безпосередньо за обчислення оцінок сутностей. В загальному випадку, його роботу можна розбити на такі кроки:

1. визначення взаємодій в яких може приймати участь кожен агент;
2. оцінка використання кожного типу ресурсу агентами;
3. оцінка використання кожного типу ресурсу взаємодіями;
4. оцінка використання кожного типу ресурсу мульти-агентною системою в цілому;
5. постановка у відповідність кожному агенту єдиної множини взаємодій.

Дана задача є варіантом задачі пакування рюкзака [55, 56]. Задача пакування рюкзака у формі проблеми вибору є NP-повною, тому в загальному випадку не існує алгоритму як правильного, так і швидкого (який виконується за поліноміальний час). Існують алгоритми [56, 57, 58], які дозволяють знайти

рішення, наближене до оптимального, але не обов'язково оптимальне. Як розглядалося у попередньому розділі, ефективність використання локального та глобального оптимумів може досить сильно відрізнятися. Тому доцільним є використання підходу, що дозволить знайти саме глобальний максимум.

З іншої сторони, складність алгоритму повного перебору прямо пропорційна добутку кількості таких сутностей як агент, задача, шаблон задачі та комунікація. Це значення не є досить великим, щоб виникли проблеми з очікуванням відпрацювання такого підходу.

На кожній ітерації необхідно проводити оцінку множини об'єктів, причому деякі з них, можливо, не змінили свій стан. Як результат, отримана оцінка може бути незмінною протягом кількох ітерацій. Виникає можливість оптимізації швидкодії, шляхом застосування кешування для цих значень.

Відповідно до обраного алгоритму та наведених вище проміжних кроків, sequence-діаграма роботи Estimator сервісу матиме вигляд зображений на рисунку 3.5.

Deployer сервіс відповідає за розгортання цифрових двійників, а отже повинен володіти усією необхідною інформацією, що необхідна для їх створення та подальшої роботи. Кроки його роботи для кожного цифрового двійника будуть наступні:

1. Визначити множину обробників корисних задач.
2. Визначити множину обробників задач аутентифікації.
3. Визначити множину статичних елементів даних.
4. Визначити множину обробників динамічних елементів даних.
5. Створити шаблон цифрового двійника готового до розгортання.

Спосіб визначення елементів, зазначених у пунктах 1-4, було приведено у розділі 2. У пункті 5 проводиться створення шаблону, що може бути

використаний Executor сервісом для старту функціонування цифрового двійника.

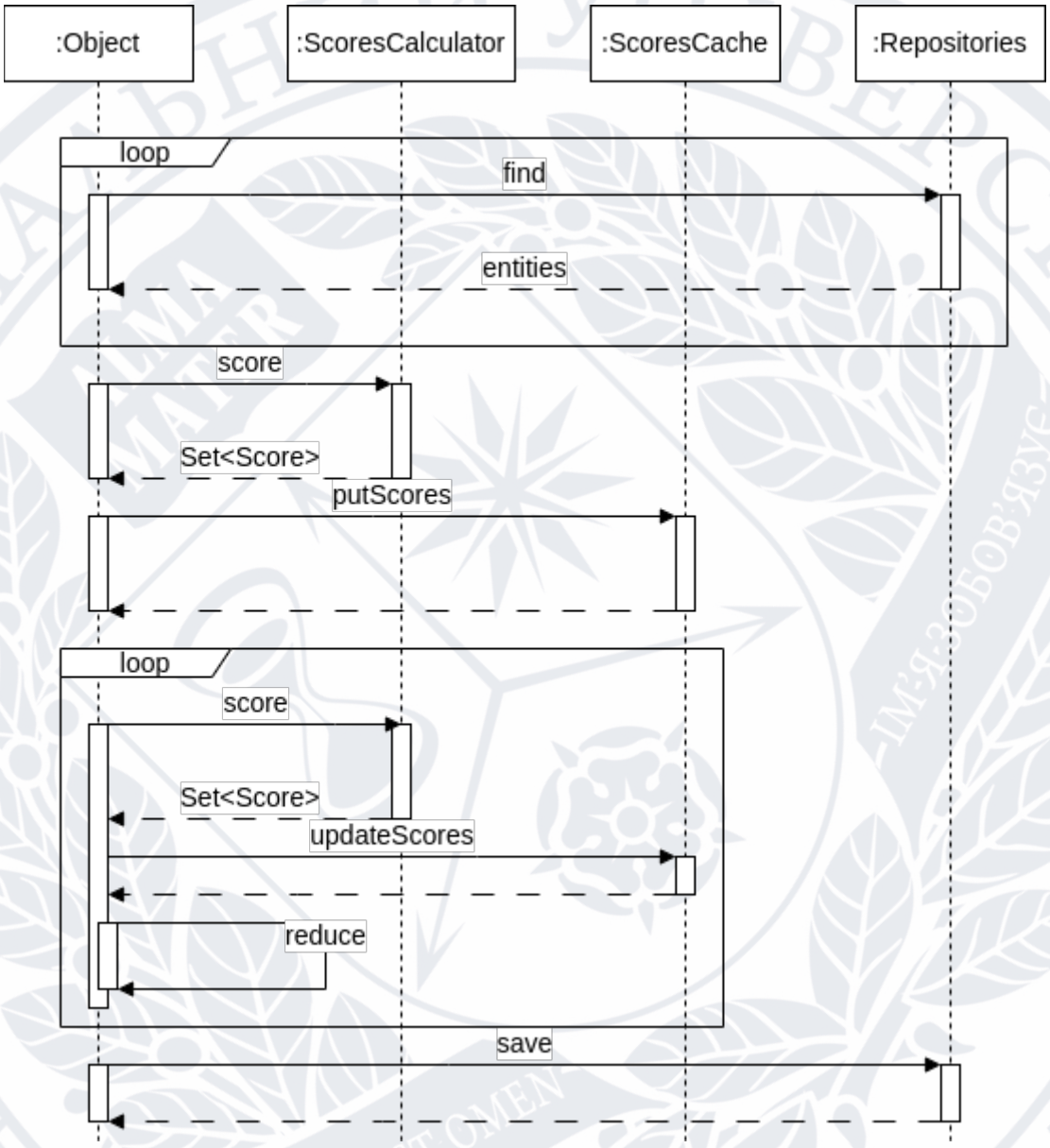

Рисунок 3.5 — Sequence-діаграма процесу роботи Estimator сервісу

Executor сервіс є безпосередньо реалізацією поведінки цифрових двійників. Для функціонування цифрового двійника необхідна наявність середовища виконання, обробників задач, обробників задач аутентифікації. Усі наведені елементи забезпечуються Deployer сервісом. Тому, по суті, Executor є інкапсуляцією середовища цифрового двійника в рамках середовища мульти-агентної системи.

Фрагмент результату роботи Estimator сервісу зображено на рисунку 3.6. Як видно з рисунку, кроки його роботи відповідають наведеним вище, спочатку відбувається відсіювання варіантів, оцінка яких перевищує максимально можливу. Далі відбувається вибір рішення з мінімальною оцінкою та делегація управління на Deployer сервіс.

```
19:29:16.341 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (69.61 > 7.30), agent: 6, resourceType: 3
19:29:16.436 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (45.86 > 41.87), agent: 3, resourceType: 1
19:29:16.466 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (60.26 > 44.46), agent: 0, resourceType: 1
19:29:16.523 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (25.99 > 16.14), agent: 4, resourceType: 0
19:29:16.538 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (90.91 > 0.74), agent: 4, resourceType: 5
19:29:16.570 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (89.88 > 72.99), agent: 4, resourceType: 4
19:29:16.631 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (94.06 > 65.03), agent: 6, resourceType: 2
19:29:16.697 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (59.22 > 7.80), agent: 5, resourceType: 3
19:29:16.792 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (91.32 > 62.90), agent: 2, resourceType: 0
19:29:16.882 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- resource limit exceeded (81.00 > 59.51), agent: 5, resourceType: 3
19:29:16.939 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- 12 configs met resource limit requirements
19:29:16.939 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- sent deployment request, agent: 2, taskTemplate: 1, interactions: 2
19:29:16.940 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- sent deployment request, agent: 0, taskTemplate: 0, interactions: 4
19:29:16.940 [main] INFO ua.edu.donnu.digitaltwinsauth.estimator.EstimationListener -- sent deployment request, agent: 1, taskTemplate: 2, interactions: 2
```

Рисунок 3.6 — Фрагмент результату роботи Estimator сервісу

В цьому розділі було створено програмне забезпечення, що дозволяє виконати задачу багатоагентної аутентифікації цифрових двійників в кібер-фізичних системах. Відповідно до результатів, отриманих в процесі формалізації задачі, було розроблено структуру бази даних для забезпечення можливості збереження, отримання та зміни метаданих, що описують мульти-агентну систему, частиною яких є цифрові двійники, що створюються. З огляду на кроки, які необхідно виконати, для реалізації життєвого циклу цифрового двійника, було визначено групи елементів функціоналу, що необхідно реалізувати. Для зменшення залежності між компонентами системи, було виділено зони відповідальності частин системи та кожній з них поставлено у відповідність окремий сервіс. Розроблено алгоритми роботи кожного сервісу та визначено спосіб їх комунікації між собою.

ВИСНОВКИ

З ростом популярності цифрових двійників в критичній інфраструктурі, промисловості, інтернеті речей, надзвичайно важливим стає забезпечення їх безпеки та цілісності, адже цифрові двійники збирають, відстежують і аналізують дані реального світу, які часто є конфіденційними. Аутентифікація відіграє ключову роль, забезпечуючи цілісність даних та підвищуючи стійкість до кіберзагроз.

В ході даної роботи було проаналізовано існуючі рішення щодо способів реалізації аутентифікації як для окремих об'єктів, так і її організації у мульти-агентних системах, учасниками яких можуть бути цифрові двійники. Це дало змогу виділити різноманітні аспекти процесу аутентифікації та їх характеристики. Отриманні результати було використано для формалізації задачі та виведення законів існування процесів пов'язаних з аутентифікацією на протязі усього життєвого циклу цифрового двійника.

Використовуючи виведені залежності, було створено програмне забезпечення, що надає можливість описувати мульти-агенту систему та, використовуючи отриману мета-модель, автоматизувати процес організації аутентифікації для множини цифрових двійників від моменту їх створення та розгортання до безпосередньо функціонування, як елементів кібер-фізичної системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. WOOLDRIDGE Michael, "the study of information interaction", 1995
2. WEISS Gerhard, "multi-agent systems: a modern approach to distributed artificial intelligence", 1999
3. JENNINGS Nick, WOOLDRIDGE Michael, "foundations of multi-agent systems", 1998
4. WOOLDRIDGE Michael, An Introduction to MultiAgent Systems, John Wiley & Sons Ltd, 2002, paperback, 366 pages, ISBN 0-471-49691-X.
5. Liviu Panait, Sean Luke: Cooperative Multi-Agent Learning: The State of the Art. Autonomous Agents and Multi-Agent Systems 11(3): 387-434 (2005)
6. S. Bhatt, T. K. Pham, M. Gupta, J. Benson, J. Park and R. Sandhu, "Attribute-Based Access Control for AWS Internet of Things and Secure Industries of the Future," in *IEEE Access*, vol. 9, pp. 107200-107223, 2021, doi: 10.1109/ACCESS.2021.3101218.
7. BARKADEHI, Mohammadreza Hazhirpasand, et al. Authentication systems: A literature review and classification. *Telematics and Informatics*, 2018, 35.5: 1491-1511.
8. WOO, Thomas YC; LAM, Simon S. A lesson on authentication protocol design. *ACM SIGOPS Operating Systems Review*, 1994, 28.3: 24-37.
9. SAADEH, Maha, et al. Authentication techniques for the internet of things: A survey. In: *2016 cybersecurity and cyberforensics conference (CCC)*. IEEE, 2016. p. 28-34.
10. TAO, Fei, et al. Digital twin-driven product design framework. *International Journal of Production Research*, 2019, 57.12: 3935-3953.
11. TAO, Fei, et al. Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, 2018, 94: 3563-3576.

12. SEMERARO, Concetta, et al. Digital twin paradigm: A systematic literature review. *Computers in Industry*, 2021, 130: 103469.
13. HAAG, Sebastian; ANDERL, Reiner. Digital twin–Proof of concept. *Manufacturing letters*, 2018, 15: 64-66.
14. VANDERHORN, Eric; MAHADEVAN, Sankaran. Digital Twin: Generalization, characterization and implementation. *Decision support systems*, 2021, 145: 113524.
15. ERRANDONEA, Itxaro; BELTRÁN, Sergio; ARRIZABALAGA, Saioa. Digital Twin for maintenance: A literature review. *Computers in Industry*, 2020, 123: 103316.
16. OMETOV, Aleksandr, et al. Multi-factor authentication: A survey. Cryptography, 2018, 2.1: 1.
17. VELÁSQUEZ, Ignacio; CARO, Angélica; RODRÍGUEZ, Alfonso. Authentication schemes and methods: A systematic literature review. Information and Software Technology, 2018, 94: 30-37.
18. IDRUS, Syed Zulkarnain Syed, et al. A review on authentication methods. Australian Journal of Basic and Applied Sciences, 2013, 7.5: 95-107.
19. IBROKHIMOV, Sanjar, et al. Multi-factor authentication in cyber physical system: A state of art survey. In: *2019 21st international conference on advanced communication technology (ICACT)*. IEEE, 2019. p. 279-284.
20. SHAH, Syed W.; KANHERE, Salil S. Recent trends in user authentication–a survey. *IEEE Access*, 2019, 7: 112505-112519.
21. ALQAHTANI, Ali Abdullah S.; EL-AWADI, Zakaria; MIN, Manki. A survey on user authentication factors. In: *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2021. p. 0323-0328.
22. LAL, Nilesh A.; PRASAD, Salendra; FARIK, Mohammed. A review of authentication methods. *International journal of scientific & technology research*, 2016, 5.11: 246-249.

23. DUNCAN, Richard. An overview of different authentication methods and protocols. *SANS Institute*, 2001.
24. KATSINI, Christina, et al. Security and usability in knowledge-based user authentication: A review. In: Proceedings of the 20th Pan-Hellenic conference on informatics. 2016. p. 1-6.
25. LAMPORT, Leslie. Password authentication with insecure communication. *Communications of the ACM*, 1981, 24.11: 770-772.
26. GERLITZ, Eva, et al. Evolution of Password Expiry in Companies: Measuring the Adoption of Recommendations by the German Federal Office for Information Security. In: *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*. 2023. p. 191-210.
27. NAIR, Vivek; SONG, Dawn. MFDPG: Multi-Factor Authenticated Password Management With Zero Stored Secrets. *arXiv preprint arXiv:2306.14746*, 2023.
28. HALLER, Neil, et al. *A one-time password system*. 1998.
29. HALLER, Neil. *The S/KEY one-time password system*. 1995.
30. ERDEM, Emir; SANDIKKAYA, Mehmet Tahir. OTPaaS—One time password as a service. *IEEE Transactions on Information Forensics and Security*, 2018, 14.3: 743-756.
31. ARAVINDHAN, K.; KARTHIGA, R. R. One time password: A survey. *International Journal of Emerging Trends in Engineering and Development*, 2013, 1.3: 613-623.
32. MERKLE, Ralph Charles. *Secrecy, authentication, and public key systems*. Stanford university, 1979.
33. HALEVI, Shai; KRAWCZYK, Hugo. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security (TISSEC)*, 1999, 2.3: 230-268.
34. SIRBU, Marvin A.; CHUANG, JC-I. Distributed authentication in Kerberos using public key cryptography. In: *Proceedings of SNDSS'97: Internet Society*

1997 Symposium on Network and Distributed System Security. IEEE, 1997. p. 134-141.

35. PETERSENL, Holger; HORSTER, Patrick. Self-certified keys—concepts and applications. *Communications and Multimedia Security: Volume 3*, 1997, 102-116.
36. ZHU, Wen-Tao; LIN, Jingqiang. Generating correlated digital certificates: framework and applications. *IEEE Transactions on Information Forensics and Security*, 2016, 11.6: 1117-1127.
37. KAPPENBERGER, Reiner. The true cost of self-signed SSL certificates. *Computer Fraud & Security*, 2012, 2012.9: 14-16.
38. CHU, YeonSung, et al. SS-DPKI: Self-signed certificate based decentralized public key infrastructure for secure communication. In: *2020 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2020. p. 1-6.
39. CHEN, Zhigang, et al. A Survey on Zero-Knowledge Authentication for Internet of Things. *Electronics*, 2023, 12.5: 1145.
40. CHI, Po-Wen; LU, Yun-Hsiu; GUAN, Albert. A Privacy-Preserving Zero-Knowledge Proof for Blockchain. *IEEE Access*, 2023.
41. LI, Wanxin, et al. Privacy-preserving traffic management: A blockchain and zero-knowledge proof inspired approach. *IEEE access*, 2020, 8: 181733-181743.
42. DIEYE, Mohameden, et al. A Self-Sovereign Identity based on Zero-Knowledge Proof and blockchain. *IEEE Access*, 2023.
43. LIU, Wenzheng; WANG, Xiaofeng; PENG, Wei. Secure remote multi-factor authentication scheme based on chaotic map zero-knowledge proof for crowdsourcing internet of things. *IEEE Access*, 2019, 8: 8754-8767.
44. SINGH, Saroj. Multi-factor authentication and their approaches. *International Research Journal of Management, IT and Social Sciences*, 2017, 4.3: 68-81.

45. YOHAN, Alexander; LO, Nai-Wei; LIE, Henry Roes. Dynamic multi-factor authentication for smartphone. In: *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, 2016. p. 1-6.
46. IWUOHA, Obioha; EMMANUEL, Nwabueze; EKWONWUNE, Emmanuel. Enhancing Multi-Factor Authentication in Modern Computing. 2017.
47. ZHAO, Shushan; HU, Wenhui. Improvement on OTP authentication and a possession-based authentication framework. *International Journal of Multimedia Intelligence and Security*, 2018, 3.2: 187-203.
48. PANSA, Detchasit; CHOMSIRI, Thawatchai. Integrating the dynamic password authentication with possession factor and captcha. In: *2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS)*. IEEE, 2018. p. 530-535.
49. M'RAIHI, David, et al. RFC 6238: TOTP: Time-Based One-Time Password Algorithm. 2011.
50. M'RAIHI, David, et al. *Totp: Time-based one-time password algorithm*. 2011.
51. PARK, Woo-Suk; HWANG, Dong-Yeop; KIM, Ki-Hyung. A TOTP-based two factor authentication scheme for hyperledger fabric blockchain. In: *2018 Tenth international conference on ubiquitous and future networks (ICUFN)*. IEEE, 2018. p. 817-819.
52. CUNHA, Vitor A., et al. TOTP Moving Target Defense for sensitive network services. *Pervasive and Mobile Computing*, 2021, 74: 101412.
53. MASSEY, Steven, et al. Trusted Working Copies for Distributed Systems Engineering. In: *AIAA SCITECH 2022 Forum*. 2022. p. 0093.
54. LIU, Mengnan, et al. Review of digital twin about concepts, technologies, and industrial applications. Journal of Manufacturing Systems, 2021, 58: 346-361.
55. SALKIN, Harvey M.; DE KLUYVER, Cornelis A. The knapsack problem: a survey. *Naval Research Logistics Quarterly*, 1975, 22.1: 127-144.

56. MARTELLO, Silvano; TOTH, Paolo. Algorithms for knapsack problems. *North-Holland Mathematics Studies*, 1987, 132: 213-257.

57. PUCHINGER, Jakob; RAIDL, Günther R.; PFERSCHY, Ulrich. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 2010, 22.2: 250-265.

58. DUDZIŃSKI, Krzysztof; WALUKIEWICZ, Stanisław. Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research*, 1987, 28.1: 3-21.

ДОДАТОК А

Лістинг програми

```
package ua.edu.donnu.digitaltwinsauth.domain;
import java.util.*;
import jakarta.annotation.*;
import jakarta.persistence.*;
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.estimator.*;
@Data
@NoArgsConstructor
@Entity
public class Agent implements Scorable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    private String name;
    @Nonnull
    private boolean twin;
    @ManyToMany(mappedBy = "agents")
    private List<Task> tasks;
    @ManyToOne(optional = false)
    private List<HandlerAgent> handlers;
    @Override
    public Double score(ResourceType resourceType, Estimator estimator) {
        return estimator.score(resourceType, this);
    }
}
```

```
package ua.edu.donnu.digitaltwinsauth.domain;
import jakarta.persistence.*;
import lombok.*;
@Data
@NoArgsConstructor
@Entity
public class AuthData {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @ManyToOne(optional = false)
    private AuthDataType type;

    private String name;

    private String value;

    @ManyToOne
    @JoinColumn(name = "protocol_id", nullable = false)
    private Protocol protocol;
}
```

```
package ua.edu.donnu.digitaltwinsauth.domain;
import jakarta.annotation.*;
import jakarta.persistence.*;
import lombok.*;
@Data
@NoArgsConstructor
```

```
@Entity
public class AuthDataType {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private long id;
        @Nonnull
        private String name;
        private Boolean dynamic;
}
```

```
package ua.edu.donnu.digitaltwinsauth.domain;
import jakarta.persistence.*;
import lombok.*;
@Data
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class BaseHandler {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private long id;

   private String reference;

   private Long orderId;
}
```

```
package ua.edu.donnu.digitaltwinsauth.domain;
import java.util.*;
import jakarta.persistence.*;
```

```
import lombok.*;
@Data
@EqualsAndHashCode(callSuper = true)
@Entity
public class HandlerAgent extends BaseHandler {
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "agent_id", referencedColumnName = "id")
    private Agent agent;
    @ManyToMany(mappedBy = "handlers")
    private List<ResourceConsumption> resourceConsumptions;
}

package ua.edu.donnu.digitaltwinsauth.domain;
import java.util.*;
import jakarta.persistence.*;
import lombok.*;
@Data
@EqualsAndHashCode(callSuper = true)
@Entity
public class HandlerInteraction extends BaseHandler {
        @OneToOne(cascade = CascadeType.ALL)
        @JoinColumn(name = "interaction_id", referencedColumnName = "id")
        private Interaction interaction;
        @ManyToMany(mappedBy = "handlers")
        private List<ResourceConsumption> resourceConsumptions;
}

package ua.edu.donnu.digitaltwinsauth.domain;
import java.util.*;
```

```
import jakarta.persistence.*;
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.estimator.*;
@Data
@NoArgsConstructor
@Entity
public class Interaction implements Scorable {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    @ManyToMany(mappedBy = "interactions")
    private List<TaskTemplate> taskTemplates;
    @ManyToMany
    @JoinTable(name = "interation_protocol", joinColumns = @JoinColumn(name =
"protocol_id"), inverseJoinColumns = @JoinColumn(name = "interaction_id"))
    private List<Protocol> protocols;
    @ManyToOne(optional = false)
    private List<HandlerInteraction> handlers;
    @Override
    public Double score(ResourceType resourceType, Estimator estimator) {
        return estimator.score(resourceType, this);
    }
}

package ua.edu.donnu.digitaltwinsauth.domain;
import java.util.*;
import jakarta.persistence.*;
import lombok.*;
@Data
```

```
@NoArgsConstructor
@Entity
public class Protocol {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    @OneToMany(mappedBy = "protocol")
    private List<AuthData> authDatas;
}

package ua.edu.donnu.digitaltwinsauth.domain;
import java.util.*;
import jakarta.persistence.*;
import lombok.*;
@Data
@NoArgsConstructor
@Entity
public class ResourceConsumption {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    private Double value;
    @ManyToOne(optional = false)
    private ResourceType resourceType;
    @ManyToMany(mappedBy = "resourceConsumptions")
    private List<BaseHandler> handlers;
}

package ua.edu.donnu.digitaltwinsauth.domain;
```

```
import jakarta.persistence.*;
import lombok.*;
@Data
@Entity
public class ResourceLimitAgent {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private long id;
	private Double value;
	@OneToOne(cascade = CascadeType.ALL)
	@JoinColumn(name = "resource_type_id", referencedColumnName = "id")
	private ResourceType resourceType;
	@OneToOne(cascade = CascadeType.ALL)
	@JoinColumn(name = "agent_id", referencedColumnName = "id")
	private Agent agent;
}

package ua.edu.donnu.digitaltwinsauth.domain;
import jakarta.persistence.*;
import lombok.*;
@Data
@Entity
public class ResourceLimitInteraction {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private long id;
	private Double value;
	@OneToOne(cascade = CascadeType.ALL)
	@JoinColumn(name = "resource_type_id", referencedColumnName = "id")
```

```
        private ResourceType resourceType;
        @OneToOne(cascade = CascadeType.ALL)
        @JoinColumn(name = "interaction_id", referencedColumnName = "id")
        private Interaction interaction;
}

package ua.edu.donnu.digitaltwinsauth.domain;
import jakarta.annotation.*;
import jakarta.persistence.*;
import lombok.*;
@Data
@NoArgsConstructor
@Entity
public class ResourceType {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private long id;
        @Nonnull
        private String name;
}

package ua.edu.donnu.digitaltwinsauth.domain;
import java.util.*;
import jakarta.persistence.*;
import lombok.*;
@Data
@NoArgsConstructor
@Entity
public class Task {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    private String name;
    @ManyToMany
    @JoinTable(name = "agent_task", joinColumns = @JoinColumn(name =
"agent_id"), inverseJoinColumns = @JoinColumn(name = "task_id"))
    private List<Agent> agents;
    @OneToMany(mappedBy = "task")
    private List<TaskTemplate> taskTemplates;
}

package ua.edu.donnu.digitaltwinsauth.domain;
import java.util.*;
import jakarta.persistence.*;
import lombok.*;
@Data
@NoArgsConstructor
@Entity
public class TaskTemplate {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    @ManyToOne
    @JoinColumn(name = "task_id", nullable = false)
    private Task task;
    @ManyToMany
```

```
    @JoinTable(name = "task_template_interation", joinColumns =
@JoinColumn(name = "interaction_id"), inverseJoinColumns = @JoinColumn(name
= "task_template_id"))
    private List<Interaction> interactions;
}

package ua.edu.donnu.digitaltwinsauth.configurer;
import java.util.*;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface AgentRepository extends JpaRepository<Agent, Long> {
  List<Agent> findByTwin(boolean twin);
}

package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface AuthDataRepository extends JpaRepository<AuthData, Long> {
}

package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
```

```
public interface AuthDataTypeRepository extends JpaRepository<AuthDataType,
Long> {
}

package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface HandlerAgentRepository extends JpaRepository<HandlerAgent,
Long> {
}

package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface HandlerInteractionRepository extends
JpaRepository<HandlerInteraction, Long> {
}

package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface InteractionRepository extends JpaRepository<Interaction, Long> {
}
```

```
package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface ProtocolRepository extends JpaRepository<Protocol, Long> {
}
```

```
package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface ResourceConsumptionRepository extends
JpaRepository<ResourceConsumption, Long> {
}
```

```
package ua.edu.donnu.digitaltwinsauth.configurer;
import java.util.*;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface ResourceLimitAgentRepository extends
JpaRepository<ResourceLimitAgent, Long> {
        List<ResourceLimitAgent> findByAgent(Agent agent);
}
```

```
package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface ResourceLimitInteractionRepository extends
JpaRepository<ResourceLimitInteraction, Long> {
}

package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface ResourceTypeRepository extends JpaRepository<ResourceType,
Long> {
}

package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface TaskRepository extends JpaRepository<Task, Long> {
}

package ua.edu.donnu.digitaltwinsauth.configurer;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.rest.core.annotation.*;
```

```
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RepositoryRestResource
public interface TaskTemplateRepository extends JpaRepository<TaskTemplate,
Long> {
}

package ua.edu.donnu.digitaltwinsauth.deployer;
import java.util.*;
import java.util.stream.*;
import org.springframework.context.*;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.*;
import lombok.*;
import lombok.extern.slf4j.*;
import ua.edu.donnu.digitaltwinsauth.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@Slf4j
@Service
@RequiredArgsConstructor
public class DeploymentListener {
   private final ApplicationEventPublisher eventPublisher;
       @EventListener
       public void onMessage(DeploymentRequestDto dto) {
          List<AuthData> authDatas = dto.getInteractions().stream()
             .map(in -> in.getProtocols())
             .flatMap(List::stream)
             .map(Protocol::getAuthDatas)
             .flatMap(List::stream)
             .collect(Collectors.toList());
```

```
            eventPublisher.publishEvent(new ExecutionRequestDto(dto.getAgent(),
dto.getInteractions(), authDatas));
        }
}

package ua.edu.donnu.digitaltwinsauth.controllers;
import org.springframework.beans.factory.annotation.*;
import org.springframework.context.*;
import org.springframework.web.bind.annotation.*;
import ua.edu.donnu.digitaltwinsauth.*;
@RequestMapping("/digitaltwin")
@RestController
public class DigitalTwinController {
   @Autowired
   private ApplicationEventPublisher eventPublisher;
   @PostMapping("/")
   public void create(ExecutionRequestDto dto) {
      eventPublisher.publishEvent(dto);
   }
}

package ua.edu.donnu.digitaltwinsauth.executor;
public interface DigitalTwin extends Runnable {
}

package ua.edu.donnu.digitaltwinsauth.executor;
import java.util.*;
import java.util.function.*;
import lombok.*;
```

```
import lombok.extern.slf4j.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@Slf4j
@RequiredArgsConstructor
public class DigitalTwinImpl implements DigitalTwin {
    private final Agent agent;
    private final List<Interaction> interactions;
    private final List<AuthData> authDatas;
    @Override
    public void run() {
        Thread.currentThread().setName(String.format("agent#%d", agent.getId()));
        authDatas.forEach(ad -> ExecutionContextHolder.get().add(ad));
        interactions.forEach(this::process);
    }
    private void process(Interaction interaction) {
        List<BaseHandler> handlers = new ArrayList<>();
        handlers.addAll(agent.getHandlers());
        handlers.addAll(interaction.getHandlers());
        handlers.sort(new Comparator<BaseHandler>() {
            @Override
            public int compare(BaseHandler o1, BaseHandler o2) {
                return Long.compare(o1.getOrderId(), o2.getOrderId());
            }
        });
        for (BaseHandler handler : handlers) {
            try {
                Class<?> handlerClass = Class.forName(handler.getReference());
                Consumer<ExecutionContext> consumer =
(Consumer<ExecutionContext>) handlerClass.newInstance();
```

```
                consumer.accept(ExecutionContextHolder.get());
            } catch (ClassNotFoundException | InstantiationException |
IllegalAccessException e) {
                log.warn("can't execute handler {}", handler.getId());
            }
        }
    }
}
```

```
package ua.edu.donnu.digitaltwinsauth.executor;
import java.util.*;
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@RequiredArgsConstructor
public class ExecutionContext {
    private final List<AuthData> params;
    public AuthData get(String name) {
        return params.stream().filter(ad -> name.equals(ad.getName())).findFirst().get();
    }
    public void add(AuthData authData) {
        params.add(authData);
    }
}
```

```
package ua.edu.donnu.digitaltwinsauth.executor;
public final class ExecutionContextHolder {
    private static ThreadLocal<ExecutionContext> context;
    public static ExecutionContext get() {
        return context.get();
```

```
    }
    public static void set(ExecutionContext context) {
        ExecutionContextHolder.context.set(context);
    }
}
```

```
package ua.edu.donnu.digitaltwinsauth.executor;
import java.util.concurrent.*;
import org.springframework.context.event.*;
import org.springframework.stereotype.*;
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.*;
@Service
@RequiredArgsConstructor
public class ExecutionListener {
        private final ForkJoinPool pool;
        @EventListener
        public void onMessage(ExecutionRequestDto dto) {
                DigitalTwin digitalTwin = new DigitalTwinImpl(dto.getAgent(),
dto.getInteractions(), dto.getAuthDatas());
                pool.execute(digitalTwin);
        }
}
```

```
package ua.edu.donnu.digitaltwinsauth.executor;
public interface Handler {
    void process(ExecutionContext executionContext);
}
```

```
package ua.edu.donnu.digitaltwinsauth.estimator;
import java.util.*;
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@Getter
@RequiredArgsConstructor
public class AgentConfig implements Scorable {
   private final Agent agent;
   private final List<Interaction> interactions;
   @Override
   public Double score(ResourceType resourceType, Estimator estimator) {
      double score1 = interactions.stream().mapToDouble(i -> i.score(resourceType,
estimator)).sum();
      double score2 = agent.score(resourceType, estimator);
      return score1 + score2;
   }
}

package ua.edu.donnu.digitaltwinsauth.estimator;
import java.util.*;
import java.util.stream.*;
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@Getter
@RequiredArgsConstructor
public class SystemConfig implements Scorable {
        private final List<AgentConfig> agents;
        public Set<Interaction> interactions() {
```

```
            return
agents.stream().map(AgentConfig::getInteractions).flatMap(List::stream).collect(Coll
ectors.toSet());
        }
        @Override
        public Double score(ResourceType resourceType, Estimator estimator) {
            return agents.stream().mapToDouble(agentConfig ->
agentConfig.score(resourceType, estimator)).sum();
        }
}

package ua.edu.donnu.digitaltwinsauth.estimator;
import ua.edu.donnu.digitaltwinsauth.domain.*;
public interface Estimator {
   Double score(ResourceType resourceType, Interaction interaction);
   Double score(ResourceType resourceType, Agent agent);
}

package ua.edu.donnu.digitaltwinsauth.estimator;
import java.util.*;
import org.springframework.beans.factory.annotation.*;
import org.springframework.context.annotation.*;
import org.springframework.stereotype.*;
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@Primary
@Service
@Qualifier("cachedEstimator")
@RequiredArgsConstructor
```

```
public class EstimatorCached implements Estimator {
    @Qualifier("estimator")
    private final Estimator delegate;
    private Map<Scorable, Double> cached = new HashMap<>();
    @Override
    public Double score(ResourceType resourceType, Interaction interaction) {
        Double score = getScore(resourceType, interaction);
        if (score != null) {
            return score;
        }
        return delegate.score(resourceType, interaction);
    }
    @Override
    public Double score(ResourceType resourceType, Agent agent) {
        Double score = getScore(resourceType, agent);
        if (score != null) {
            return score;
        }
        return delegate.score(resourceType, agent);
    }
    private Double getScore(ResourceType resourceType, Scorable scorable) {
        return cached.getOrDefault(scorable, null);
    }
}

package ua.edu.donnu.digitaltwinsauth.estimator;
import java.util.*;
import org.springframework.beans.factory.annotation.*;
import org.springframework.stereotype.*;
```

```
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@Service
@Qualifier("estimator")
@RequiredArgsConstructor
public class EstimatorImpl implements Estimator {
    @Override
    public Double score(ResourceType resourceType, Interaction interaction) {
        return interaction.getHandlers().stream()
                .map(h -> h.getResourceConsumptions())
                .mapToDouble(rc -> getScore(rc, resourceType))
                .sum();
    }
    @Override
    public Double score(ResourceType resourceType, Agent agent) {
        return agent.getHandlers().stream()
            .map(h -> h.getResourceConsumptions())
            .mapToDouble(rc -> getScore(rc, resourceType))
            .sum();
    }
    private double getScore(List<ResourceConsumption> resourceConsumptions,
ResourceType resourceType) {
        return resourceConsumptions.stream()
                .filter(rc -> rc.getResourceType().getId() == resourceType.getId())
                .mapToDouble(rc -> rc.getValue())
                .sum();
    }
}
```

```
package ua.edu.donnu.digitaltwinsauth.estimator;
import ua.edu.donnu.digitaltwinsauth.domain.*;
public interface Scorable {
    Double score(ResourceType resourceType, Estimator estimator);
}

package ua.edu.donnu.digitaltwinsauth.estimator;
import java.util.*;
import java.util.concurrent.atomic.*;
import java.util.stream.*;
import org.springframework.context.*;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.*;
import lombok.*;
import lombok.extern.slf4j.*;
import ua.edu.donnu.digitaltwinsauth.*;
import ua.edu.donnu.digitaltwinsauth.configurer.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@Slf4j
@Service
@RequiredArgsConstructor
public class EstimationListener {
    private final ApplicationEventPublisher eventPublisher;
    private final Estimator estimator;
    private final AgentRepository agentRepository;
    private final ResourceLimitAgentRepository resourceLimitAgentRepository;
    private final ResourceTypeRepository resourceTypeRepository;
    @EventListener
    public void onMessage(EstimationRequestDto dto) {
```

```
        List<Agent> digitalTwins = agentRepository.findByTwin(true);
        Map<Agent, List<AgentConfig>> initialSetup =
createInitialSetup(digitalTwins);
        List<SystemConfig> possibleConfigs = generatePermutations(initialSetup);
        Iterator<SystemConfig> iterator = possibleConfigs.iterator();
        while (iterator.hasNext()) {
            SystemConfig systemConfig = iterator.next();
            if (systemConfig.getAgents().stream().anyMatch(this::hasLimitExceed)) {
                iterator.remove();
            }
        }
        SortedSet<SystemConfig> sortedConfigs = new TreeSet<>(new
SystemConfigComparator());
        possibleConfigs.forEach(sortedConfigs::add);
        log.info("{} configs met limit requirement", sortedConfigs.size());
        sortedConfigs.first().getAgents().forEach(config -> {
            DeploymentRequestDto event = new
DeploymentRequestDto(config.getAgent(), config.getInteractions());
            eventPublisher.publishEvent(event);
            log.info("sent deployment request, agent: {}, interactions: {}",
config.getAgent().getId(), config.getInteractions().size());
        });
    }
    private Map<Agent, List<AgentConfig>> createInitialSetup(List<Agent>
digitalTwins) {
        Map<Agent, List<AgentConfig>> result = new HashMap<>();
        for (Agent agent : digitalTwins) {
            result.putIfAbsent(agent, new ArrayList<>());
            List<Interaction> agentInteractions = new ArrayList<>();
```

```
            for (Task task : agent.getTasks()) {
                for (TaskTemplate taskTemplate : task.getTaskTemplates()) {
                    agentInteractions.addAll(taskTemplate.getInteractions());
                }
            }
            result.get(agent).add(new AgentConfig(agent, agentInteractions));
        }
        return result;
    }
    private List<SystemConfig> generatePermutations(Map<Agent,
List<AgentConfig>> initialSetup) {
        List<SystemConfig> result = new ArrayList<>();
        Map<Agent, AtomicInteger> offsets = new HashMap<>();
        initialSetup.keySet().forEach(agent -> offsets.put(agent, new AtomicInteger()));
        for (Map.Entry<Agent, List<AgentConfig>> entry : initialSetup.entrySet()) {
            SystemConfig systemConfig = new SystemConfig(new ArrayList<>());
            Agent agent = entry.getKey();
            List<AgentConfig> configs = entry.getValue();
            for (AgentConfig agentConfig : configs) {
                systemConfig.getAgents().add(agentConfig);
                for (Map.Entry<Agent, AtomicInteger> offsetEntry : offsets.entrySet()) {
                    Agent nextAgent = offsetEntry.getKey();
                    AtomicInteger offset = offsetEntry.getValue();
                    if (nextAgent != agent) {
                        AgentConfig nextConfig =
initialSetup.get(nextAgent).get(offset.incrementAndGet());
                        systemConfig.getAgents().add(nextConfig);
                    }
                }
```

```
            }
        }
        return result;
    }
    private boolean hasLimitExceed(AgentConfig config) {
        Map<ResourceType, Double> limits =
resourceLimitAgentRepository.findByAgent(config.getAgent()).stream()
                .collect(Collectors.toMap(ResourceLimitAgent::getResourceType,
ResourceLimitAgent::getValue));
        for (ResourceType resourceType : resourceTypeRepository.findAll()) {
            Double estimated = config.score(resourceType, estimator);
            Double limit = limits.get(resourceType);
            if (estimated > limit) {
                log.info("resource limit exceeded ({} > {}), agent: {}, resourceType: {}",
estimated, limit,
                        config.getAgent().getId(), resourceType.getId());
                return true;
            }
        }
        return false;
    }
}
```

```
package ua.edu.donnu.digitaltwinsauth;
public class EstimationRequestDto {
}
```

```
package ua.edu.donnu.digitaltwinsauth;
import java.util.*;
```

```
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@Getter
@RequiredArgsConstructor
public class DeploymentRequestDto {
        private final Agent agent;
        private final List<Interaction> interactions;
}

package ua.edu.donnu.digitaltwinsauth;
import java.util.*;
import lombok.*;
import ua.edu.donnu.digitaltwinsauth.domain.*;
@Getter
@RequiredArgsConstructor
public class ExecutionRequestDto {
   private final Agent agent;
   private final List<Interaction> interactions;
   private final List<AuthData> authDatas;
}

package ua.edu.donnu.digitaltwinsauth;
import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
@SpringBootApplication
public class DigitalTwinsAuthApplication {
        public static void main(String[] args) {
                SpringApplication.run(DigitalTwinsAuthApplication.class, args);
        }}
```